Hypergraph Neural Networks Accelerate MUS Enumeration

Hiroya Ijima **Koichiro Yawata**
Hitachi, Ltd.

Abstract

Enumerating Minimal Unsatisfiable Subsets (MUSes) is a fundamental task in constraint satisfaction problems (CSPs). Its major challenge is the exponential growth of the search space, which becomes particularly severe when satisfiability checks are expensive. Recent machine learning approaches reduce this cost for Boolean satisfiability problems but rely on explicit variable-constraint relationships, limiting their application domains. This paper proposes a domain-agnostic method to accelerate MUS enumeration using Hypergraph Neural Networks (HGNNs). The proposed method incrementally builds a hypergraph with constraints as vertices and MUSes enumerated until the current step as hyperedges, and employs an HGNN-based agent trained via reinforcement learning to minimize the number of satisfiability checks required to obtain an MUS. Experimental results demonstrate the effectiveness of our approach in accelerating MUS enumeration, showing that our method can enumerate more MUSes within the same satisfiability check budget compared to conventional methods.

1 INTRODUCTION

Minimal Unsatisfiable Subset (MUS) enumeration is a fundamental task of constraint satisfaction problems (CSPs). An MUS is a subset of constraints that cannot be satisfied simultaneously, and removing any constraint from the subset renders it satisfiable. Identifying MUSes is crucial for various applications, such as conflict identification in product configurations (Herud et al., 2022), inconsistency detection in software requirements (Bendík, 2017) and extracting irreducible infeasible constraint sets (IISes) in optimization problems (Chinneck and Dravnieks, 1991). MUSes are expected to play an increasingly important role with the recent advancements in AI technologies, as they can contribute to explainability for machine learning (Ignatiev et al., 2019, 2020). Similarly, Maximal Satisfiable Subsets (MSSes) are important. An MSS is a subset that can be satisfied simultaneously, and adding any constraint to the subset renders it unsatisfiable. Since MSSes correspond to Pareto-optimal combinations of constraints, their identification is equally important.

A major challenge of MUS/MSS enumeration is the high computational cost. The search space of MUS enumeration grows exponentially with respect to the number of constraints, requiring numerous satisfiability checks. This issue becomes particularly critical when the satisfiability checks are expensive, such as in cases involving complex simulations or large-scale optimization problems. Efficiently enumerating MUSes/MSSes while minimizing the number of checks is therefore essential for practical applications.

For efficient MUS/MSS enumeration, various algorithms have been proposed. The main approach for MUS/MSS extraction is the deletion-based approach, which iteratively shrinks an unsatisfiable subset by removing constraints until an MUS is obtained, or grows a satisfiable subset of constraints by adding constraints until an MSS is obtained (Chinneck and Dravnieks, 1991; Bakker et al., 1993). Since enumerating all MUSes is computationally infeasible in many practical cases, online enumeration methods, which output MUSes/MSSes as they are found, are often employed. Various online enumeration algorithms have been proposed to improve the efficiency of MUS/MSS enumeration by effectively mapping the explored search space and selecting unexplored subsets to shrink/grow (Previti and Marques-Silva, 2013; Liffiton and Malik, 2013; Liffiton et al., 2016; Bendík et al., 2016, 2018).

Recently, machine learning models have been applied to accelerate MUS enumeration. For example, Neu-

Proceedings of the 29th International Conference on Artificial Intelligence and Statistics (AISTATS) 2026, Tangier, Morocco. PMLR: Volume 300.

