Large Neighborhood Search meets Iterative Neural Constraint Heuristics

Yudong W. Xu[1], Wenhao Li[1], Scott Sanner[1,2], and Elias B. Khalil[1]

[1] Department of Mechanical & Industrial Engineering, University of Toronto
[2] Vector Institute
wil.xu@mail.utoronto.ca, chriswenhao.li@mail.utoronto.ca,
ssanner@mie.utoronto.ca, elias.khalil@utoronto.ca

Abstract. Neural networks are being increasingly used as heuristics for constraint satisfaction. These neural methods are often recurrent, learning to iteratively refine candidate assignments. In this work, we make explicit the connection between such iterative neural heuristics and Large Neighborhood Search (LNS), and adapt an existing neural constraint satisfaction method—ConsFormer—into an LNS procedure. We decompose the resulting neural LNS into two standard components: the destroy and repair operators. On the destroy side, we instantiate several classical heuristics and introduce novel prediction-guided operators that exploit the model's internal scores to select neighborhoods. On the repair side, we utilize ConsFormer as a neural repair operator and compare the original sampling-based decoder to a greedy decoder that selects the most likely assignments. Through an empirical study on Sudoku, Graph Coloring, and MaxCut, we find that adapting the neural heuristic to an LNS procedure yields substantial gains over its vanilla settings and improves its competitiveness with classical and neural baselines. We further observe consistent design patterns across tasks: stochastic destroy operators outperform greedy ones, while greedy repair is more effective than sampling-based repair for finding a single high-quality feasible assignment. These findings highlight LNS as a useful lens and design framework for structuring and improving iterative neural approaches.

Keywords: Recurrent Transformer · Large Neighborhood Search · Constraint Satisfaction Problems.

1 Introduction

Constraint satisfaction is ubiquitous in automated decision-making applications that involve planning, scheduling, and resource management, among others. By combining search, inference, and heuristics, constraint solvers have exhibited continual improvement over the past few decades, leading to reliable performance on a diverse range of challenging problems. That being said, there is also emerging interest in a complementary direction, namely the development of fast heuristics based on end-to-end machine learning (ML) [25, 4].

By design, ML heuristics exploit statistical patterns within a homogeneous distribution of training instances, e.g., coloring problems on graphs with a similar structure. While an appropriately trained heuristic can perform well on test instances similar to those seen in training, it is unlikely to perform well out of distribution, e.g., when the number of nodes or the density of the graph changes in a coloring problem. One-shot solution prediction methods are particularly prone to this failure mode as they have a fixed per-instance computational budget that cannot adapt to instance size or complexity [26]. Classical solvers, on the other hand, simply search further, as desired.

To address this limitation and adapt to out-of-distribution instances, *recurrence* has been introduced to many existing neural architectures. This includes masked diffusion models [30], recurrent Transformers [11, 40], and single-step iterative Transformers [39]. In these approaches, the model is applied repeatedly over multiple iterations. The number of iterations can scale with the difficulty of the instance, partially bypassing the fixed-compute constraint of one-shot solution prediction and often improving generalization to harder or larger problems [39, 11]. A common thread across these iterative neural methods is that they implicitly or explicitly refine a solution while optimizing some objective that captures the degree of constraint satisfaction. Viewed at a higher level, this is closely related to classical local search techniques and, in particular, Large Neighborhood Search (LNS) [24, 29].

In this paper, we make this connection explicit by viewing a recent neural constraint heuristic, ConsFormer [39], as an implicit neural LNS procedure. We formalize the deployment of ConsFormer within a generic LNS loop and decompose the resulting neural LNS into two standard components: the destroy operator and the repair operator. We refer to the resulting procedure as ConsFormer-LNS. This perspective allows us to import standard LNS design dimensions and to ask: which classical LNS heuristics carry over to the neural setting, and what new heuristics become possible when the repair operator is a learned model with access to rich internal signals? To this end, we make the following contributions:

- We reinterpret the ConsFormer [39] as an implicit neural LNS solver in which the Transformer acts as part of the repair operator and embed it in an explicit LNS procedure.
- We propose prediction-guided destroy operators that exploit the neural network's logits as signals for variable selection, alongside classical heuristics.
- We conduct a systematic empirical study of classical and neural-specific LNS heuristics on several CSP benchmarks. Adapting the neural heuristic to an LNS procedure yields substantial gains over its vanilla iterative deployment.

2 Related Work & Background

2.1 Constraint Satisfaction Problems

A *constraint satisfaction problem* (CSP) is given by a tuple $(X, \mathcal{D}, \mathcal{C})$, where $X = \{x_1, \ldots, x_n\}$ is a set of variables, each x_i takes values in a finite discrete

domain $\mathcal{D}_i \in \mathcal{D} = \{\mathcal{D}_1, \ldots, \mathcal{D}_n\}$, and $\mathcal{C} = \{c_1, \ldots, c_m\}$ is a set of constraints. Each constraint $c_j \in \mathcal{C}$ is defined over a subset of variables $X_j \subseteq X$ and restricts the assignments to X_j. Let $\mathcal{X} \triangleq \prod_{i=1}^{n} D_i$ denote the Cartesian product over variable domains, and an assignment by $\mathbf{x} = (x_1, \ldots, x_n) \in \mathcal{X}$. The objective of a CSP is to find $\mathbf{x}$ that satisfies every constraint: $c_j(\mathbf{x}) = \texttt{true}, \quad \forall j \in [m]$.

2.2 Large Neighborhood Search

Large Neighborhood Search is a popular approach for solving CSPs [24, 29]. LNS iteratively destroys part of an existing solution and then rebuilds it. A destroy operator removes specific components of a solution. A repair operator then reinstantiates the removed elements to construct a new, complete feasible solution.

Neural LNS has been an active area of recent research [5, 12]. Existing work includes both utilizing neural methods as the destroy operator as well as the repair operator. For example, various works have studied learning a neural destroy operator to select the neighborhood in integer programs using reinforcement learning [38, 31], imitation learning [32], contrastive learning [18], graph convolutional networks [42], and hindsight relabeling [13]. Hottung et al. use reinforcement learning to learn destroy and repair operators for vehicle routing problems [15, 17, 16]. Falkner et al. [10] attack similar problems with graph neural networks as repair operators.

