Exploring Integrality Grip for Mixed-integer Programming by MCTS Planning

Anonymous Author(s)
Affiliation
Address
email

Abstract

In modern Mixed-integer Programming(MIP) solvers, the concept of heuristic is well rooted as a principle underlying the search of high-quality solutions. In this respect, Large Neighborhood Search (LNS) has been the first refinement heuristics for improving existing solutions through a generic MIP solver used as a black box. For a refinement heuristic, the quality of the search neighborhood is of critical importance. However, existing methods have not fully investigated the strategy for balancing exploration and exploitation of search spaces. In this work, we introduce a novel refinement strategy for improving MIP solutions. The proposed framework leverages the ideas of integrality grip to guide the neighborhood selection. Moreover, in order to achieve a good trade-off between exploration and exploitation of the solution space, the LNS search is further improved by investigating the convex relaxations of LNS sub-problems with Monte Carlo Tree Search (MCTS). In particular, at each iteration of LNS, MCTS is firstly executed to evaluate the integrality grip of the convex relxations of next LNS sub-problems. Then the expanded MCTS tree will select a promising solution neighborhood, which will be solved to produce improving solutions. Our MCTS method reduces the challenging LNS neighborhood selection problem to solving a series of LP relaxations. Those LP problems are polynomial-time solvable, ensuring computational tractability. We have conducted comprehensive computational experiments demonstrating significant performance improvements of our proposed algorithms over existing LNS methods, particularly in complex MIP scenarios.

1 Introduction

Combinatorial Optimization plays a crucial role in solving a wide range of complex decision-making problems with discrete structures. At the core of modeling these problems lies Mixed-integer Programming (MIP) [Jünger et al., 2009, Wolsey, 2020], a mathematical framework designed to optimize specific objective functions subject to a set of constraints. Despite the broad applicability of MIP for representing intricate scenarios, the inherent NP-hardness of many such problems poses significant computational challenges, often requiring innovative algorithmic strategies to find feasible solutions within reasonable time.

Recent advancements in solving MIPs involve integrating sophisticated heuristics [Blum and Roli, 2003, Berthold, 2006], Branch-and-Bound (B&B) [Land and Doig, 2010], Cutting Planes [Balas et al., 1993, Marchand et al., 2002] and preprocessing [Achterberg et al., 2020] within modern MIP solvers. These methods collectively aim to enhance the solver's capability to manage large-scale problems. Heuristic methods, in particular, are crucial for providing high-quality solutions rapidly, complementing the slower, more resource-intensive exact methods such as B&B, which, although powerful, may not be practical for very complex problems due to their computational demands.

Submitted to 38th Conference on Neural Information Processing Systems (NeurIPS 2024). Do not distribute.

Among heuristic approaches, Large Neighborhood Search (LNS) [Shaw, 1998] stands out for its effectiveness in exploring extensive solution spaces. LNS iteratively modifies a subset of decision variables within an existing solution to probe various neighborhood structures methodically. However, traditional LNS implementations often struggle with efficiency and adaptability, as they typically rely on manually crafted rules that may not fully leverage the problem's structure and dynamic information to select neighborhoods during the search process [Danna et al., 2005].

In response to these limitations, recent research has explored the integration of machine learning techniques to inform the neighborhood selection process [Sonnerat et al., 2021, Wu et al., 2021a,b, Liu et al., 2022, Huang et al., 2023]. Those approaches leverage available data and train deep learning models to predict the most promising solution neighborhoods during the search. However, deep learning based methods often face generalization issues and may not perform well on heterogeneous datasets, such as those found in MIPLIB, due to the diversity of problem structures and instances.

To address these challenges, this paper introduces two LNS heuristics, designed to optimize dynamic neighborhood selection and systematically explore the solution space for improving MIP solving. The first LNS algorithm, called *Integrality Grip Induced LNS* (IG-LNS), utilizes the concept of integrality grip—a metric that measures the closeness of a localized LP relaxation's solution to integrality—to dynamically select variables for refinement, focusing the search on regions likely to yield significant improvements.

Further advancing this concept, the second algorithm, *MCTS Enhanced IG-LNS* (MIG-LNS), integrates Monte Carlo Tree Search (MCTS) method into the IG-LNS framework. Specifically, at each iteration of LNS, MCTS is firstly executed to assess the outcome of candidate solution neighborhoods by solving the convex relaxations of expanded LNS sub-problems. Then the algorithm will select a promising LNS neighborhood from the expanded MCTS tree, which will be solved by the off-the-shelf MIP solver to produce improved solutions. By simulating various neighborhood configurations and evaluating their potential outcomes through LP relaxations, the MIG-LNS algorithm optimizes the neighborhood selection process to adapt to the evolving landscape of the solution space dynamically.

Main Contributions:

- We propose a new class of LNS heuristic for MIP that leverages the concept of integrality grip to guide neighborhood selection, enhancing the traditional LNS approach.
- We design an efficient MCTS algorithm to improve IG-LNS heuristic. Our MCTS method is more adaptable and efficient by reducing the challenging LNS neighborhood selection problem to solving a series of LP relaxations. These LP problems are polynomial-time solvable, ensuring computational tractability.
- Our methods do not require any machine learning pre-training, making the framework more generalizable and adaptable to broader classes of MIP problems, particularly beneficial for new problems with limited data availability.
- We conduct comprehensive computational experiments demonstrating significant performance improvements of both proposed algorithms over existing LNS methods, particularly in complex MIP scenarios.

