Learning to Repair Infeasible* Problems with Deep Reinforcement Learning on Graphs

M. Zouitine[1,2], A. Berjaoui[1], A. Lagnoux[2,3], C. Pellegrini[2,3], and E. Rachelson[4]

[1] IRT Saint Exupéry, Toulouse, France
{mehdi.zouitine,ahmad.berjaoui}@irt-saintexupery.com
[2] Institut de Mathématiques de Toulouse; UMR5219. Université de Toulouse; CNRS.
[3] UT2J, F-31058 Toulouse, France, UT3, F-31062 Toulouse, France
{agnes.lagnoux,clement.pellegrini}@math.univ-toulouse.fr
[4] ISAE-SUPAERO, France
emmanuel.rachelson@isae-supaero.fr

Abstract. In the last few years, deep learning has demonstrated significant potential in Operations Research across various tasks. In this work, we tackle the problem of repairing infeasible constraint satisfaction problems by introducing a novel deep reinforcement learning approach. Our method leverages graph deep learning to represent infeasible problems, utilizing a graph representation of Constraint Satisfaction Problems. By employing bipartite graph neural networks to encode the constraints of these problems, we train a deep learning agent to identify and extract a subset of constraints that restores feasibility solely from the reward signal, requiring no labeled data. We evaluate our approach using several bipartite graph neural network architectures and demonstrate its effectiveness in two domains: maximizing feasibility in Linear Programs and maximizing satisfiability in Boolean satisfiability problems. Our results show that the agent is competitive with existing heuristics in both solution quality and computational efficiency across these domains. An open source implementation of our methods is available at https://github.com/MehdiZouitine/Learning_to_repair_infeasible_problems_with_DRL_and_GNN.

Keywords: Infeasibility Analysis · Graph Neural Networks · Learning Based Heuristics · Linear Feasibility · Boolean Satisfiability · Deep Reinforcement Learning

1 Introduction

Constraints are fundamental building blocks of many Operation Research (OR) problems, as they model real-life scenarios. Constraint Satisfaction Problem (CSP) solving algorithms aim to find a solution that satisfies the constraints.

[1] *Throughout this paper, we use the terms "feasibility" and "satisfiability" interchangeably to refer to the same concept.

However, in some cases, a problem may contain inconsistent or incompatible constraints, rendering it infeasible. In such situations, it becomes crucial to analyze which constraints are responsible for this infeasibility. More specifically, an important question to address is "What is the maximum subset of constraints that can be retained to make the problem feasible?" or its complementary question, "What is the minimum subset of constraints that can be removed to achieve feasibility?" This subset is called the maximum feasible subset of constraints and the discovery of such a subset is known to be NP-hard [2]. The answer to these questions can help users understand the main causes of infeasibility and enable efficient repair of the problem. Recently, Machine Learning (ML) has emerged as a powerful tool for discovering new heuristics to tackle NP-hard problems. Approaching these challenges from a statistical learning perspective allows for the rapid development of effective heuristics, reducing the need for extensive domain expertise, and improving the overall efficiency.

Although most ML approaches focus on learning solutions for specific OR problems, we explore how to repair infeasible CSPs and illustrate our method on two families of CSPs. Linear Feasibility Problems (LF) and Boolean Satisfiability Problems (SAT), using graph deep reinforcement learning (RL). More precisely, we present an approach to infeasibility repair using deep RL, expanding the scope of ML for OR. Our method encodes infeasible problems using bipartite Graph Neural Networks (GNNs). This encoding demonstrates robustness and flexibility in varying the size of the problem. By designing a deep RL policy, we eliminate the need for labeled data, enabling the model to learn efficient heuristics. We apply this policy to find the maximum feasible subsets in both LF and SAT problems, achieving superior generalization and efficiency compared to existing heuristics.

The paper is structured as follows. Initially, we review the pertinent literature in operations research, reinforcement learning, and deep learning. We then detail our methodology, explaining how we model CSPs as graphs, outline the Markov Decision Process (MDP), and describe the neural architectures designed to learn heuristics. Lastly, we provide a comprehensive evaluation of our approach in the LF and SAT domains, focusing on its efficiency (in terms of solution quality and computational time), generalization capabilities, and limitations.

2 Background and related work

Constraints Satisfaction Problem (CSP). CSPs [21] are a powerful framework for modeling and solving a wide range of complex problems in artificial intelligence and OR. At their core, CSPs involve finding a set of values for variables that satisfy a given set of constraints. Two notable examples of CSPs are LF and SAT problems. In LF, constraints are expressed in terms of linear combinations of variables. For example, a constraint might take the form $2v_1 + 3v_2 \leq 10$ and the goal is to find values of v_1 and v_2 that satisfy this and other similar constraints. This formulation is particularly useful for problems involving resource allocation, scheduling, and optimization in various industrial settings. In

SAT problems, we deal with Boolean variables, and the constraints are typically expressed in Conjunctive Normal Form (CNF). In CNF, a constraint is a conjunction (AND) of clauses, where each clause is a disjunction (OR) of literals (variables or their negation). For example, a SAT constraint might look like $(v_1 \vee \neg v_2) \wedge (\neg v_1 \vee v_3)$, where v_1, v_2, and v_3 are boolean variables. The goal in SAT problems is to find an assignment of true/false values to these variables that satisfies all clauses simultaneously. SAT solvers are widely used in circuit design, software verification, and AI planning. Both the LF and SAT problems illustrate how CSPs can model diverse real-world scenarios, each with a unique constraint representation, showcasing the versatility and power of the CSP framework in problem solving across various domains.