2.3 Iterative Neural Solvers

Masked Generative Transformers [7], Recurrent Transformers [8], and Masked Diffusion Models [30] have gained popularity as neural heuristics for discrete data, and have often been applied to solve simple constraint satisfaction problems such as Sudoku [3, 19, 40] and harder combinatorial optimization problems in general [28, 33, 39]. Within these recurrent neural heuristics, techniques like remasking and subset improvement have been used to enable iterative solution improvement where the framework forgets or refines parts of the solution [36, 39], which closely mirrors the LNS destroy-repair pattern.

2.4 ConsFormer

In this work, we adapt the ConsFormer [39], a recent model utilizing the Transformer architecture to solve constraint satisfaction problems by iteratively improving a single solution (cf. Section 3.1 for a full problem definition). ConsFormer takes in a complete, potentially infeasible variable assignment $\mathbf{x} = (x_1, \ldots, x_n)$ as a sequence of tokens. Each variable's input representation combines a learned value embedding as well as positional information, i.e., information about the index i of variable x_i.

The model randomly selects a subset of variables $S \subset \mathbf{x}$ to update during a forward pass. A specialized learnable embedding $\mathbf{e}_s$ is added to the tokens corresponding to variables in S to signal their eligibility for modification. The

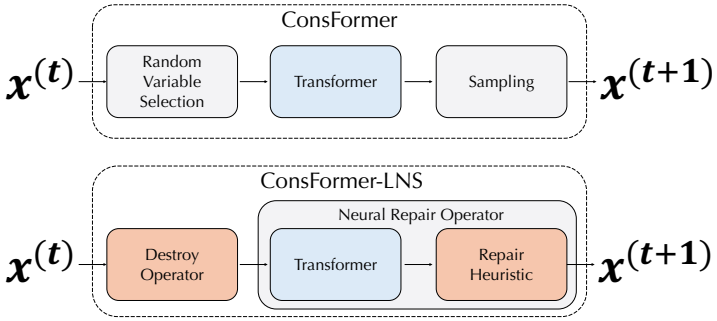

Fig. 1. Comparison between the original ConsFormer update and our ConsFormer-LNS. **Top:** ConsFormer takes the current assignment $x^{(t)}$, selects a random subset of variables, applies the Transformer, and samples a new assignment $x^{(t+1)}$. **Bottom:** In ConsFormer-LNS, a destroy operator selects variables to modify, and the Transformer acts as part of a neural repair operator that proposes the next assignment $x^{(t+1)}$.

Transformer processes this sequence to produce updated assignments; for variables in S, new values are sampled via the Gumbel-Softmax operator applied to the output logits to maintain differentiability, while variables not in S remain unchanged.

ConsFormer is trained in a self-supervised way using differentiable constraint penalty functions p_k, such that $p_k(\mathbf{x}) = 0$ if $\mathbf{x}$ satisfies the k-th constraint and $p_k(\mathbf{x}) > 0$ otherwise. Given the model's soft outputs for an instance, the training loss is a weighted sum of these penalties, $\mathcal{L}(\mathbf{x}; \mathbf{\Theta}) = \sum_k \lambda_k f(p_k(\mathbf{x}))$, where f is typically a quadratic transformation, weights λ_k are hyperparameters, and $\mathbf{\Theta}$ are the learnable model parameters. This loss directly measures constraint violation under the model's predictions and provides gradients with respect to the prediction scores. In our work, we reuse ConsFormer's architecture and self-supervised loss, and reinterpret it as part of a neural repair operator inside an LNS framework.

3 Methodology

We reinterpret ConsFormer as a neural Large Neighborhood Search (LNS) procedure, allowing us to modify its components along the destroy and repair axes.

3.1 Neural LNS View of ConsFormer

We consider a CSP instance with variables $X = \{x_1, \ldots, x_n\}$, domains $\mathcal{D}_i$, and constraints $\mathcal{C}$. A complete assignment is denoted by $\mathbf{x} = (x_1, \ldots, x_n)$, with a cost function $\text{cost}(\mathbf{x})$ measuring constraint violation. As described in Section 2.4, ConsFormer iteratively refines an assignment $\mathbf{x}^{(t)}$. We map this iterative process to the standard LNS components as follows:

- **Destroy (The Subset Selection):** The random selection of the subset $S^{(t)} \subset \mathbf{x}$ (and the application of the embedding $\mathbf{e}_s$) functions as a stochastic destroy operator. By flagging these variables for update, the model effectively "unassigns" them, rendering them eligible for modification while keeping $X \setminus S^{(t)}$ fixed.
- **Repair (The Transformer Pass):** The Transformer's forward pass acts as a learned repair operator. Given the current assignment to the variables and the "destroyed" subset $S^{(t)}$, the model proposes a new assignment $\mathbf{x}^{(t+1)}$ for the subset of variables by sampling from their output logits $z_i^{(t)}$ for $i \in S$ via Gumbel-Softmax.

Figure 1 illustrates this reinterpretation: the original ConsFormer loop can be seen as an LNS procedure with a random destroy operator and a neural repair operator that samples for the next solution. We build on this view to study alternative design choices for each of the components.

3.2 Destroy Operator

At each iteration t, the destroy operator defines a binary mask $m^{(t)} \in \{0,1\}^n$, where $m_i^{(t)} = 1$ means variable x_i is selected. Following ConsFormer, the size of the selected set $S^{(t)} = \{i : m_i^{(t)} = 1\}$ adheres to a specified degree of destruction $\rho \in (0,1]$. We adapt several classical destroy heuristics from the LNS literature [21] and introduce neural prediction-guided methods that exploit internal signals from the ConsFormer model.

Classical Destroy Operators

Random removal. Let $\pi_i^{(t)} \in [0,1]$ denote the probability of destroying variable x_i at iteration t. The simplest strategy is to select the subset at random by setting all $\pi_i^{(t)}$ to the rate of destruction ρ, which corresponds to the original ConsFormer. We sample the binary mask by drawing a Bernoulli random variable for each variable:

$$\pi_i^{(t)} = \rho, \quad m_i^{(t)} \sim \text{Bernoulli}(\pi_i^{(t)}).$$

Greedy worst removal. A more targeted strategy destroys variables that contribute the most to constraint violations. Using ConsFormer's relaxed constraint penalty loss $\mathcal{L}$, we define violation scores per variable to be $v_i(\mathbf{x}^{(t)})$ by evaluating the loss on the discretized variable assignments and extracting the per-variable contributions. In practice, we compute

$$v_i(\mathbf{x}^{(t)}) = \left\| \frac{\partial \mathcal{L}\big(\text{OneHot}(\mathbf{x}^{(t)})\big)}{\partial \big(\text{OneHot}(\mathbf{x}_i^{(t)})\big)} \right\|_1 .$$

We can then set $m_i^{(t)} = 1$ for the top-k variables which yields a deterministic worst variable removal operator.