The remainder of the paper is organized as follows: Section 2 reviews related works, Section 3 discusses preliminary concepts and foundational algorithms, Section 4 and 5 details the methodologies of IG-LNS and MCTS Improved IG-LNS, including the integration of GNN-based techniques, Section 6 presents experimental results and discussions, and Section 7 concludes with final remarks and future research directions.

2 Related Work

The progress in machine learning (ML) has stimulated increasing research interest in applying ML for solving MIPs. These works can be broadly divided into two categories, *learning auxiliary strategies within MILP solvers* and *learning heuristics*.

The first approach investigates the use of ML to learn to make algorithmic decisions within a MILP solver, which is typically built upon a general B&B framework. The learned policies can be either cheap approximations of existing expensive methods, or more sophisticated strategies that are new to be discovered. Related works include: learning to select branching variables [Khalil et al., 2016, Balcan, 2018, Gasse et al., 2019], learning to select branching nodes [He et al., 2014], learning to select cutting planes [Tang et al., 2020], and learning to optimize the usage of primal heuristics [Khalil et al., 2017, Chmiela et al., 2021].

The *learning heuristics* approach is to learn algorithms to produce primal solutions for MIPs. There are a few works in this direction that typically use ML methods to develop LNS heuristics. Within an LNS scheme, ML models are trained to predict promising solution neighborhoods that are expected to contain improving solutions. Nair et al. [2020] used neural networks to predict partial solutions. The subproblems defined by fixing the predicted partial solutions are solved by a MIP solver. Song et al. [2020] proposed a decomposition-based LNS heuristic. They use imitation learning and reinforcement learning to decompose the set of integer variables into subsets of fixed size. Each subset defines a subproblem and the number of subsets is fixed as a hyperparameter. Sonnerat et al. [2021] proposed a LNS heuristic based on a “learn to destroy" strategy, which frees part of the current solution. The variables to be freed are selected by trained neural networks using imitation learning.

3 Preliminaries

3.1 Mixed-integer Programming

We consider a MIP problem of the form,

$$
\begin{aligned}
(P) \quad \min \quad & \boldsymbol{c}^T \boldsymbol{x} && (1)\\
\text{s.t.} \quad & \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}, && (2)\\
& x_j \in \{0,1\},\ \forall j \in \mathcal{B}, && (3)\\
& x_j \in \mathbb{Z}^+,\ \forall j \in \mathcal{G}, && (4)\\
& x_j \geq 0,\ \forall j \in \mathcal{C}, && (5)
\end{aligned}
$$

where the index set of decision variables $\mathcal{N} := \{1, \ldots, n\}$ is partitioned into $\mathcal{B}, \mathcal{G}, \mathcal{C}$, which are the index sets of binary, general integer and continuous variables, respectively.

3.2 Large Neighborhood Search

Large neighborhood search (LNS) is a refinement heuristic. In general, one iteration of LNS consists of 3 building blocks,

- **Destroy** function: destructs a part of the current solution $\boldsymbol{x}$ by freeing a subset of variables and produces a solution neighborhood $N(\boldsymbol{x})$;
- **Repair** function: rebuilds the destroyed solution, typically by solving a sub-MIP defined by $N(\boldsymbol{x})$. Note: for some cases, the repaired solution can be worse than the destroyed solution;
- **Accept** function: decides whether the new solution should be accepted or rejected.

Given as an input a feasible solution $\bar{\boldsymbol{x}}$, it searches the best feasible solution in neighbourhood of $\bar{\boldsymbol{x}}$ (the size of the neighborhood is a parameter). Once the best feasible solution $\tilde{\boldsymbol{x}}$ in the neighborhood is found, the procedure updates $\bar{\boldsymbol{x}} = \tilde{\boldsymbol{x}}$. The method keeps searching for the best feasible solution in the new neighborhood until the stopping criterion is reached.

3.3 Mont Carlo Tree Search

Monte Carlo Tree Search (MCTS) is an innovative search algorithm widely recognized for its effectiveness in handling complex decision-making processes, particularly in environments characterized by vast decision trees and stochastic outcomes. Initially popularized through its applications in board games like Go, MCTS has diversified its utility across a range of strategic and planning problems in artificial intelligence Browne et al. [2012].

Central to MCTS is its strategic use of random simulations to accumulate statistically meaningful data that informs robust decision-making. This algorithm diverges from conventional exhaustive search methods by preferentially expanding promising moves through an iterative process comprised of four key phases: *selection*, *expansion*, *simulation*, and *backpropagation*.