Infeasibility and unsatisfiability analysis. Investigating infeasibility or unsatisfiability in CSPs often involves finding relevant subsets of constraints or clauses. These include Unsatisfiable Cores [25], Minimal Unsatisfiable Subsets [23], Irreducible Infeasible Subsets [11], and the focus of this paper: the Maximum Feasible Subset (MAXFS) in Linear Programming (LP) and the Maximum Satisfiability Problem (MAXSAT). Being NP-hard problems [1], exact solutions for MAXFS are achievable only for small instances. However, its importance spans numerous real-world applications [2,6,26], which requires efficient heuristics that provide high-quality solutions. Although there are many exact formulations [11,27,28], their practical application is limited due to the NP-hard nature of the problem [11]. Among heuristic approaches, Chinneck's LP based heuristics [9,10] are widely recognized and utilized, having demonstrated superior performance compared to other methods in experimental settings. In the SAT domain, the exact algorithms for MAXSAT are well-established, [7,36]. As in LF, there also exist efficient heuristic solvers like RC2 [17] for MAXSAT, represent the state of the art in this field.

Reinforcement learning. RL [31] considers the problem of learning a decision-making policy for an agent interacting over multiple time steps with a dynamic environment. At each time step, the agent and the environment are described through a state $s \in \mathcal{S}$ and an action $a \in \mathcal{A}$ is performed; then the system transitions to a new state s' according to probability $p(s'|s,a)$, while receiving a reward $r(s,a)$. The tuple $M = (\mathcal{S}, \mathcal{A}, p, r)$ is called a Markov Decision Process (MDP) [29], which is often complemented by the knowledge of an initial state distribution $p_0(s)$. A policy π_θ parameterized by θ is a function $(s,a) \mapsto \pi_\theta(a|s)$ that maps states to distributions conditioned by actions. Training an RL agent consists of finding the policy that maximizes the discounted expected return $J(\pi_\theta) = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)]$, where $\gamma \in [0,1)$ is a discount factor.

Graph Neural Networks. In recent years, GNNs have become powerful tools for various tasks defined in graph structures, using their ability to process graph data through message passing mechanisms [19]. The core idea behind GNNs is to exchange and aggregate information from the neighborhood of nodes and edges, allowing the network to capture complex dependencies inherent in graph-structured data. A particularly relevant type of graph in this context is the bipartite graph, which has been extensively studied in GNN research. Several

architectures have been developed specifically to address problems such as assignment tasks [14,38] or to make predictions related to LP problems, including Mixed Integer Linear Programming (MILP) [13,8]. One of the key strengths of GNNs lies in their flexibility and generality. They can be trained on one set of graphs and applied effectively to different graph structures, demonstrating their robustness across various types of graphs [18].

Learning based heuristics. In recent years, a new set of methods known as learning based heuristics has emerged as a promising paradigm for solving optimization problems from a statistical learning point of view. In a foundational work, [34] introduced the pointer network, a novel architecture designed to solve permutation-based problems such as convex-hull and travelling salesman problems. In this view, [5] used RL to learn heuristics solely from rewards, tackling both traveling salesman and knapsack problems. RL is used to learn constructive heuristics functions that incrementally build solutions step by step, optimizing a cumulative reward based on partial solutions. The efficiency of such constructive heuristics is significantly based on the chosen neural architecture. Finding the right architecture is crucial for the heuristic to effectively navigate the solution space and make smart incremental decisions. [20] proposed an attention model rooted in transformer architecture [33], demonstrating impressive results on a variety of routing challenges. Recently, several advanced architectures have been proposed, each meticulously designed to address distinct OR challenges. Notable among these are solutions to the job shop scheduling problem using directed acyclic GNNs [37] and to assignment problems using bipartite GNNs [14,38]. The works most closely related to ours include research on predicting the feasibility of LP [8] and attempt to solve MAXSAT problems [24]. Both approaches are based on supervised learning techniques, in which the generation of labels for each instance is required to solve the problems to optimality.

3 Methodology

Problem formulation. Given an infeasible problem, defined by a set of constraints $C = \{c_1, c_2, \ldots, c_m\}$, the problem cannot be solved because the constraints do not allow a feasible solution. In such cases, a natural question arises: How can we make the problem feasible? To address this, the goal is to find the minimum subset of constraints called the coverset $\bar{C} \subseteq C$, that must be removed so that the remaining set $C \setminus \bar{C}$ allows the problem to be feasible (illustrated in Figure 1).

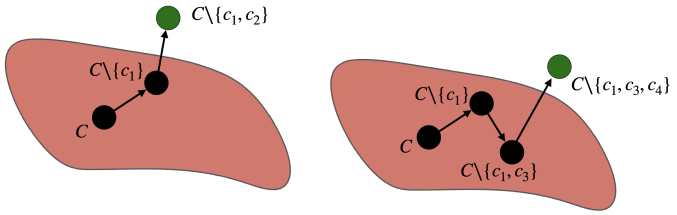

Fig. 1: Finding a good set $\bar{C}$ can be framed as a shortest path problem. The top solution removes only two constraints, while the bottom one leads to a longer path, requiring the removal of three.

Graph representation. It is known that many constraint optimization problems [13,14,24,38] can be represented as a bipartite graph $\mathcal{G} = (\mathbf{C}, \mathbf{E}, \mathbf{V})$, where $\mathbf{C}$ denotes the set of constraint features, $\mathbf{V}$ the variable features, and $\mathbf{E}$ the edges (illustrated in Figure 2).

In the LF domain, each node feature i in the vector $\mathbf{C}$ is constituted of the right-hand side weight b_i, a categorical variable specifying whether the constraint is an inequality or equality. The edges $\mathbf{E} = (e_{i,j})_{i,j}$ of the bipartite graph correspond to the weights of the variables within the constraints. The variable features $\mathbf{V}$ are set to zero, as our work focuses solely on feasibility, not on the coefficients of the objective function. In the SAT domain, the node feature vector $\mathbf{C}$ is set to 0. Each edge is set to 0 if the variable does not appear in the clause, 1 if the variable is included in the clause, and -1 if the variable appears in its negated form. Similarly to the LF domain, the variable features $\mathbf{V}$ are also set to 0, focusing solely on assessing the satisfiability of clauses without the influence of variable weights.