Stochastic worst removal. To improve diversification, we can randomize the selection while biasing towards highly violating variables [24]. We compute a normalized score from $v_i(\mathbf{x}^{(t)})$ such that the average probability matches ρ:

$$\pi_i^{(t)} \propto v_i(\mathbf{x}^{(t)}), \quad \text{s.t.} \quad \frac{1}{n}\sum_{i=1}^{n} \pi_i^{(t)} \approx \rho,$$

$$m_i^{(t)} \sim \text{Bernoulli}(\pi_i^{(t)})$$

Variables with higher violation scores thus have a higher selection probability.

Stochastic related removal. Introduced by Shaw [29], related removal destroys a group of "related" variables at once. In our setting, a natural notion of relatedness is participation in a common constraint. At each iteration we sample a random subset of constraints and then destroy all variables that belong to any selected constraint. Concretely, we draw a Bernoulli mask over constraints and define the destroy set accordingly:

$$m_k^{\text{constr}} \sim \text{Bernoulli}(\rho), \quad S^{(t)} = \bigcup_{k:m_k^{\text{constr}}=1} \text{vars}(c_k).$$

For graph-based problems where the number of constraints significantly exceeds the number of variables, we rescale the per-constraint Bernoulli parameter so that the expected number of selected constraints remains proportional to the desired variable-level degree of destruction ρ.

Greedy related removal. We can build on random related removal by greedily selecting the constraints with the largest per constraint penalty p_k introduced in Section 2.4. Given the current assignment $\mathbf{x}^{(t)}$, each constraint has a violation score $p_k(\mathbf{x}^{(t)}) \geq 0$, we select the subset of constraints with the largest penalties and destroy all variables that participate in them:

$$\mathcal{K}^{(t)} = \text{top-}k\{p_k(\mathbf{x}^{(t)}) : c_k \in \mathcal{C}\}, \quad S^{(t)} = \bigcup_{k\in\mathcal{K}^{(t)}} \text{vars}(c_k).$$

Prediction-guided Destroy Operators We can design heuristics unique to the neural setting by utilizing the model's internal latent embeddings.

Gradient-guided removal. After the final hidden layer, ConsFormer produces a vector of logits $z_i^{(t)} \in \mathbb{R}^{|\mathcal{D}_i|}$ corresponding to the variables x_i, which we can be interpreted as the model's current belief over the values of x_i. Our first neural strategy uses the gradient of the loss function with respect to the model's current belief to drive the destroy operator.

To do this, we evaluate the penalty loss used during training on the current belief. We then compute the gradient of $\mathcal{L}$ with respect to the logits

$$g_i^{(t)} = \frac{\partial \mathcal{L}(\text{softmax}(z^{(t)}))}{\partial z_i^{(t)}} \quad \text{for each variable } x_i,$$

and use the gradients as a per-variable score. We note that this is different from the gradient used for worst removal, since we rely on the model's internal signals instead of a discretized variable assignment. We include a greedy and a stochastic variant similar to classical approaches, where the greedy variant selects the top-k variables with the largest gradients, and the random variant defines

$$\pi_i^{(t)} \propto \left\|g_i^{(t)}\right\|_1, \quad \text{s.t.} \quad \frac{1}{n}\sum_{i=1}^{n} \pi_i^{(t)} \approx \rho,$$

$$m_i^{(t)} \sim \text{Bernoulli}(\pi_i^{(t)})\,.$$

Variables with larger $\pi_i^{(t)}$ are those for which small changes in the prediction would most strongly affect the loss. In contrast to purely violation-based heuristics, this gradient-based removal explicitly leverages the neural model and the training loss to identify variables that are most "responsible" for the current soft constraint violations.

Confidence-margin removal. Inspired by recent work in Masked Diffusion Models [20, 3], we introduce a strategy using solely the model's current belief. Intuitively, we target variables for which the model is highly uncertain, regardless of their current violation.

We first transform the logits into probabilities $q_i^{(t)} = \text{softmax}\big(z_i^{(t)}\big)$, we then quantify confidence using the gap between the top two probabilities in $q_i^{(t)}$. Let v_1, v_2 be the assignment values with the largest and second-largest entries in q_i, we define a confidence margin

$$\text{margin}_i^{(t)} = q_i^{(t)}(v_1) - q_i^{(t)}(v_2).$$

This margin can be used as the score for selecting the variables. Intuitively, small margins indicate that the model is unsure between multiple values. We again introduce a greedy and a stochastic variant where $\pi_i^{(t)} \propto (-margin_i^{(t)})$.This encourages the search to repeatedly revisit parts of the assignment where the model has low confidence and may benefit from additional refinement.

3.3 Repair Operator

The repair operator takes the current assignment $\mathbf{x}^{(t)}$ and destroy set $S^{(t)}$, and proposes a candidate assignment $\mathbf{x}^{(t+1)}$ by modifying only variables in $S^{(t)}$. In our framework, ConsFormer is utilized as a neural repair operator. Given $\mathbf{x}^{(t)}$ and mask $m^{(t)}$, the model produces logits and corresponding probabilities $q_i^{(t)}(v)$ for each variable $x_i^{(t)}$ and value assignment v. We consider two decoding strategies: sampling and greedy decoding.

Stochastic sampling. This is the original ConsFormer implementation. For each variable with $m_i^{(t)} = 1$, we sample $x_i^{(t+1)} \sim q_i^{(t)}(v)$ using the Gumbel–Softmax sampler; for $m_i^{(t)} = 0$, we keep $x_i^{(t+1)} = x_i^{(t)}$. This yields a stochastic repair operator that explores the neighborhood induced by the destroy mask.

Greedy decoding. A deterministic alternative selects the most likely value for each variable with $m_i^{(t)} = 1$:

$$x_i^{(t+1)} = \arg\max_{v \in \mathcal{D}_i} q_i^{(t)}(v)$$

This corresponds to a greedy repair step based on the model's current beliefs.