Algorithm 1 Basic LNS heuristic

Input: instance dataset $\mathcal{P} = \{\boldsymbol{p}_j\}_{j=1}^{M}$
for *instance* $\boldsymbol{p}_j \in \mathcal{P}$ **do**
  initialize the state $\boldsymbol{s}$ with an initial solution $\bar{\boldsymbol{x}}$;
  $\boldsymbol{x}^* \leftarrow \boldsymbol{x}$;
  repeat
    $N(\boldsymbol{x}) \leftarrow destroy(\boldsymbol{x})$;
    $\boldsymbol{x}' \leftarrow repair(N(\boldsymbol{x}))$;
    if $accept(\boldsymbol{x}', \boldsymbol{x})$ **then**
      $\boldsymbol{x} \leftarrow \boldsymbol{x}'$;
    end
    if $f(\boldsymbol{x}) < f(\boldsymbol{x}^*)$ **then**
      $\boldsymbol{x}^* \leftarrow \boldsymbol{x}$;
    end
  until *termination condition is reached*;
end
return $\boldsymbol{x}^*$

- **Selection**: This phase involves traversing the tree from the root to a leaf node by selecting optimal child nodes at each level. The selection strategy is governed by a balance between exploiting nodes with high win ratios and exploring under-visited nodes to ensure a comprehensive search distribution. This is typically guided by a policy like the Upper Confidence Bound (UCB) applied to trees.
- **Expansion**: Upon reaching a leaf node that does not terminate the game, the tree is expanded by adding one or more child nodes. This expansion is contingent upon the possible moves from the current game state, thereby incrementally building the tree structure.
- **Simulation**: Also known as the playout or rollout phase, a simulation is conducted from the newly expanded nodes using a default or random policy to play out the game until a terminal state or predefined depth is reached. These simulations are crucial as they provide insights into the potential outcomes of moves, which are otherwise not assessed through deep analytical computations.
- **Backpropagation**: The outcomes of the simulations are propagated back through the tree, from the leaf nodes up to the root. Each node visited during the simulation phase is updated to reflect the new data, adjusting metrics such as average win rates and visit counts. This iterative updating ensures that the tree gradually evolves to reflect more accurate assessments of potential moves.

The iterative four-step process continues until a termination condition is met, such as a time constraint. Subsequently, the move associated with the highest-reward or most frequently visited child node of the root is executed. The opponent then makes their move, and the cycle recommences with a fresh search tree that reflects the current state of the game.

4 Integrality Grip Enhanced LNS

In this section, we present a novel class of MIP LNS heuristic, *Integrality Grip Induced LNS* (IG-LNS), through a combination of local constraints, LP relaxations and LNS strategies. By focusing on the idea of integrality grip, which refers to the measure of how close the LP relaxation's solution is to being entirely integral, the algorithm effectively narrows the search space and improves solution quality. The algorithm can utilize any local constraint to construct an LP relaxation around the current incumbent solution and employs the fractionality of this solution to dynamically select variables for constructing targeted LNS sub-problems.

4.1 Integrality Grip

The concept of "integrality grip" quantifies the closeness of a solution obtained from the LP relaxation of a sub-MIP around an existing integer solution of the original MIP. The sub-MIP is typically defined by some local constraints (e.g., local branching constraints [Fischetti and Lodi, 2003]) structured from the current integer solution. This metric evaluates how closely the LP relaxation's solution approximates the current integral solution, emphasizing the influence of the integer solution in shaping the LP relaxation. We formally the define the integrality grip as:

$$G(x', x^*) = 1 - \frac{1}{n} \sum_{i=1}^{n} |x'_i - x^*_i| \tag{6}$$

where $x' = (x'_1, x'_2, \ldots, x'_N)$ is the solution vector from the LP relaxation of the sub-MIP, and $x^* = (x^*_1, x^*_2, \ldots, x^*_N)$ is the initial integer solution. The term N denotes the number of variables. $|x'_i - x^*_i|$ calculates the deviation of each component x'_i from the corresponding integer component x^*_i, determining how far each component of the LP solution is from being integral. The integrality grip $G(x', x^*)$ varies from 0 to 1, where 0 indicates a solution with maximum fractionality and 1 denotes a solution where all variables are integral.

Example Given an initial integer solution $x^* = (4, 2, 4, 3)$ of a simple MIP problem, and an LP relaxation of a sub-MIP formed with a local branching constraint around x^*, suppose this LP relaxation produces a solution vector $x' = (3.5, 1.8, 4.0, 2.9)$. Substituting these values into the integrality grip formula yields an integrality grip of 0.8, indicating that the solution x' is relatively close to the integral values specified by x^*.

Integrality grip is pivotal in determining the focus areas for the LNS, as it identifies where small, targeted modifications can potentially lead to substantial improvements in the overall solution quality.

4.2 The IG-LNS Heuristc

Now we present our IG-LNS heuristic, which consists of the following components.

- **Building LP Relaxation with Local Constraints** The IG-LNS algorithm starts by selecting a current integer solution, $\bar{\boldsymbol{x}}$, around which the LP relaxation is constructed. This relaxation incorporates a local constraint that limits the search space to a neighborhood defined by a Hamming distance from $\bar{\boldsymbol{x}}$. The constraint is formulated as:

$$\Delta(\boldsymbol{x}', \bar{\boldsymbol{x}}) = \sum_{i \in J} |x_i - \bar{x}_i| \leq k \tag{7}$$

  where J is the set of indices of integer variables. Parameter k controls the neighborhood size.