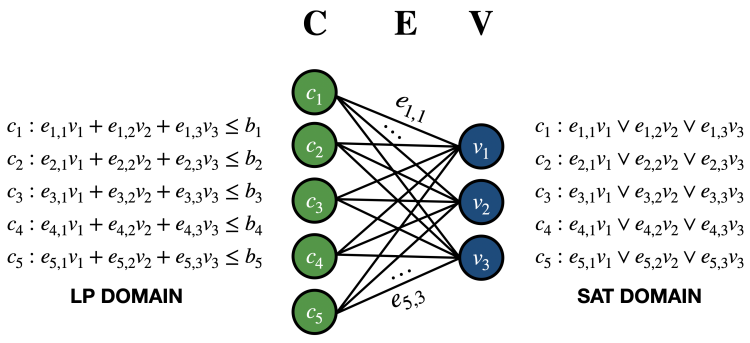

Fig. 2: Bipartite graph representation of LF constraints and SAT clauses.

Markov Decision Process. To learn the constraint selection policy, we define the underlying MDP as outlined below and illustrated in Figure 3.

The **state** s_t represents the current state of the environment at time step t. Specifically, the agent takes the graph features $s_t := (\mathcal{G}_t, \bar{C}_t)$ that encapsulate the current problem instance and the set of constraints $\bar{C}_t$ that have been removed so far.

The **action** a_t refers to the index of the next constraint to be removed and added to the subset $\bar{C}_t$.

The **transition** function $p(s_{t+1} \mid s_t, a_t)$ is deterministic. The next state s_{t+1} is obtained by removing the constraint indexed by a_t and updating the bipartite graph accordingly. The state s_{t+1} can be terminal if the problem becomes feasible after this action.

The **reward** function r is designed to minimize the number of constraints removed to achieve feasibility. The reward in each step is defined as $r(s_t, a_t) = -1$ if the problem remains infeasible after removing the constraint indexed by a_t. The reward encourages the agent to remove the smallest possible set of constraints, as the constant -1 reward frames the MDP as a stochastic shortest path problem.

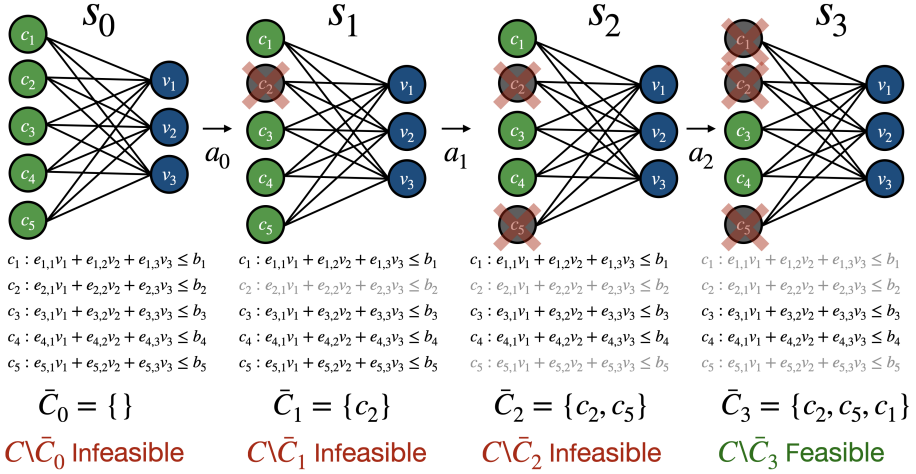

Fig. 3: MDP on a the LF domain. The agent's goal is to construct the smallest subset of constraints that restore the feasibility of the problem. At each step t, the agent selects a constraint to remove and increments the subset $\bar{C}_t$. The agent continues until the problem becomes feasible.

Bipartite GNN and policy. Our approach focuses on learning rich representations of constraints, which can then be used by an RL policy (see Figure 4).

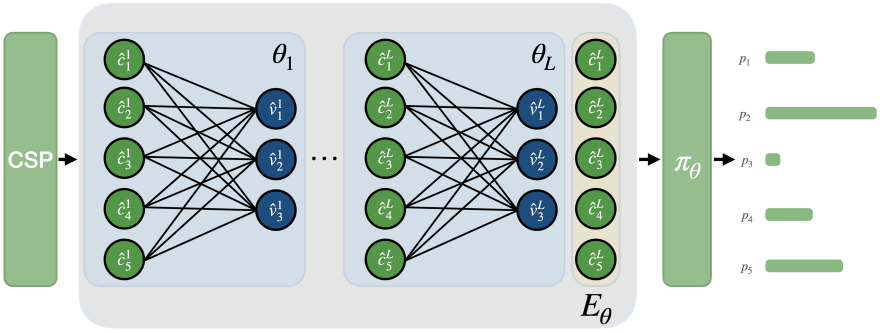

Fig. 4: The general framework of applying GNNs policy to solve MAXFS and MAXSAT problems involves converting the problem into a bipartite graph. Each pair of node and edge in the graph is assigned an initial embedding, which is iteratively updated through a message passing process. Finally, the policy outputs a score for each constraint.

In this work, we propose two encoder architectures for learning constraint embeddings. Both architectures take a bipartite graph $\mathcal{G} = (\mathbf{C}, \mathbf{E}, \mathbf{V})$ as input and output the embeddings for constraints and variables, denoted as $\hat{\mathbf{C}}$ and $\hat{\mathbf{V}}$, after L layers of processing. The first encoder E_θ^{GCNN} is based on the Graph Convolutional Neural Network (GCNN) model introduced by [13], which uses simple graph convolutions to construct constraint embeddings. This process involves message passing between nodes in the bipartite graph, with updates occurring in two stages: from variables to constraints and then from constraints to variables. The second E_θ^{DKA}, based on the Deep K-partite Assignment (DKA) model proposed by [38], employs a self-attention mechanism to learn constraint embeddings. Here, the embeddings are derived by averaging the edge embeddings, which are updated using attention layers within the DKA framework. The specific details of the message-passing mechanisms for both architectures are provided in [13,38].