4 Experiments

This section aims to empirically answer the following research questions:

- **RQ1 (ConsFormer vs. ConsFormer-LNS):** Does adapting ConsFormer to a neural LNS procedure improve its performance?
- **RQ2 (Classical vs. Prediction-guided Destroy Operators):** How do classical and prediction-guided destroy operators contribute to performance gains?
- **RQ3 (Greedy vs. Stochastic Destroy Operators):** How important is randomness in the destroy operator?
- **RQ4 (Greedy vs. Stochastic Repair Operators):** Given a fixed destroy strategy, does greedy repair generally outperform the original sampling-based repair, and how does its per-instance behavior differ from sampling?
- **RQ5 (ConsFormer-LNS vs. Other Solvers):** How does the LNS-enhanced ConsFormer perform compared to other neural or classical solvers?

4.1 Experimental Setup

Problem Selection We evaluate on the same datasets as Xu et al. [39], excluding the nurse rostering problem for which ConsFormer already solves 100% of the instances.

Sudoku is a simple CSP that involves filling a 9×9 grid with digits from 1 to 9 such that each row, column, and 3×3 sub-grid contains all 9 numbers. A single Sudoku instance is defined by a partially filled board and its difficulty is determined by the number of initial values: fewer initially filled cells in the board involve a larger space of possible assignments to the unfilled cells and is therefore harder. Following Xu et al. [39], we use the dataset from SATNet [37] for training and in-distribution testing, and the dataset from RRN [22] for harder out-of-distribution testing.

Graph Coloring seeks an assignment of colors to vertices in a graph such that no two neighboring nodes share the same color. The problem is defined by the graph's structure and the number of available colors k. We study the same 5-coloring and 10-coloring datasets, where training graphs have 50 vertices for $k = 5$ and 100 vertices for $k = 10$ whereas OOD graphs have 100 for $k = 5$ and 200 for $k = 10$.

MaxCut aims to identify a cut in a graph such that the number of edges crossing the partition is maximized. MaxCut can be alternatively viewed as a 2-coloring Max-CSP which seeks an assignment that maximizes the number of satisfied constraints. Since our framework works by reducing the number of violated constraints, it can be applied directly for MaxCut. The model is trained on small generated graphs and tested on benchmark instances from the GSET dataset [41], which includes weighted graphs with sizes ranging from 800 to 10000 vertices.

Baselines For RQ.5, we compare against a robust set of neural and classical baselines established in the literature following Xu et al. [39]. OR-Tools [23][3], a state-of-the-art constraint programming solver, is the primary non-learning baseline; note that many of its internal sub-solving routines utilize Large Neighborhood Search (LNS) [23]. We omit OR-Tools from the Sudoku experiments, as these instances are computationally trivial for exact solvers and are primarily used to benchmark the reasoning capabilities of neural approaches.

Implementation Details We train our models on single H100 GPU nodes. We adopt the hyperparameter configurations from Xu et al. [39] and retrain a separate model for each destroy/repair operator pairing[4].

4.2 Results & Analysis

Experimental results comparing the different destroy and repair operators are presented in Tables 1 to 3. Italic numbers correspond to the original ConsFormer configuration (random destroy + sampling-based repair); bold and underlined numbers mark the best and second best result within each dataset, respectively.

RQ1: ConsFormer vs. ConsFormer-LNS. Across all benchmarks, adapting ConsFormer into an explicit LNS procedure enhances model performance. On Sudoku (Table 1), ConsFormer-LNS improves the out-of-distribution (OOD) percentage of instances solved from 85.8% to 91.8%. On MaxCut (Table 2), ConsFormer-LNS reduces the gap to the best known cut sizes from 16.33 to 4.44 for $|V|$=800, from 12.44 to 8.00 for $|V|$=1k, from 52.11 to 30.56 for $|V|$=2k, and from 115.25 to 63.63 for $|V|$=3k. On graph coloring (Table 3), OOD percentage of instances solved increased from 46.3% to 54.2% for $k = 5$ and from 10.2% to 18.4% for $k = 10$.

RQ2: Classical vs. Prediction-guided destroy. Overall, the best performing classical and prediction-guided destroy operators achieve similar performance

[3] All OR-Tools results are reported using version 9.10 and were run on the same hardware as the ConsFormer-LNS models.

[4] Our code is available at https://github.com/khalil-research/ConsFormer.

Table 1. Sudoku instances solved (%) for different combinations of destroy and repair operators. Columns correspond to the destroy operators specified in Section 3.2 where the Greedy and Stochastic variants are labeled (Gr.) and (St.) respectively. Rows vary the repair operator (sampling-based vs. greedy). Test instances contain 1,000 instances from the SATNet dataset, OOD refers to Out-of-Distribution evaluation on the RRN test dataset which contains 18K instances. All configurations are evaluated for 2K iterations.

Dataset	Repair	Random	Worst		Related		Gradient		Confidence	
			Gr.	St.	Gr.	St.	Gr.	St.	Gr.	St.
Test	Sampled	***100***	94.3	**100**	99.7	**100**	20.6	**100**	0.0	**100**
	Greedy	**100**	85.6	**100**	99.9	**100**	0.0	**100**	0.0	**100**
OOD	Sampled	*85.8*	16.7	71.7	32.1	48.7	0.1	90.6	0.0	90.8
	Greedy	81.2	11.8	**91.8**	30.9	63.9	0.1	84.1	0.0	91.5

Table 2. MaxCut performance on GSET: average gap to the best known cut (lower is better) for different graph sizes. Columns and Rows correspond to destroy and repair operators as in Table 1. All configurations are evaluated with a 180s time limit.