- **The Upper Bound of Local Constraint** Let $\boldsymbol{x}'$ be the optimal LP solution of the original MIP model without any local constraint, and let k' be the value of the left-hand side of the local constraint evaluated using $\boldsymbol{x}'$. Specifically, k' is computed by

$$k' = \Delta(\boldsymbol{x}', \bar{\boldsymbol{x}}). \tag{8}$$

  If parameter k is greater than or equal to value k', the LP solution is likely to remain unchanged after adding the local constraint. Consequently, k' serves as an upper bound for k. We can therefore parametrize k as

$$k = \phi\, k', \tag{9}$$

  where $\phi \in [0, 1]$. Therefore, the value of ϕ should be determined when building a local LP relaxation constrained by (7).

- **Solving the LP Relaxation** The local LP relaxation is solved to produce an LP solution x', which is evaluated for its integrality grip. The fractional components of x' indicate the variables that are most challenging to integrate, providing a direct method to target the subsequent LNS steps.

- **Constructing and Solving the LNS Sub-MIP** Using the fractional variables identified from x', a sub-MIP is constructed. This sub-MIP selectively "destroys" and "repairs" parts of x^* by allowing these variables to vary, while keeping others fixed. This targeted disruption aims to explore the solution space more deeply where the LP relaxation indicates potential for improvement.

- **Iterative Refinement** The process iterates with the newly found solutions from the LNS sub-MIPs being used to redefine the neighborhood in the LP relaxation, continually refining the approach towards an optimal solution. Each iteration recalibrates the parameter k based on the integrality grip and the outcomes of the LNS, adjusting the balance between exploration and exploitation.

5 Exploring Integrality Grip by MCTS Planning

To further enhance the IG-LNS algorithm, in this section, we study how to explore the integrality grip to improve the LNS search by MCTS planning. This approach is designed to dynamically explore and optimize the neighborhood through a guided search within the action space. We will first model the LNS into a Markov Decision Process (MDP) and then introduce how to efficiently integrate MCTS planning into the IG-LNS algorithm.

5.1 Markov Decision Process Modeling of LNS

Given an MIP instance with an initial feasible solution. The procedure can be formulated as a Markov Decision Process(MDP), wherein at each step, a LNS neigbhorhood is selected to build a sub-MIP and an off-the-shelf MIP solver is called to solve the LNS sub-problem.

- **State**($\mathcal{S}$): The state consists of the solution status of sub-MIP and the encoded state of any MILP instance. After each LNS step, the resulting state can generally be classified into one of the four groups, depending on the status of the sub-MIP and the objective value, shown in Figure 1.
- **Action**(a): The set of possible actions consists of the portion d of variables to be destroyed from the current solution, where $d \in [0, 1]$. The number of destroyed variables n will be $n = d * N$, where N is the number of integer variables in the MIP model.
- **Policy**(π): The policy maps a state to an action in action space.
- **Reward**(r): By applying the updated k, the next iteration of LNS will be executed with time limit t_{limit}. Then the solution status of sub-MIP will be collected to create the next state. The reward will be formulated according to both the computing time and the quality of the new solution.

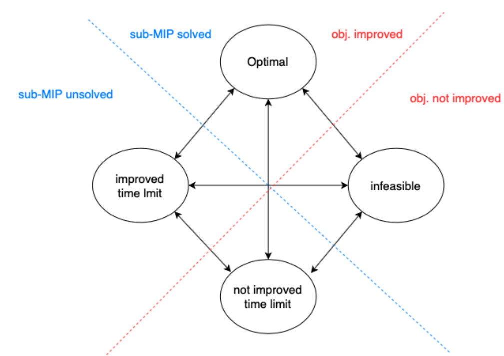

Figure 1: Solution states of sub-MIP in MDP of LNS

The definition above is just one possibility to build a MDP for this problem. In fact, how to define a compact MDP (e.g. state, reward) is crucial for constructing efficient LNS algorithms.

5.2 MCTS Integration into IG-LNS

MCTS is applied to the action space determined by the neighborhood size k, which is crucial for controlling the breadth of the search of LNS. In our method, we define the action a to determine the value of ϕ within the range $[0, 1]$. As introduced in (9), ϕ dictates the proportion of integer variables considered in each iteration, such that $k = \phi \cdot k'$, where k' represents the upper bound of parameter k computed by (8).

MCTS Expansion Strategy The tree in MCTS is expanded by selecting actions based on the proportion of variables to be included in k. At each node of the MCTS, an LNS sub-MIP is constructed and only the LP relaxation of this sub-MIP is solved. This strategy ensures that each MCTS iteration remains computationally efficient, as solving the LP relaxation is polynomially tractable, thereby allowing rapid generation of the tree.

Reward Function in MCTS The reward function in the MCTS framework is composed of three main components:

- **Objective Improvement:** This component measures the improvement in the objective value of the LP relaxation of the LNS sub-MIP compared to the previous iterations.
- **Solving Time:** The computational time required to solve the LP relaxation of the LNS sub-MIP is considered, emphasizing efficiency.
- **Integrality Grip:** Assesses how closely the LP solution approximates an integral solution, providing insights into the quality of the LP solution in terms of feasibility.