Building on top of the constraint embeddings generated by the GNN, we propose a policy network that uses a simple Multi-Layer Perceptron (MLP) to score each constraint. This policy network, denoted as π_θ, is responsible for guiding the sequential selection of constraints to be removed to achieve feasibility. Mathematically, policy $\pi_\theta(\hat{\mathbf{C}})$ accepts constraint embeddings $\hat{\mathbf{C}}$ as input and outputs a score p_i for each constraint, representing the probability of removing that constraint at the current time step. The value network takes as input $\bar{g}$ defined as: $\bar{g} = \frac{1}{|C|}\sum_{\hat{c}\in\hat{C}}\hat{c} + \frac{1}{|V|}\sum_{\hat{v}\in\hat{V}}\hat{v}$. Here, $\bar{g}$ represents a global graph pooling of the graph embedding, where $\hat{C}$ and $\hat{V}$ are the sets of constraint and variable embeddings, respectively. The policy network is trained using Proximal Policy Optimization (PPO) [30] to maximize the expected cumulative reward and, equivalently, to minimize the number of constraints removed to achieve feasibility. Like the policy described in [5], our approach also incorporates a

masking mechanism to ensure that the constraints are not removed more than once. Overall, this actor-critic architecture leverages the GNN-based constraint embeddings to effectively navigate the space of possible constraint removals. Following [5], we scale the node scores (before masking) within $[-K, K]$ using as an activation function tanh (see Equation 1). Then we compute the final output probability p_i using a softmax:

$$u_i = \begin{cases} K \cdot \tanh\left(\pi_\theta(\hat{c}_i)\right) & \text{if } c_i \notin \bar{C} \\ -\infty & \text{otherwise} \end{cases} \qquad p_i = \frac{e^{u_i}}{\sum_{j=1}^{|C|} e^{u_j}}. \tag{1}$$

4 Experiments

Result on linear feasibility and sat. To validate our approach, we generate an extensive benchmark consisting of various infeasible problems, including LF and SAT problems of varying sizes, defined by the number of variables ($\mathbf{v}$) and constraints ($\mathbf{c}$). Given the lack of large datasets of infeasible instances in both domains, we generated our own instances of different sizes to ensure a comprehensive evaluation. We train our agent in these diverse instances and test its performance on unseen new instances.

Algorithm 1 Applying the learned heuristic at inference

Require: Infeasible set of constraints C, trained encoder E_θ, trained policy π_θ,
  $\bar{C} \leftarrow \emptyset$
  while `infeasible`$(C \setminus \bar{C})$ **do**
    $\mathcal{G} \leftarrow$ `construct_graph`$(C, \bar{C})$ {Construct the graph}
    $\hat{\mathbf{C}} \leftarrow E_\theta(\mathcal{G})$ {Compute constraint embeddings}
    $P \leftarrow \pi_\theta(\hat{\mathbf{C}}, \bar{C})$ {Compute scores for each constraint}
    $c \leftarrow \text{argmax}_{c' \in C \setminus \bar{C}} P(c')$ {Select constraint with highest score}
    $\bar{C} \leftarrow \bar{C} \cup \{c\}$ {Add selected constraint to coverset}
  end while
  return $\bar{C}$

For the LF domain, the performance of our method is compared with several baselines, including a simple deletion heuristic described in [4]. We also compare our approach to the state-of-the-art heuristics, Chinneck's heuristics, which include two variants: Chinneck [11] and Fast-Chinneck [10]. Both heuristics are based on iteratively solving elastic LP, where the Chinneck heuristic aims to provide higher-quality solutions, while the Fast-Chinneck heuristic seeks a trade-off some solution quality for improved time efficiency. For smaller instances, we further compare our approach with exact solutions to assess its optimality, using the Big-M MILP formulation [11]. The deletion metric will be applied exclusively to very large instances, providing a reliable baseline in expansive settings (see Table 3). Due to the extensive execution time, the classical Chinneck heuristics will not be included in Table 3.

In the SAT domain (see Table 2), we compare our method to a state-of-the-art solver, RC2 [17], implemented in the PySAT library [16]. Additionally, we implement a random heuristic that iteratively selects a constraint at random (uniformly) for removal until satisfiability is achieved, providing an upper bound on the number of constraints removed. To further evaluate our method, we consider two types of settings: one in which each constraint is assigned equal weights, denoted as LF and SAT, and another in which weights are sampled from a uniform distribution, denoted as wLF (weighted LF) and wSAT (weighted SAT).

In each table, the cost of a solution refers to the number of constraints removed; in wLF and wSAT, it corresponds to the weighted number of constraints removed. In Table 1, the first column is not bolded as it represents the reference optimal solution cost.

Table 1: Average solution cost for various heuristics across 300 instances of MAXFS and weighted MAXFS; lower values indicate better performance. Bold text indicates the best overall heuristics, while gray cells denote the second best. Black cells signify that the method was not applied due to intractability.

	Big-M (Opt) [27]	Chinneck [11]	Chinneck-F [10]	Ours + GCNN [13]	Ours + DKA [38]
LF(**c**10**v**2)	2.83 ± 1.24	3.07 ± 1.48	3.16 ±1.48	2.88 ± 1.23	**2.84 ± 1.22**
LF(**c**20**v**5)	4.29 ± 1.54	4.81 ± 1.9	4.98 ±1.92	4.58 ± 1.71	**4.38 ± 1.55**
LF(**c**50**v**10)	11.09 ± 2.11	12.98 ± 2.79	13.24 ± 2.80	12.55 ± 4.17	**11.33 ± 2.27**
LF(**c**100**v**20)		25.30 ± 3.95	25.54 ± 3.99	26.06 ± 3.5	**21.84 ± 2.80**
LF(**c**150**v**30)		36.60 ± 4.78	36.97 ± 4.92	36.30 ± 3.51	**31.42 ± 3.50**
wLF(**c**10**v**2)	1.30 ± 0.74	1.46 ± 0.86	1.48 ± 0.87	1.37 ± 0.78	**1.32 ± 0.74**
wLF(**c**20**v**5)	1.80 ± 0.82	2.11 ± 1.03	2.18 ± 1.08	2.00 ± 0.83	**1.85 ± 0.89**
wLF(**c**50**v**10)	4.80 ± 1.27	5.98 ± 1.79	6.09 ± 1.84	6.00 ± 1.49	**5.04 ± 1.31**
wLF(**c**100**v**20)		11.61 ± 2.44	11.93 ± 2.45	11.99 ± 2.11	**9.38 ± 1.68**
wLF(**c**150**v**30)		16.73 ± 2.80	17.24 ± 2.92	17.09 ± 2.5	**15.99 ± 2.06**