Size	Repair	Random	Worst		Related		Gradient		Confidence	
			Gr.	St.	Gr.	St.	Gr.	St.	Gr.	St.
$\|V\|$=800	Sampled	*16.333*	430.111	473.000	923.667	472.333	167.778	45.111	489.667	9.778
	Greedy	31.667	503.111	38.556	915.000	71.333	137.000	**4.444**	397.000	21.000
$\|V\|$=1K	Sampled	*12.444*	382.889	417.667	781.000	418.556	154.889	30.889	469.222	9.778
	Greedy	25.000	414.111	33.222	714.333	57.000	116.556	**8.000**	364.111	18.778
$\|V\|$=2K	Sampled	*52.111*	832.889	944.778	1646.111	944.667	297.000	69.889	1014.000	37.111
	Greedy	74.667	842.889	97.000	1472.333	157.333	207.444	**30.556**	674.000	62.778
$\|V\| \geq 3K$	Sampled	*115.250*	1268.750	1618.000	2311.500	1467.375	530.625	122.750	1921.125	106.625
	Greedy	142.500	1286.500	188.000	2085.500	229.875	354.500	**63.625**	1230.125	140.250

Table 3. Graph-Coloring instances solved (%). Top: Coloring-5, Bottom: Coloring-10. Columns and Rows correspond to destroy and repair operators as in Table 1. OOD refers to Out-of-Distribution evaluation for ANYCSP and ConsFormer where the number of vertices n in the graph is larger than that of the training instances. All datasets have 1200 instances. All configurations are evaluated with a 10s time limit.

Dataset	Repair	Stochastic	Worst		Related		Gradient		Confidence	
			Gr.	St.	Gr.	St.	Gr.	St.	Gr.	St.
Graph-Coloring-5 ($n = 50 \rightarrow n = 100$)										
Test	Sampled	*81.6*	33.4	77.8	32.0	81.8	41.7	79.9	25.7	82.3
	Greedy	**82.9**	32.9	82.5	22.1	82.8	41.2	**82.9**	32.7	81.8
OOD	Sampled	*46.3*	0.0	41.0	0.0	47.6	0.7	44.8	0.0	49.0
	Greedy	**54.2**	0.1	52.5	0.0	53.7	2.6	54.0	0.0	49.4
Graph-Coloring-10 ($n = 100 \rightarrow n = 200$)										
Test	Sampled	*53.6*	0.0	53.8	0.0	53.5	11.9	**53.9**	0.0	53.8
	Greedy	**53.9**	0.4	**53.9**	0.0	**53.9**	14.8	**53.9**	5.5	**53.9**
OOD	Sampled	*10.2*	0.0	14.7	0.0	14.2	2.6	14.5	0.0	14.8
	Greedy	17.4	0.0	16.8	0.0	**18.4**	5.1	16.6	1.0	16.3

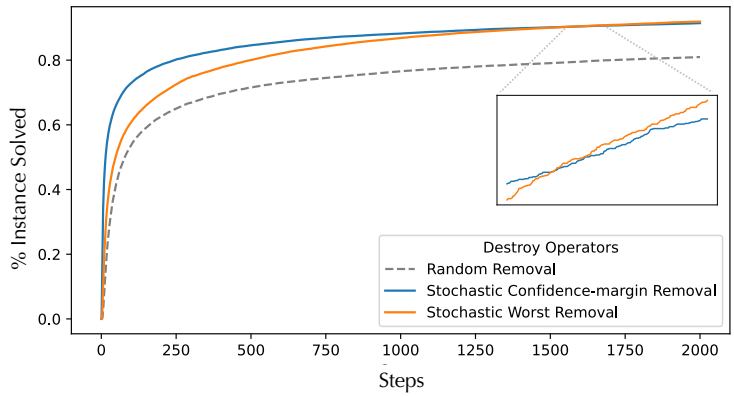

Fig. 2. Sudoku instances solved (%) over LNS steps for the baseline random destroy and the best-performing classical (stochastic worst removal) and prediction-guided (stochastic confidence-margin removal) destroy operators. Both heuristics outperform random removal, with confidence-margin removal improving fastest early on and stochastic worst removal eventually achieving the highest solved percentage.

gain from the baseline, with the exception of MaxCut, where the gradient-guided removal achieves much stronger performance. We further compare the best performing methods by examining the instance solved percentage across iterations in Figure 2. We observe that the confidence-based removal rapidly improves in early iterations, but is overtaken by the random worst removal operator towards the end.

The strong early performance of the confidence-margin model is consistent with the intuition that the model is prioritizing the variable assignments it has the least confidence in. However, this destroy operator receives no explicit signal from the current solution's quality, therefore, as the model becomes more certain in the later iterations, its improvement slows down and eventually gets overtaken by methods with richer penalty-derived signals. This observation aligns with the efficiency-accuracy trade-off reported in the Masked Diffusion literature [3].

The lack of a clear winner among destroy operators, across all benchmarks, is in line with the findings of classical LNS literature: problem structure determines the most suitable Destroy operator [24]. Further augmentation of ConsFormer-LNS with an adaptive mechanism [21] is therefore a promising direction for future work.

RQ3: Greedy vs. Stochastic Destroy Operators The failure of purely greedy destroy operators is observed across all benchmarks. Greedy variants of the different strategies often collapse to near-zero solved percentages on OOD Sudoku and coloring and exhibit very poor cut values on MaxCut. This echoes the classical LNS literature, where randomization is often needed to escape local optima. In contrast, the randomized variants tend to perform well, highlighting

the importance of controlled randomness in the destroy step. Figure 3 shows the per-cell accuracy scores of the best performing destroy operators, i.e., the fraction of variables that are assigned to the correct value in the groundtruth unique feasible Sudoku solution. It can be seen that the greedy variant plateaus much earlier while the stochastic variants are able to achieve much higher scores.

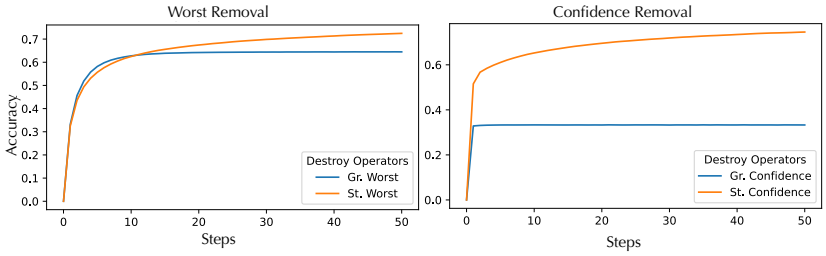

Fig. 3. Cell accuracy for Sudoku over iterations for greedy (blue) and stochastic (orange) variants of Worst and Confidence destroy operators. Greedy variants plateau early, while stochastic variants continue to improve and reach substantially higher accuracies.