These components collectively guide the search towards areas of the action space that potentially yield significant improvements in solution quality and computational efficiency. The balance between exploration and exploitation is managed through the selection and backpropagation steps of the MCTS, ensuring that the algorithm progressively refines the neighborhood size K towards optimal settings.

5.3 Integrality Grip Tarlored MCTS

Our implementation of MCTS is customized to explore the action space of the neighborhood size k.

- **Selection Phase:** During the selection phase, the MCTS algorithm assesses each node by exploring potential actions based on their historical success and exploratory value using the Upper Confidence Bound (UCB) strategy. This phase is critical for navigating the expansive solution space, aiming to incrementally approach the most promising areas that could yield significant improvements in the heuristic's performance.
- **Expansion Phase:** Upon selecting a node, the expansion phase involves introducing new child nodes into the tree. Each node corresponds to a different action a, which variably adjusts the k parameter of LNS neighborhood. This strategic expansion allows the heuristic to probe various configurations of local constraints, enriching the diversity of solutions explored and identifying potentially optimal neighborhood sizes.
- **Simulation Phase:** The simulation phase at each node involves solving only the LP relaxation of the LNS sub-MIP associated with the current node configuration. This focused simulation is key to maintaining the method's efficiency, as solving the LP relaxation is polynomially solvable, ensuring that the algorithm can rapidly evaluate a vast number of potential configurations without excessive computational costs.
- **Backpropagation Phase:** Following the simulation, results are used to update the tree during the backpropagation phase. This process adjusts the statistical values of the nodes, from the expanded node back to the root, based on the outcome of the LP relaxation. These updates refine the decision-making process, enhancing the algorithm's capability to make more informed selections in subsequent iterations.

By integrating MCTS into the IG-LNS heuristic, we obtain an extended LNS algorithm, namely *MCTS-enhanced IG-LNS* (MIG-LNS). This extension provides a more adaptive and targeted approach to managing the LNS neighborhood. This enhancement is expected to lead to faster convergence to high-quality solutions, particularly in complex combinatorial optimization problems where traditional LNS might stall or converge prematurely.

6 Experiments

In this section, we present our experimental results over three MIP benchmarks. We compare different settings of our approach against the original LNS algorithm, using SCIP [Gerald et al., 2020] as the underlying MILP solver.

6.1 Dataset

MIP Benchmark We first apply our framework to MIPLIB [Gleixner et al., 2021], the most well-konwn state-of-the-art MIP benchmark. The MIPLIB instances are selected from the following process: We want to collect a reasonable intermediate solution as the starting point for LNS search. To get this, we run the SCIP 7.0.1 solver to solve the root node of each instance in MIPLIB2017 with a time limit of 1 hour, and filter out all the instances that have reached to optimal on root node or still can not find a solution after 1 hour. By this process, we filter out some instances that are too easy or too difficult to find a starting solution for LNS search, which results in 126 instances. For each instance, an initial feasible solution is required to start the LNS heuristic. We use an intermediate

Table 1: Statistics(mean, standard deviation (STD), maximum, minimum) on final primal gap and final primal integral over SC dataset.

	Primal Gap			Primal Integral		
Algorithm	Mean ± STD	Max	Min	Mean ± STD	Max	Min
Default SCIP	5.060 ± 16.133	101.100	0.0	19.935 ± 9.013	35.749	6.493
LB	18.705 ± 23.765	58.966	0.0	19.901 ± 11.168	36.780	4.045
Random-LNS	11.343 ± 36.439	209.764	0.0	24.703 ± 8.826	42.156	10.759
GNN-LNS	0.869 ± 0.696	2.643	0.0	15.214 ± 10.364	33.954	4.907
IG-LNS	0.241 ± 0.497	**2.217**	0.0	13.071 ± 10.331	**28.613**	3.945
MIG-INS	**0.203 ± 0.492**	2.486	0.0	**12.664 ± 9.832**	32.973	**3.911**

Table 2: Statistics(mean, standard deviation (STD), maximum, minimum) on final primal gap and final primal integral over GISP dataset.

	Primal Gap			Primal Integral		
Algorithm	Mean ± STD	Max	Min	Mean ± STD	Max	Min
Default SCIP	8.329 ± 30.008	163.106	0.0	6.995 ± 9.696	37.301	0.008
LB	12.809 ± 21.585	88.829	0.0	8.901 ± 10.761	40.905	0.011
Random-LNS	8.534 ± 30.016	163.122	0.0	5.883 ± 8.394	37.273	0.010
GNN-LNS	7.896 ± 30.053	46.775	0.0	4.990 ± 10.598	41.330	**0.005**
IG-LNS	7.728 ± 34.153	87.857	0.0	4.813 ± 8.925	40.055	0.006
MIG-LNS	**5.949 ± 26.410**	**45.140**	0.0	**4.385 ± 8.581**	**29.705**	0.006

solution found by SCIP, typically the best solution obtained by SCIP at the end of the root node computation, i.e., before branching.

SC and GISP Benchmarks In practice, there are many specific MIP applications where instances from the same class of problem are formulated and solved repeatedly. Therefore, in order to demonstrate how effective our approach is on those homogeneous problems, we conduct further computational experiments to two classes of MIP benchmarks: set covering (SC) [Balas and Ho, 1980] and generalized independent set problem (GISP) [Hochbaum and Pathria, 1997, Colombi et al., 2017]. For SC benchmark, we generate 200 instances with 5000 rows and 2000 columns. For GISP, we use the public dataset from Chmiela et al. [2021].