Table 2: Average solution cost for various heuristics across 300 test instances of MAXSAT and weighted MAXSAT; lower values indicate better performance.

	RC2 [17]	Rnd	Ours + GCNN [13]	Ours + DKA [38]
SAT(**c**20**v**2)	**5.04 ± 1.23**	15.95 ± 2.32	7.53 ± 2.16	**5.04 ± 1.23**
SAT(**c**40**v**4)	**5.46 ± 1.46**	29.06 ± 5.42	5.65 ± 1.50	5.51 ± 1.61
SAT(**c**60**v**3)	**12.87 ± 2.11**	52.74 ± 3.71	17.37 ± 3.58	12.96 ± 2.32
SAT(**c**100**v**5)	**12.78 ± 2.22**	85.63 ± 6.92	19.75 ± 3.89	12.91 ± 2.33
SAT(**c**200**v**10)	**10.20 ± 2.38**	153.2 ± 22.3	19.85 ± 4.09	10.27 ± 2.38
wSAT(**c**20**v**2)	**2.32 ± 0.75**	7.90 ± 1.67	2.59 ± 0.91	**2.32 ± 0.75**
wSAT(**c**40**v**2)	**2.42 ± 0.82**	17.71 ± 2.16	3.40 ± 1.28	2.43 ± 0.82
wSAT(**c**60**v**3)	**6.04 ± 1.22**	26.22 ± 2.80	7.11 ± 1.54	6.08 ± 1.27
wSAT(**c**100**v**5)	**5.86 ± 1.35**	42.67 ± 4.32	9.56 ± 2.43	5.94 ± 1.39
wSAT(**c**200**v**10)	**4.57 ± 1.35**	76.11 ± 11.46	5.73 ± 1.69	4.81 ± 1.40

Better solution quality. In both domains, our RL approach exhibits very good results compared to other heuristics, achieving near-optimal solutions for small instances and outperforming Chinneck's heuristics and RC2 solver. The DKA-based agent demonstrates better overall performance compared to GCNN, as the attention mechanism can capture more sophisticated relationships, leading to better embedding of constraints.

Generalization to unseen larger graph. In Table 3, we also demonstrate in the LF domain that our agent can generalize in a zero-shot manner to larger instances than those encountered during training. This capability underscores the flexibility and robustness of the model in handling increasing complexities of problems.

Table 3: Zero-shot generalization performance of a model trained on LF(150,30).

	Deletion [4]	Chinneck-F [10]	Ours + GCNN [13]	Ours +DKA [38]
LF(**c**200**v**40)	97.86 ± 18.26	48.76 ± 5.62	45.06 ± 12.83	**37.07 ± 9.35**
LF(**c**300**v**60)	159.6 ± 26.6	72.28 ± 7.72	62.98 ± 28.86	**28.71 ± 10.67**
LF(**c**600**v**60)	345.26 ± 69.2	192.88 ± 11.1	166.59 ± 58.81	**69.59 ± 26.3**
LF(**c**1000**v**80)	463.4 ± 145.3	342.5 ± 28.9	87.19 ± 89.67	**27.3 ± 20.70**

Better time efficiency. For medium and large instances, our model also demonstrates far better time efficiency than Chinneck's heuristics and is quite close to the Chinneck-Fast heuristic (see Figure 5). For the largest instances, both the DKA and GCNN agents are significantly more efficient. This shows that our approaches scale well in terms of both solution quality and time efficiency compared to the already known heuristics.

Graph convolutional neural network vs attention. Both architectures show very good results: the GCNN model exhibits lower memory requirements, lower training, and inference time (see Figure 5) while the DKA model leads to better solution quality, generalization, stability, and sample efficiency (see Figure 6).

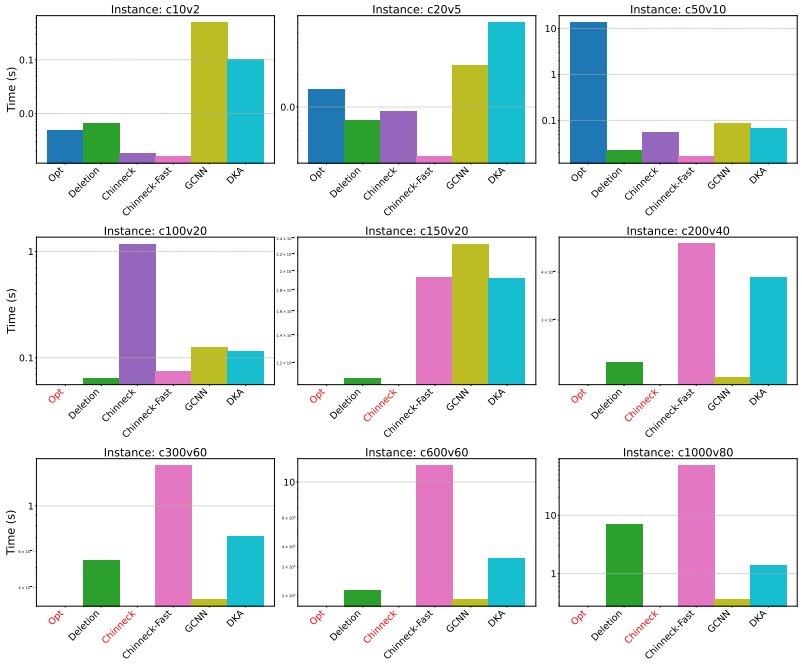

Fig. 5: Time comparison (log scale) of different heuristics across various infeasible LF instances. Red labels indicate methods that could not be run within a reasonable time frame for the given instance.