RQ4: Greedy vs. Stochastic Repair Operators The greedy repair operator almost always outperforms stochastic sampling-based repair across all benchmarks. To better understand this result, we run ConsFormer-LNS with the baseline and best-performing destroy operators for a small number of iterations for Sudoku. We plot the distribution of the constraint-satisfaction rate as well as the cumulative percentage of instance solved in Figure 4. Across all three configurations, we observe a few clear patterns.

The Greedy repair operator has a better improvement in the first step, but gets overtaken by the stochastic repair operator quickly. At later iterations, the stochastic repair operator has a better overall distribution with a mean closer to 100% accuracy, while greedy repair has a long tail of unsolved instances and a spike at 100% accuracy. Despite the worse performance per step, the greedy repair operator consistently achieves a higher cumulative instance solved percentage in the later steps.

This suggests that while the stochastic sampling-based repair operator learns the solution distribution better through exploration, the greedy repair operator can find the single best solution more often. Intuitively, the superior performance of the greedy operator in our setting is due to our objective of finding a single best solution minimizing our constraint penalty. Our evaluation metrics for the tasks only depend on the best solution and not on the quality or diversity of the underlying solution distribution. Given that the neural model is trained to minimize a constraint violation penalty for the assignments, greedy repair is effectively a *maximum a posteriori*—or MAP—decoding step that directly

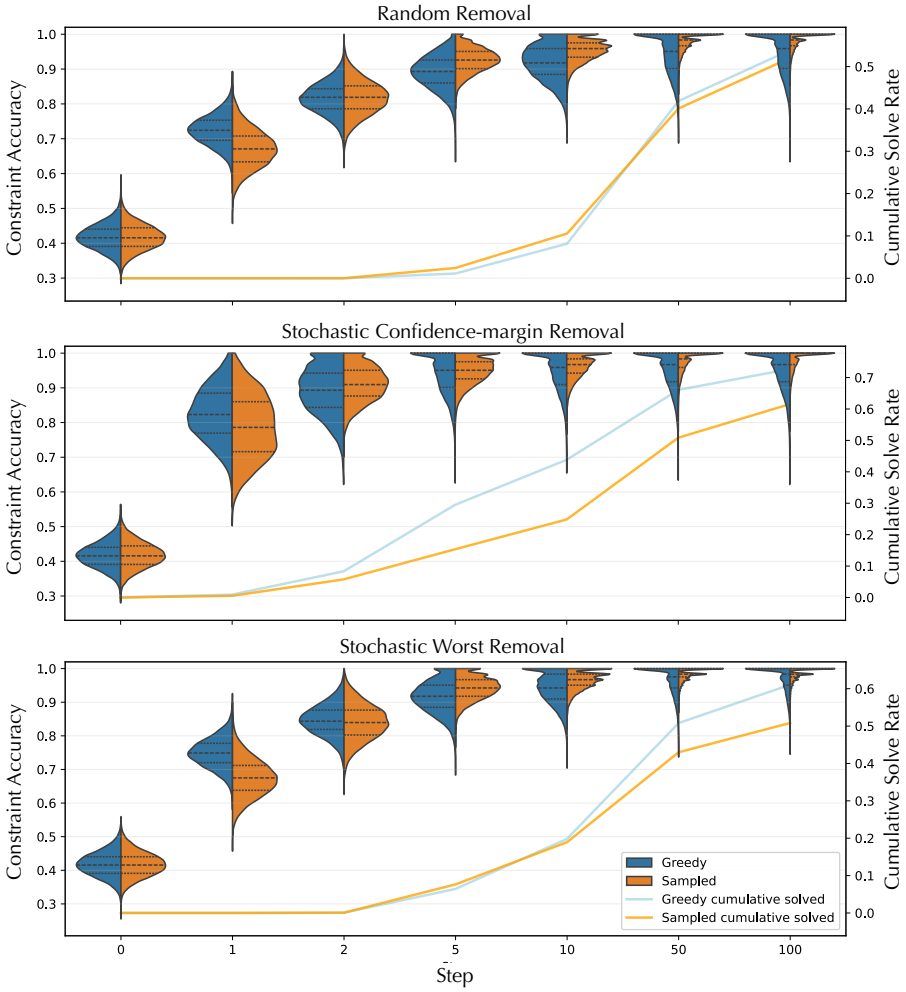

Fig. 4. Sudoku constraint accuracy on OOD Test dataset over LNS steps for greedy (blue) and sampling-based (orange) repair under Random and two best-performing destroy operators (Stochastic Confidence-margin, Stochastic Worst). Split violin plots show the distribution of per-instance constraint satisfaction at selected steps, while the lines on the right axis show the cumulative fraction of instances solved. Greedy repair achieves higher final solved percentages but exhibits a wider tail of unsolved instances, whereas sampling attains higher average constraint accuracy at intermediate steps.

targets a single high-quality assignment, whereas sampling is better aligned with exploring multiple diverse assignments.

RQ5 ConsFormer-LNS vs. Other Solvers We compare the best ConsFormer-LNS configuration on each benchmark against classical and neural baselines (Tables 4 to 6). Overall, enhancing ConsFormer with LNS substantially improves its position among existing methods. On Sudoku, ConsFormer-LNS is compared to other neural approaches and attains the highest OOD percentage of instance solved (91.8% vs. 62.1% for IRED [9]), while matching the best methods on the in-distribution test set. For graph coloring with $k = 5$, ConsFormer-LNS improves on ConsFormer but is still slightly behind OR-Tools on the smaller instances. On the harder graph coloring tasks with $k = 10$, it becomes strongest overall. It marginally surpasses OR-Tools on the test set and achieves a substantially higher OOD solved percentage (18.4% vs.10.25%). For MaxCut, ConsFormer-LNS does not outperform ANYCSP, but it consistently reduces the gap of ConsFormer achieves second across all graph sizes.

Table 4. Performance comparison for Sudoku. In-distribution test instances contain $1,000$ instances; OOD refers to RRN test dataset (18K instances). For all but ConsFormer and ConsFormer-LNS, the reported instance solved (%) are based on the results from Xu et al. [39].

Dataset	**SATNet** [37]	**RRN** [22]	**Recurrent Trans.**[40]	**IRED** [9]	**ConsFormer**	**ConsFormer -LNS**
Test	98.3	<u>99.8</u>	**100**	99.4	**100**	**100**
OOD	3.2	28.6	32.9	62.1	<u>85.8</u>	**91.8**

Table 5. Performance comparison for Graph-Coloring tasks. All methods are evaluated with a 10s time limit. OOD refers to Out-of-Distribution evaluation where n is larger than training instances. For all but ConsFormer and ConsFormer-LNS, the reported instance solved (%) are based on the results from Xu et al. [39].