6.2 Algorithmic Comparisons

We conduct experiments to compare the following algorithms:

- **SCIP**, the SCIP solver with default setting;
- **Random-LNS**, the LNS baseline algorithm;
- **LB**, the Local Branching heuristic [Fischetti and Lodi, 2003];
- **GNN-LNS**, the most commonly used state-of-the-art GNNs [Sonnerat et al., 2021] that have been trained for LNS neighborhood predictionss;
- **IG-LNS**, basic version of our proposed Integrality Grip Enhanced LNS;
- **MIG-LNS**, our extended IG-LNS improved from exploring integrality grip by MCTS planning.

All the algorithms use SCIP as the underlying MIP solver.

We use the *primal integral* [Berthold, 2013] and standard *primal gap* to measure the performance of the compared MIP algorithms. Detailed information and formulations for computing the two metrics can be found in Appendix A.1.

6.3 Experimental Results

We evaluate the compared algorithms over the three benchmarks. The results of all the algorithms are shown in Table 1, 2, 3.

Table 3: Statistics(mean, standard deviation (STD), maximum, minimum) on final primal gap and final primal integral over MIPLIB dataset.

	Primal Gap			Primal Integral		
Algorithm	Mean $\pm$ STD	Max	Min	Mean $\pm$ STD	Max	Min
Default SCIP	8.257 $\pm$ 29.894	163.024	0.0	6.912 $\pm$ 9.654	37.222	0.001
LB	12.750 $\pm$ 21.530	88.786	0.0	8.880 $\pm$ 10.722	40.848	0.001
Random-LNS	8.485 $\pm$ 29.959	163.0241	0.0	5.86516 $\pm$ 8.367	37.245	0.001
GNN-LNS	7.865 $\pm$ 30.005	46.729	0.0	4.969 $\pm$ 10.574	41.305	0.001
IG-LNS	7.692 $\pm$ 34.110	87.826	0.0	10.891 $\pm$ 8.894	40.024	0.001
MIG-LNS	**5.917 $\pm$ 26.363**	**45.114**	0.0	**4.373 $\pm$ 8.558**	**29.687**	0.001

From the results, our IG-LNS and MIG-LNS algorithm presents the best heuristic behavior over all the compared algorithms in terms of both primal integral and primal gap, showing that the proposed local LP relaxation based LNS method is able to produce promising and robust LNS neighborhoods by gripping the integrality of candidate solutions within the neighborhood. The results of MIG-LNS also demonstrate that our MCTS algorithm achieves a better trade-off between exploitation and exploration of the solution space during LNS search. The LNS behavior of our approach is robuster than the compared baselines by improving both the feasibility and the objective of solutions within the selected neighborhoods.

A significant advantage of our proposed methods is that they do not require any machine learning pre-training. This feature enhances the generalizability and adaptability of our framework to a broader range of MIP problems. It is particularly beneficial for new problems with limited data availability, as our methods can be applied directly without the need for extensive training on large datasets.

7 Conclusion

In this work, we introduce the *Integrality Grip Induced Large Neighborhood Search* (IG-LNS) algorithm, a novel class of LNS heuristic for Mixed-Integer Programming (MIP). Our approach leverages the concept of integrality grip to dynamically guide neighborhood exploration, thereby enhancing the effectiveness of the classic LNS method. The integrality grip measures how closely an LP relaxation's solution approximates integrality, enabling more targeted and efficient searches within the solution space.

Building upon the IG-LNS framework, we integrate an efficient Monte Carlo Tree Search (MCTS) algorithm to further refine and improve the heuristic. The MCTS method addresses the challenging problem of LNS neighborhood selection by reducing it to solving a series of LP relaxations. These LP problems are polynomial-time solvable, ensuring computational tractability and making the search process more adaptable and efficient. We conduct comprehensive computational experiments to validate our approaches, demonstrating significant performance improvements over existing LNS methods.

While the IG-LNS algorithm with MCTS planning demonstrates significant improvements in solving MIP problems, there are still limitations and open questions for future research. For example, although the MCTS framework improves neighborhood selection efficiency, the computational overhead of maintaining and updating the tree structure can be substantial for very large-scale problems. Future research could focus on enhancing the scalability of the MCTS algorithm by exploring parallelization techniques or hybrid approaches that combine MCTS with other metaheuristics. Another promising direction is to investigate adaptive mechanisms that can dynamically adjust the parameters of the integrality grip and MCTS based on problem characteristics. Finally, while our approach does not require pre-training, exploring the integration of lightweight learning models to enhance decision-making processes within the heuristic could offer additional performance gains without compromising adaptability.

References

Tobias Achterberg, Robert E Bixby, Zonghao Gu, Edward Rothberg, and Dieter Weninger. Presolve reductions in mixed integer programming. *INFORMS Journal on Computing*, 32(2):473–506, 2020.

Egon Balas and Andrew Ho. Set covering algorithms using cutting planes, heuristics, and subgradient optimization: a computational study. In *Combinatorial Optimization*, pages 37–60. Springer, 1980.