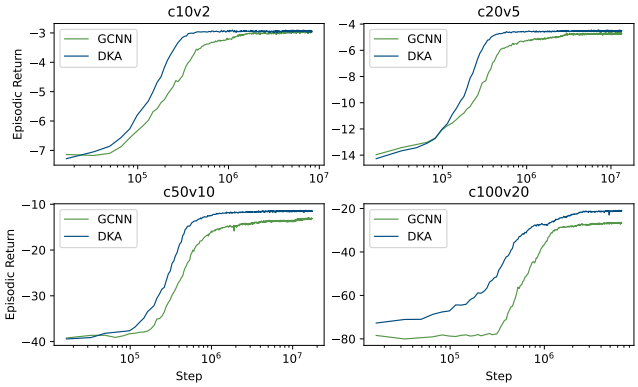

Fig. 6: Training curves comparison between MAXFS agents trained using GCNN and DKA encoders. The x-axis is presented on a logarithmic scale for improved visibility.

Additional results on time efficiency. Most heuristics applied to the benchmarks (excluding the big-M formulation) rely on multiple calls to a solver to verify the feasibility of a given set of constraints. As demonstrated in Figure 5, our RL approach is significantly more time efficient. This efficiency comes from the fact that the RL agent requires at most $|C|$ calls to the solver. In contrast, other approaches may invoke the solver far more frequently than the number of constraints. The Figure 7 compares the number of solver calls across LF domains, illustrating that RL approaches are considerably more effective in reducing solver interactions.

Furthermore, as shown in Figure 6 for the LF context, the use of attention-based encoders (DKA) is also highly effective in the SAT domains (see Figure 8). The performance gap between GCNN and DKA is particularly pronounced in these domains compared to the LF domains. In the latter, the constraint coefficients are scalar, and the SAT domains feature categorical (combinatorial) constraints, which make the problem landscape significantly more complex and challenging to encode. We believe that attention mechanisms act as a superior function approximator in these scenarios, potentially capturing the richer structure and dependencies of the SAT problem space more effectively. However, further investigation is needed to validate this conjecture and better understand the role of attention in these domains.

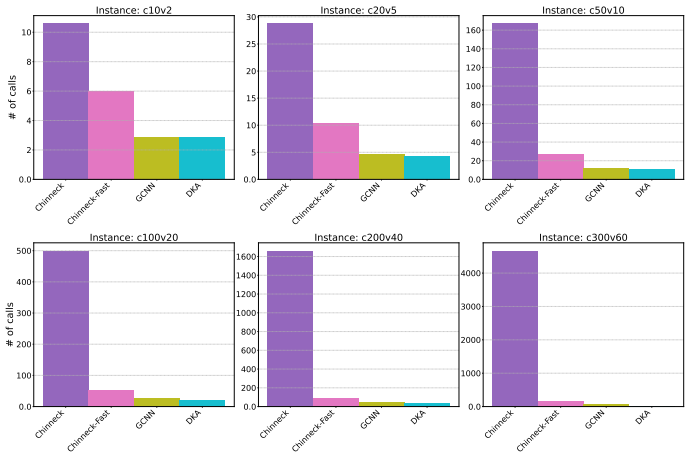

Fig. 7: Number of solver calls for different heuristics across various infeasible RL instances.

Instances generation. To generate infeasible LF instances, we randomly create a constraint matrix and bound values. Specifically, the coefficients for the matrix and bounds are sampled uniformly from a range of integers between -100 and 100. Randomness in these values often leads to conflicting constraints, resulting in infeasible LFs. For infeasible SAT instances, we generate a random

CNF formula by creating clauses using random variables, where each variable is either included as-is or negated. The clauses are designed to collectively create contradictions, ensuring that the SAT instance is unsatisfiable.

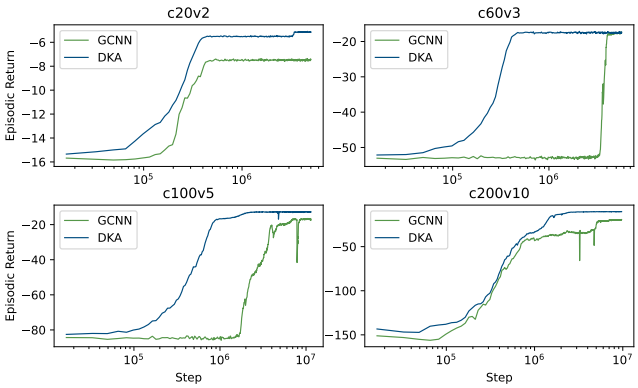

Fig. 8: Training curves comparison between MAXSAT agents trained using GCNN and DKA encoders. The x-axis is presented on a logarithmic scale for improved visibility.

Additional results on the SATLIB. Although the generation of instances for SAT problems was relatively random, we also trained and evaluated our agent using a distribution consistent with SATLIB Uniform Random-3-SAT (see Table 4) . These instances are specifically generated to avoid being trivially unsatisfiable and are sampled from the phase transition region, where the problem instances are the most challenging. More details on the instances and characteristics of the transition phase can be found at the following URL: `https://www.cs.ubc.ca/~hoos/SATLIB/Benchmarks/SAT/RND3SAT/descr.html`.

Table 4: Average solution cost for various heuristics across 300 test instances of MAXSAT following the SATLIB Uniform Random-3-SAT distribution; lower values indicate better performance.