Dataset	**Greedy**	**OR-Tools** [23]	**ANYCSP** [35]	**ConsFormer**	**ConsFormer -LNS**
Graph-Coloring-5 ($n = 50 \to n = 100$)					
Test	32.42	**83.08**	79.17	81.60	<u>82.9</u>
OOD	0.0	**57.16**	34.83	47.33	<u>54.2</u>
Graph-Coloring-10 ($n = 100 \to n = 200$)					
Test	0.75	52.41	0.00	<u>53.60</u>	**53.90**
OOD	0.0	<u>10.25</u>	0.00	10.2	**18.4**

Table 6. Performance comparison for MaxCut tasks on GSET. Numbers reported are the average gap to the best known cut size, the lower the better. Values are as reported in Xu et al.[39]. We similarly set a time limit of 180 seconds.

Size	Greedy	SDP [14]	RUNCSP [34]	ECO-DQN [1]	ECORD [2]	ANYCSP [35]	OR-Tools [23]	ConsFormer	ConsFormer -LNS
$\|V\|$=800	411.44	245.44	185.89	65.11	8.67	**1.22**	143.89	16.33	4.44
$\|V\|$=1K	359.11	229.22	156.56	54.67	8.78	**2.44**	112.78	12.44	8.0
$\|V\|$=2K	737.00	-	357.33	157.00	39.22	**13.11**	365.89	52.11	30.56
$|V|\geq$3K	774.25	-	401.00	428.25	187.75	**51.63**	378.62	115.25	63.63

5 Conclusion

In this work, we have made the connection between iterative neural constraint solvers and Large Neighborhood Search (LNS) explicit. We reinterpreted ConsFormer, a recent neural heuristic, as a neural repair operator and adapted it into an LNS procedure. We implemented classical as well as novel prediction-guided destroy operators that leverage the model's internal confidence and gradient signals. We systematically evaluated combinations of destroy and repair strategies.

Our empirical evaluation on Sudoku, Graph Coloring, and MaxCut demonstrated a few key findings:

- Porting the ConsFormer into the LNS framework yielded substantial performance gains across all benchmarks.
- Echoing classical LNS findings, purely greedy destroy operators frequently collapsed to poor local minima, whereas their stochastic counterparts continued to improve over iterations and reached much better final performance.
- Our novel destroy operators showed strong performance in early iterations and specific tasks like MaxCut, but no single destroy operator dominated across all tasks. This suggests that an Adaptive LNS [21, 27] that utilizes multiple destroy operators is a promising direction for future work.
- Although sampling-based repair attained better average constraint satisfaction at intermediate steps, greedy decoding almost always achieved higher overall performance, consistent with a MAP-style decoding optimizing for a single best assignment rather than exploring a diverse solution distribution.
- Compared to other approaches, ConsFormer-LNS attains the strongest OOD performance among neural Sudoku solvers, becomes competitive with OR-Tools on graph coloring and strongest on the hardest k=10 OOD setting, and closes the gap to ANYCSP on MaxCut instances.

While neural methods offer promising heuristics for constraint satisfaction, the overlay of an LNS framework has clearly provided many avenues for improvement. To this end, future work is needed to fully cross-pollinate from the LNS literature to iterative neural approaches in order to reliably outperform both historical LNS and neural methods across a variety of problems. Finally, there is further potential to extend the LNS paradigm to neural Diffusion models [30, 6, 3] that bear a striking resemblance to LNS methods and may offer more constrained and controllable diffusion in a range of generative AI applications.

Acknowledgments

We thank the anonymous reviewers for their insightful feedback. This work was supported by the Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korean Government (MSIT) (No. RS-2024-00457882, National AI Research Lab Project).

References

1. Barrett, T., Clements, W., Foerster, J., Lvovsky, A.: Exploratory combinatorial optimization with reinforcement learning. In: Proceedings of the AAAI conference on artificial intelligence. vol. 34, pp. 3243–3250 (2020)
2. Barrett, T.D., Parsonson, C.W., Laterre, A.: Learning to solve combinatorial graph partitioning problems via efficient exploration. arXiv preprint arXiv:2205.14105 (2022)
3. Ben-Hamu, H., Gat, I., Severo, D., Nolte, N., Karrer, B.: Accelerated sampling from masked diffusion models via entropy bounded unmasking. In: The Thirty-ninth Annual Conference on Neural Information Processing Systems (2025), https://openreview.net/forum?id=WBcBhT1NKO
4. Bengio, Y., Lodi, A., Prouvost, A.: Machine learning for combinatorial optimization: a methodological tour d'horizon. European Journal of Operational Research **290**(2), 405–421 (2021)
5. Cappart, Q., Guns, T., Lombardi, M., Pesant, G., Tsouros, D.: Combining constraint programming and machine learning: From current progress to future opportunities. Journal of Artificial Intelligence Research **84** (2025)
6. Cardei, M., Christopher, J.K., Hartvigsen, T., Bartoldson, B.R., Kailkhura, B., Fioretto, F.: Constrained discrete diffusion. arXiv preprint arXiv:2503.09790 (2025)
7. Chang, H., Zhang, H., Jiang, L., Liu, C., Freeman, W.T.: Maskgit: Masked generative image transformer. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). pp. 11315–11325 (2022)
8. Dehghani, M., Gouws, S., Vinyals, O., Uszkoreit, J., Kaiser, L.: Universal transformers. In: International Conference on Learning Representations (2019), https://openreview.net/forum?id=HyzdRiR9Y7
9. Du, Y., Mao, J., Tenenbaum, J.B.: Learning iterative reasoning through energy diffusion. In: International Conference on Machine Learning (ICML) (2024)
10. Falkner, J.K., Thyssens, D., Schmidt-Thieme, L.: Large neighborhood search based on neural construction heuristics. arXiv preprint arXiv:2205.00772 (2022)
11. Fan, Y., Du, Y., Ramchandran, K., Lee, K.: Looped transformers for length generalization. In: The Thirteenth International Conference on Learning Representations (2025), https://openreview.net/forum?id=2edigk8yoU
12. Feng, S., Sun, W., Li, S., Talwalkar, A., Yang, Y.: A comprehensive evaluation of contemporary ml-based solvers for combinatorial optimization. arXiv preprint arXiv:2505.16952 (2025)
13. Feng, S., Sun, Z., Yang, Y.: Spl-lns: Sampling-enhanced large neighborhood search for solving integer linear programs. arXiv preprint arXiv:2508.16171 (2025)
14. Goemans, M.X., Williamson, D.P.: Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. Journal of the ACM (JACM) **42**(6), 1115–1145 (1995)