Egon Balas, Sebastián Ceria, and Gérard Cornuéjols. A lift-and-project cutting plane algorithm for mixed 0–1 programs. *Mathematical programming*, 58(1):295–324, 1993.

Maria-Florina others Balcan. Learning to branch. In *International conference on machine learning*, pages 344–353. PMLR, 2018.

Timo Berthold. *Primal heuristics for mixed integer programs*. PhD thesis, Zuse Institute Berlin (ZIB), 2006.

Timo Berthold. Measuring the impact of primal heuristics. *Operations Research Letters*, 41(6): 611–614, 2013.

Christian Blum and Andrea Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM computing surveys (CSUR)*, 35(3):268–308, 2003.

Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43, 2012.

Antonia Chmiela et al. Learning to schedule heuristics in branch-and-bound. *arXiv preprint arXiv:2103.10294*, 2021.

Marco Colombi et al. The generalized independent set problem: Polyhedral analysis and solution approaches. *European Journal of Operational Research*, 260(1):41–55, 2017.

Emilie Danna, Edward Rothberg, and Claude Le Pape. Exploring relaxation induced neighborhoods to improve mip solutions. *Mathematical Programming*, 102(1):71–90, 2005.

Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric. *arXiv preprint arXiv:1903.02428*, 2019.

Matteo Fischetti and Andrea Lodi. Local branching. *Mathematical programming*, 98(1-3):23–47, 2003.

Gerald Gamrath et al. The scip optimization suite 7.0. 2020.

Maxime Gasse et al. Exact combinatorial optimization with graph convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 15554–15566, 2019.

Gamrath Gerald et al. The SCIP Optimization Suite 7.0. ZIB-Report 20-10, Zuse Institute Berlin, March 2020. URL `http://nbn-resolving.de/urn:nbn:de:0297-zib-78023`.

Ambros Gleixner, Gregor Hendel, Gerald Gamrath, Tobias Achterberg, Michael Bastubbe, Timo Berthold, Philipp Christophel, Kati Jarck, Thorsten Koch, and Jeff Linderoth. Miplib 2017: data-driven compilation of the 6th mixed-integer programming library. *Mathematical Programming Computation*, pages 1–48, 2021.

He He et al. Learning to search in branch and bound algorithms. *Advances in neural information processing systems*, 27:3293–3301, 2014.

Dorit S Hochbaum and Anu Pathria. Forest harvesting and minimum cuts: a new approach to handling spatial constraints. *Forest Science*, 43(4):544–554, 1997.

Taoan Huang, Aaron M Ferber, Yuandong Tian, Bistra Dilkina, and Benoit Steiner. Searching large neighborhoods for integer linear programs with contrastive learning. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 13869–13890. PMLR, 23–29 Jul 2023. URL `https://proceedings.mlr.press/v202/huang23g.html`.

Michael Jünger, Thomas M Liebling, Denis Naddef, George L Nemhauser, William R Pulleyblank, Gerhard Reinelt, Giovanni Rinaldi, and Laurence A Wolsey. *50 Years of integer programming 1958-2008: From the early years to the state-of-the-art*. Springer Science & Business Media, 2009.

Elias Khalil et al. Learning to branch in mixed integer programming. *Proceedings of the AAAI Conference on Artificial Intelligence*, 30(1), Feb. 2016. URL `https://ojs.aaai.org/index.php/AAAI/article/view/10080`.

Elias B Khalil et al. Learning to run heuristics in tree search. In *IJCAI*, pages 659–666, 2017.

Ailsa H Land and Alison G Doig. An automatic method for solving discrete programming problems. In *50 Years of Integer Programming 1958-2008*, pages 105–132. Springer, 2010.

Defeng Liu et al. Learning to search in local branching. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36(4):3796–3803, Jun. 2022. doi: 10.1609/aaai.v36i4.20294. URL `https://ojs.aaai.org/index.php/AAAI/article/view/20294`.

Stephen Maher et al. PySCIPOpt: Mathematical programming in python with the SCIP optimization suite. In *Mathematical Software – ICMS 2016*, pages 301–307. Springer International Publishing, 2016. doi: 10.1007/978-3-319-42432-3_37.

Hugues Marchand, Alexander Martin, Robert Weismantel, and Laurence Wolsey. Cutting planes in integer and mixed integer programming. *Discrete Applied Mathematics*, 123(1-3):397–446, 2002.

Vinod Nair et al. Solving mixed integer programs using neural networks. *arXiv preprint arXiv:2012.13349*, 2020.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, and Luca Antiga. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32: 8026–8037, 2019.

Paul Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In *International conference on principles and practice of constraint programming*, pages 417–431. Springer, 1998.

Jialin Song et al. A general large neighborhood search framework for solving integer linear programs. *arXiv preprint arXiv:2004.00422*, 2020.

Nicolas Sonnerat et al. Learning a large neighborhood search algorithm for mixed integer programs. *arXiv preprint arXiv:2107.10201*, 2021.

Yunhao Tang et al. Reinforcement learning for integer programming: Learning to cut. In *International Conference on Machine Learning*, pages 9367–9376. PMLR, 2020.