	RC2 [17]	Rnd	Ours + GCNN [13]	Ours + DKA [38]
SAT(**c**19**v**3)	**1.03 ± 0.18**	4.82 ± 2.66	1.62 ± 0.74	**1.03 ± 0.18**
SAT(**c**50**v**10)	**1.06 ± 0.23**	8.15 ± 5.1	2.03 ± 1.2	1.09 ± 0.28
SAT(**c**75**v**15)	**1.12 ± 0.32**	10.77 ± 6.1	2.11 ± 1.26	1.32 ± 0.54
SAT(**c**91**v**20)	**1.06 ± 0.23**	9.20 ± 6.23	2.55 ± 1.52	1.42 ± 0.63

Implementation details. To account for the removal of a variable, we chose to introduce a binary indicator variable rather than physically removing the node from the graph, simplifying the implementation. In Tables 1 and 2, a separate model is trained for each instance distribution and evaluated on 300 unseen instances from the same distribution, resulting in a total of 20 models. In Table 3, a single model is trained on one specific distribution and evaluated on 300 instances from four different distributions, each with a higher number of variables and constraints.

Settings. To check the feasibility of our set of constraints, we used two open-source solvers. HiGHS [15], available in SciPy [35] for LF, and Glucose4 [3] for SAT, available in PySAT [16]. All experiments were run on a desktop machine (Intel i9, 10th generation processor, 64GB RAM) with a single NVIDIA RTX 3090 GPU, using Python. We used PyTorch and PyTorch Geometric [12] to implement our DRL policy and Gymnasium [22] to model the RL environment. The parameters in Table 5 were used to optimize our policy. Table 6 presents the training times in hours.

Table 5: Hyperparameters

Hyperparameter	**Value**
Number of layers	4
Number of attention heads per layer	8
Node embeddings dimension	128
Number of parallel environments	64
Episode length	256
Discount factor γ	0.999
Optimizer (θ of PPO)	Adam
Optimizer learning rate	2e-5
PPO epochs	3
PPO clip	0.2
PPO GAE λ	0.95
PPO value loss coefficient	0.5
PPO entropy loss coefficient	0.01
PPO batch size	256

Table 6: Training time (hours) for various instances size of LF and SAT problems

	GCNN	DKA
LF(**c**10**v**2)	0.55	0.16
LF(**c**20**v**5)	1.22	0.36
LF(**c**50**v**10)	10	4.5
LF(**c**100**v**20)	26	23
LF(**c**150**v**30)	71	61
SAT(**c**20**v**2)	10	7
SAT(**c**40**v**4)	1	0.46
SAT(**c**60**v**3)	1.9	0.71
SAT(**c**100**v**5)	6.62	5
SAT(**c**200**v**10)	20	14

5 Conclusion, limitation and future work

In this work, we have investigated how deep RL and GNNs can be leveraged to efficiently repair infeasible problems. Focusing on LF and SAT, which encompass a wide spectrum of problems in OR, we developed DRL policies capable of competing with (and even outperforming) classical heuristics without any expert knowledge. We introduced two agents that employ different GNN encoders: one with reduced memory and computational requirements, and the other based on attention mechanisms that demonstrate superior performance in solution quality, stability, generalization, and sample efficiency. A limitation, common to neural approaches, is that despite promising results, it remains difficult to clearly surpass the most advanced state-of-the-art solvers, such as RC2 in the case of

MAXSAT. Moreover, our approach relies on repeatedly checking the feasibility of a CSP, which can become computationally expensive (particularly for complex problems such as MILPs or hard SAT instances, where feasibility checks are themselves costly).

Our framework can be seen as a black-box method that only requires feasibility checks at each step, without the need for access to the internal state of the solver, which makes it very general and widely applicable.

Future work could address several challenges. One promising direction is learning to repair MILP and CP instances, as our GNN framework can also encode these problems. We propose a method for repairing CSPs by removing constraints. Future research could explore alternative methods of problem repair, such as modifying the coefficients of constraints within an LP for continuous repair, or providing recommendations for specific cases, such as determining the appropriate number to place in a cell to restore feasibility in a Sudoku puzzle. Furthermore, approaches that employ a general representation of CSP, as demonstrated by the architecture proposed in [32], may offer a wide applicability beyond specific domains.

Another promising extension of this work is to leverage the internal state of the solver, which checks feasibility, to improve the heuristic. This could result in a hybrid method that combines both learning-based and traditional approaches, thereby transforming it into a white-box method.

Finally, applying our approach to large datasets of real-world infeasible problems could further validate and enhance the practicality of our methods.

Acknowledgment We acknowledge the support of the IRT-MINDS project.

References

1. Amaldi, E., Kann, V.: The complexity and approximability of finding maximum feasible subsystems of linear relations. Oper. Res. **147**, 181–210 (1994)
2. Amaldi, E.: From finding maximum feasible subsystems of linear systems to feedforward neural network design. PhD dissertation, Swiss Federal Institute of Technology at Lausanne (EPFL) (1994)
3. Audemard, G., Simon, L.: Glucose: a solver that predicts learnt clauses quality. SAT Competition pp. 7–8 (2009)
4. Bailey, J., Stuckey, P.J.: Discovery of minimal unsatisfiable subsets of constraints using hitting set dualization. In: Practical Aspects of Declarative Languages: 7th International Symposium, PADL 2005, Long Beach, CA, USA, January 10-11, 2005. Proceedings 7. pp. 174–186. Springer (2005)
5. Bello, I., Pham, H., Le, Q.V., Norouzi, M., Bengio, S.: Neural Combinatorial Optimization with Reinforcement Learning (Jan 2017). https://doi.org/10.48550/arXiv.1611.09940, `http://arxiv.org/abs/1611.09940`, arXiv:1611.09940 [cs, stat]
6. Bennett, K.P., Bredensteiner, E.J.: A parametric optimization method for machine learning. INFORMS Journal on Computing **9**(3), 311–318 (1997)
7. Borchers, B., Furman, J.: A two-phase exact algorithm for max-sat and weighted max-sat problems. Journal of Combinatorial Optimization **2**, 299–306 (1998)