15. Hottung, A., Tierney, K.: Neural large neighborhood search for the capacitated vehicle routing problem. In: ECAI 2020, pp. 443–450. IOS Press (2020)
16. Hottung, A., Wong-Chung, P., Tierney, K.: Neural deconstruction search for vehicle routing problems. Transactions on Machine Learning Research (2025), https://openreview.net/forum?id=bCmEP1Ltwq
17. Hottung, A., Tierney, K.: Neural large neighborhood search for routing problems. Artificial Intelligence **313**, 103786 (2022)
18. Huang, T., Ferber, A.M., Tian, Y., Dilkina, B., Steiner, B.: Searching large neighborhoods for integer linear programs with contrastive learning. In: International conference on machine learning. pp. 13869–13890. PMLR (2023)
19. Jolicoeur-Martineau, A.: Less is more: Recursive reasoning with tiny networks. arXiv preprint arXiv:2510.04871 (2025)
20. Kim, J., Shah, K., Kontonis, V., Kakade, S.M., Chen, S.: Train for the worst, plan for the best: Understanding token ordering in masked diffusions. In: Forty-second International Conference on Machine Learning (2025), https://openreview.net/forum?id=DjJmre5IkP
21. Mara, S.T.W., Norcahyo, R., Jodiawan, P., Lusiantoro, L., Rifai, A.P.: A survey of adaptive large neighborhood search algorithms and applications. Computers & Operations Research **146**, 105903 (2022)
22. Palm, R., Paquet, U., Winther, O.: Recurrent relational networks. Advances in neural information processing systems **31** (2018)
23. Perron, L., Didier, F.: CP-SAT, https://developers.google.com/optimization/cp/cp_solver/
24. Pisinger, D., Ropke, S.: Large neighborhood search. In: Handbook of metaheuristics, pp. 99–127. Springer (2018)
25. Popescu, A., Polat-Erdeniz, S., Felfernig, A., Uta, M., Atas, M., Le, V.M., Pilsl, K., Enzelsberger, M., Tran, T.N.T.: An overview of machine learning techniques in constraint solving. Journal of Intelligent Information Systems **58**(1), 91–118 (2022)
26. Qiu, R., Sun, Z., Yang, Y.: Dimes: A differentiable meta solver for combinatorial optimization problems. Advances in Neural Information Processing Systems **35**, 25531–25546 (2022)
27. Ropke, S., Pisinger, D.: An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. Transportation science **40**(4), 455–472 (2006)
28. Sanokowski, S., Hochreiter, S., Lehner, S.: A diffusion model framework for unsupervised neural combinatorial optimization. In: International Conference on Machine Learning. pp. 43346–43367. PMLR (2024)
29. Shaw, P.: Using constraint programming and local search methods to solve vehicle routing problems. In: International conference on principles and practice of constraint programming. pp. 417–431. Springer (1998)
30. Shi, J., Han, K., Wang, Z., Doucet, A., Titsias, M.: Simplified and generalized masked diffusion for discrete data. Advances in neural information processing systems **37**, 103131–103167 (2024)
31. Song, J., Yue, Y., Dilkina, B., et al.: A general large neighborhood search framework for solving integer linear programs. Advances in Neural Information Processing Systems **33**, 20012–20023 (2020)
32. Sonnerat, N., Wang, P., Ktena, I., Bartunov, S., Nair, V.: Learning a large neighborhood search algorithm for mixed integer programs. arXiv preprint arXiv:2107.10201 (2021)
33. Sun, Z., Yang, Y.: Difusco: Graph-based diffusion solvers for combinatorial optimization. Advances in neural information processing systems **36**, 3706–3731 (2023)

34. Toenshoff, J., Ritzert, M., Wolf, H., Grohe, M.: Graph neural networks for maximum constraint satisfaction. Frontiers in artificial intelligence **3**, 580607 (2021)
35. Tönshoff, J., Kisin, B., Lindner, J., Grohe, M.: One model, any csp: graph neural networks as fast global search heuristics for constraint satisfaction. In: Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence. IJCAI '23 (2023). https://doi.org/10.24963/ijcai.2023/476, https://doi.org/10.24963/ijcai.2023/476
36. Wang, G., Schiff, Y., Sahoo, S.S., Kuleshov, V.: Remasking discrete diffusion models with inference-time scaling. In: The Thirty-ninth Annual Conference on Neural Information Processing Systems (2025), https://openreview.net/forum?id=IJryQAOy0p
37. Wang, P.W., Donti, P., Wilder, B., Kolter, Z.: Satnet: Bridging deep learning and logical reasoning using a differentiable satisfiability solver. In: International Conference on Machine Learning. pp. 6545–6554. PMLR (2019)
38. Wu, Y., Song, W., Cao, Z., Zhang, J.: Learning large neighborhood search policy for integer programming. In: Beygelzimer, A., Dauphin, Y., Liang, P., Vaughan, J.W. (eds.) Advances in Neural Information Processing Systems (2021), https://openreview.net/forum?id=IaM7U4J-w3c
39. Xu, Y., Li, W., Sanner, S., Khalil, E.B.: Self-supervised transformers as iterative solution improvers for constraint satisfaction. In: Forty-second International Conference on Machine Learning (2025), https://openreview.net/forum?id=IQN6ID0snT
40. Yang, Z., Ishay, A., Lee, J.: Learning to solve constraint satisfaction problems with recurrent transformer. In: The Eleventh International Conference on Learning Representations (2023), https://openreview.net/forum?id=udNhDCr2KQe
41. Ye, Y.: The gset dataset (2003)
42. Zhou, J., Wu, Y., Cao, Z., Song, W., Zhang, J., Chen, Z.: Learning large neighborhood search for vehicle routing in airport ground handling. IEEE Transactions on knowledge and data engineering **35**(9), 9769–9782 (2023)