Laurence A Wolsey. *Integer programming*. John Wiley & Sons, 2020.

Yaoxin Wu, Wen Song, Zhiguang Cao, and Jie Zhang. Learning large neighborhood search policy for integer programming. *Advances in Neural Information Processing Systems*, 34:30075–30087, 2021a.

Yaoxin Wu et al. Learning large neighborhood search policy for integer programming. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 30075–30087. Curran Associates, Inc., 2021b. URL `https://proceedings.neurips.cc/paper_files/paper/2021/file/fc9e62695def29ccdb9eb3fed5b4c8c8-Paper.pdf`.

A Appendix

A.1 Metrics for Measuring the Performance of MIP Algorithms

The *primal integral* was originally proposed to measure the performance of primal heuristics for solving mixed-integer programs. The metric takes into account both the quality of solutions and the computing time spent to find those solutions during the solving process. To define the primal integral, we first consider a *scaled primal gap function* $p(t)$ as a function of time, defined as

$$p(t) = \begin{cases} 1, & \text{if no incumbent until time } t, \\ \bar{\gamma}(\tilde{\boldsymbol{x}}(t)), & \text{otherwise,} \end{cases}$$

where $\tilde{\boldsymbol{x}}(t)$ is the incumbent solution at time t, and $\bar{\gamma}(\cdot) \in [0, 1]$ is the *scaled primal gap*

$$\bar{\gamma}(\tilde{\boldsymbol{x}}) = \frac{|f(\tilde{\boldsymbol{x}}_{\text{opt}}) - f(\tilde{\boldsymbol{x}})|}{|f(\tilde{\boldsymbol{x}}_{\text{opt}}) - f(\tilde{\boldsymbol{x}}_{\text{init}})|},$$

where $f(\tilde{\boldsymbol{x}})$ denotes the objective value given solution $\tilde{\boldsymbol{x}}$, $\tilde{\boldsymbol{x}}_{\text{opt}}$ is either the optimal solution or the best one known for the instance and $\tilde{\boldsymbol{x}}_{\text{init}}$ is the initial solution.

The standard *primal gap* without scaling is defined as

$$\gamma(\tilde{\boldsymbol{x}}) = \frac{|f(\tilde{\boldsymbol{x}}_{\text{opt}}) - f(\tilde{\boldsymbol{x}})|}{|f(\tilde{\boldsymbol{x}}_{\text{opt}})|}.$$

Let $t_{\text{max}} > 0$ be the time limit for executing the heuristic. The primal integral measure is then defined as

$$P(t_{\text{max}}) = \int_0^{t_{\text{max}}} p(t)\, dt.$$

A.2 Experimental Settings and Hyperparameters

For training GNN models, we used the focal loss as the loss function. For tuning the learning rate, we have experimented different learning rates from 10^{-5} to 10^{-1} and have chosen a learning rate of 10^{-4}. We trained the model with a limit of 500 epochs.

For the LNS hyperparameters, we set a time limit of 3 seconds for each LNS iteration for all the compared algorithms. The global time limit for all algorithms is set to 3600 seconds.

For the baselines, we compare the performance of our extended GNNs against state-of-the-art message-passing based GNNs used in other works and also against classic LNS algorithm and default SCIP solver baseline. We are aware of the fact that there are more learning-based LNS baselines in the literature which could be potentially added to the list for a fair comparison. However, some of existing works have not revealed their code to public and it is challenging to fairly implement their models.

Our code is written in Python 3.8 and we use Pytorch 1.7.1 Paszke et al. [2019], Pytorch Geometric 2.0.2 Fey and Lenssen [2019], PySCIPOpt 3.1.1 Maher et al. [2016], SCIP 7.01 Gamrath et al. [2020] for developing our models and sovling MIPs. Our experiments were conducted on 2.70 GHz Intel Xeon Gold 6258R machines with 8 cores.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: We have carefully write our main claims in the abstract and introduction to reflect the paper's contributions and scope.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We have discussed the limitations of the work and also provide perspectives for future research.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory Assumptions and Proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [NA]

Justification: This paper does not provide theoretical results.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental Result Reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: We have provided all the information needed to reproduce the main experimental results of the paper. Code is provided and detailed hyperparameter settings can be found in Appendix.

Guidelines:

- The answer NA means that the paper does not include experiments.
- If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: We have provided code and data.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental Setting/Details**

Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

Answer: [Yes]

Justification: We have provided those experimental details in the paper and code package.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment Statistical Significance**

Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [Yes]

Justification: We have reported statistical details in the report of the experimental results.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)

- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments Compute Resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: We have provided those detailed information on the computer resources needed to reproduce the experiments.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code Of Ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: We have read NeurIPS Code of Ethics and we do not find any concern related to our submission.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader Impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: We have discussed the potential societal impacts of our work.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.
    - If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.

- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: We do not find any of those risk listed above for this submission.

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [NA]

    Justification: This submission does not use existing assets.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.

- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New Assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [NA]

    Justification: The paper does not release new assets.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and Research with Human Subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: The paper does not involve crowdsourcing nor research with human subjects. All the data are generated from simulator.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional Review Board (IRB) Approvals or Equivalent for Research with Human Subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.