8. Chen, Z., Liu, J., Wang, X., Lu, J., Yin, W.: On representing linear programs by graph neural networks. arXiv preprint arXiv:2209.12288 (2022)
9. Chinneck, J.W.: An effective polynomial-time heuristic for the minimum-cardinality iis set-covering problem. Annals of Mathematics and Artificial Intelligence **17**(1), 127–144 (1996)
10. Chinneck, J.W.: Fast heuristics for the maximum feasible subsystem problem. INFORMS Journal on Computing **13**(3), 210–223 (2001)
11. Chinneck, J.W.: Feasibility and Infeasibility in Optimization: Algorithms and Computational Methods, vol. 118. Springer Science & Business Media (2007)
12. Fey, M., Lenssen, J.E.: Fast graph representation learning with PyTorch Geometric. In: ICLR Workshop on Representation Learning on Graphs and Manifolds (2019)
13. Gasse, M., Chételat, D., Ferroni, N., Charlin, L., Lodi, A.: Exact combinatorial optimization with graph convolutional neural networks. Advances in neural information processing systems **32** (2019)
14. Gibbons, D., Lim, C.C., Shi, P.: Deep learning for bipartite assignment problems. In: 2019 IEEE International Conference on Systems, Man and Cybernetics (SMC). pp. 2318–2325 (2019). https://doi.org/10.1109/SMC.2019.8914228
15. Huangfu, Q., Hall, J.J.: Parallelizing the dual revised simplex method. Mathematical Programming Computation **10**(1), 119–142 (2018)
16. Ignatiev, A., Morgado, A., Marques-Silva, J.: Pysat: A python toolkit for prototyping with sat oracles. In: International Conference on Theory and Applications of Satisfiability Testing. pp. 428–437. Springer (2018)
17. Ignatiev, A., Morgado, A., Marques-Silva, J.: Rc2: an efficient maxsat solver. Journal on Satisfiability, Boolean Modeling and Computation **11**(1), 53–64 (2019)
18. Joshi, C.K., Cappart, Q., Rousseau, L.M., Laurent, T.: Learning the Travelling Salesperson Problem Requires Rethinking Generalization (May 2022). https://doi.org/10.4230/LIPIcs.CP.2021.33, http://arxiv.org/abs/2006.07054, arXiv:2006.07054 [cs, stat]
19. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907 (2016)
20. Kool, W., van Hoof, H., Welling, M.: Attention, Learn to Solve Routing Problems! (Feb 2019). https://doi.org/10.48550/arXiv.1803.08475, http://arxiv.org/abs/1803.08475, arXiv:1803.08475 [cs, stat]
21. Kumar, V.: Algorithms for constraint-satisfaction problems: A survey. AI magazine **13**(1), 32–32 (1992)
22. Kwiatkowski, A., Towers, M., Terry, J., Balis, J.U., Cola, G.D., Deleu, T., Goulão, M., Kallinteris, A., Krimmel, M., KG, A., Perez-Vicente, R., Pierré, A., Schulhoff, S., Tai, J.J., Tan, H., Younis, O.G.: Gymnasium: A standard interface for reinforcement learning environments (2024), https://arxiv.org/abs/2407.17032
23. Liffiton, M.H., Sakallah, K.A.: Algorithms for computing minimal unsatisfiable subsets of constraints. Journal of Automated Reasoning **40**, 1–33 (2008)
24. Liu, M., Huang, P., Jia, F., Zhang, F., Sun, Y., Cai, S., Ma, F., Zhang, J.: Can graph neural networks learn to solve the maxsat problem?(student abstract). In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 37, pp. 16264–16265 (2023)
25. Lynce, I., Marques-Silva, J.P.: On computing minimum unsatisfiable cores (2004)
26. Mangasarian, O.L.: Misclassification minimization. Journal of Global Optimization **5**(4), 309–323 (1994)
27. Parker, M.R.: A set covering approach to infeasibility analysis of linear programming problems and related issues. Ph.D. thesis, University of Colorado at Denver Denver, Colorado (1995)

28. Pfetsch, M.E.: Branch-and-cut for the maximum feasible subsystem problem. SIAM Journal on Optimization **19**(1), 21–38 (2008)
29. Puterman, M.L.: Markov decision processes: discrete stochastic dynamic programming. John Wiley & Sons (2014)
30. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347 (2017)
31. Sutton, R.S., Barto, A.G.: Reinforcement learning: An introduction. MIT press (2018)
32. Tönshoff, J., Kisin, B., Lindner, J., Grohe, M.: One model, any csp: graph neural networks as fast global search heuristics for constraint satisfaction. arXiv preprint arXiv:2208.10227 (2022)
33. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I.: Attention is all you need. Advances in neural information processing systems **30** (2017)
34. Vinyals, O., Fortunato, M., Jaitly, N.: Pointer Networks (Jan 2017). https://doi.org/10.48550/arXiv.1506.03134, http://arxiv.org/abs/1506.03134, arXiv:1506.03134 [cs, stat]
35. Virtanen, P., Gommers, R., Oliphant, T.E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., et al.: Scipy 1.0: fundamental algorithms for scientific computing in python. Nature methods **17**(3), 261–272 (2020)
36. Xing, Z., Zhang, W.: Maxsolver: An efficient exact algorithm for (weighted) maximum satisfiability. Artificial intelligence **164**(1-2), 47–80 (2005)
37. Zhang, C., Song, W., Cao, Z., Zhang, J., Tan, P.S., Chi, X.: Learning to dispatch for job shop scheduling via deep reinforcement learning. Advances in neural information processing systems **33**, 1621–1632 (2020)
38. Zouitine, M., Berjaoui, A., Lagnoux, A., Pellegrini, C., Rachelson, E.: Learning heuristics for combinatorial optimization problems on k-partite hypergraphs. In: International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research. pp. 304–314. Springer (2024)