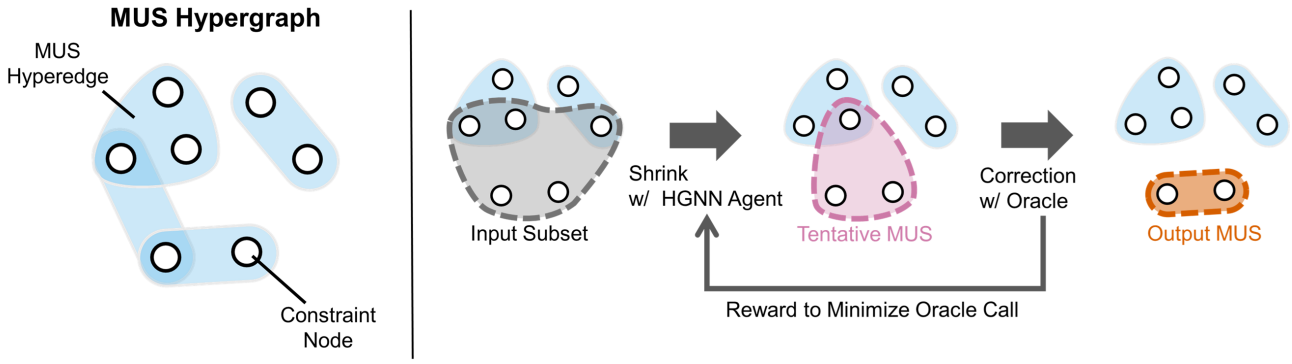

Figure 1: Overview of HyMUSE. HyMUSE is a domain-agnostic method to accelerate MUS/MSS enumeration using Hypergraph Neural Networks (HGNNs). HyMUSE constructs a hypergraph whose vertices are constraints and whose hyperedges are MUSes and MCSes (only MUSes are illustrated in this figure). The HGNN-based agent takes the hypergraph of MUSes/MCSes enumerated until the current step and the subset of constraints to shrink/grow as input, and outputs a tentative MUS/MSS candidate. The candidate is then corrected to a valid MUS/MSS. The agent is trained via reinforcement learning with reward designed to minimize the number of calls to the satisfiability checking oracle required for the correction.

roMUSX (Moriyama et al., 2023) represents variable-constraint relationships in Boolean satisfiability problems as graphs and utilizes graph neural networks (GNNs) to reduce the number of satisfiability checks in deletion operations of MUS extraction by trimming unsatisfiable cores. GRAPE-MUST (Lymperopoulos and Liu, 2024) also employs GNNs to narrow down the search space of MUS enumeration. These methods have shown promising results in accelerating MUS enumeration for Boolean satisfiability problems.

However, existing machine learning-based methods have limitations in their applicability. They rely on explicit variable-constraint relationships, which are not available in many practical CSPs involving real-valued variables or higher-order constraints. For such CSPs, constructing variable-constraint graphs is difficult, limiting the applications of existing machine learning-based methods to other domains.

In this paper, we propose a novel domain-agnostic method **HyMUSE**, which accelerates MUS/MSS enumeration using Hypergraph Neural Networks (HGNNs). Instead of a variable–constraint relation graph, our method constructs a hypergraph representing the constraint–MUS/MSS relations obtained during the enumeration process, with constraints as nodes and incrementally identified MUSes/MCSes as hyperedges. Given the MUS/MCS hypergraph as input, an HGNN-based agent iteratively deletes/adds constraints to produce a tentative MUS/MSS candidate, which is then corrected to a valid MUS/MSS. The agent is trained via reinforcement learning to minimize the number of satisfiability checks required for the correction. Since the MUS/MCS hypergraph is available in any CSPs during MUS/MSS enumeration, our method is domain-agnostic.

Experimental results demonstrate the effectiveness of our approach in accelerating MUS/MSS enumeration. By integrating our method with existing enumeration algorithms, more MUSes/MSSes can be enumerated within the same satisfiability check budget. The proposed method also exhibits generalization capabilities across different problem distributions from the training data, showing its potential for practical applications.

The main contributions of this paper are as follows:

- We propose a domain-agnostic method for MUS/MSS enumeration that uses hypergraph neural networks (HGNNs) to exploit constraint-MUS/MSS relationships, without relying on the explicit variable-constraint relationships required by existing machine learning approaches.
- Our method employs an HGNN-based agent that selects constraints to delete/add during shrink/grow operations, producing a tentative MUS/MSS candidate that is then corrected into a valid solution with few satisfiability checks.
- The agent is trained via reinforcement learning to minimize the number of satisfiability checks required for the correction.
- Experiments show that our method finds more MUSes/MSSes within the same satisfiability-check budget than conventional methods and generalizes across different problem types.

2 RELATED WORK

2.1 MUS Enumeration

Minimal Unsatisfiable Subset (MUS) enumeration is a task of identifying MUSes in a given set of constraints. MUS correspond to irreducible explanations of unsatisfiability, and thus MUS enumeration plays a crucial role in various applications (Herud et al., 2022; Bendík, 2017; Ignatiev et al., 2019, 2020; Chinneck and Dravnieks, 1991). Since the candidate subsets of MUS are 2^C, where C is the set of constraints, the search space grows exponentially with the number of constraints, making MUS enumeration computationally expensive. This challenge becomes particularly severe when satisfiability checks are expensive, therefore, minimizing the number of checks is desirable.

To enumerate MUSes efficiently, various algorithms have been proposed. A widely used approach for single MUS/MSS extraction is the deletion-based approach, which iteratively shrinks an unsatisfiable set of constraints by removing constraints until an MUS is obtained, or grows a satisfiable set of constraints by adding constraints until an MSS is obtained (Chinneck and Dravnieks, 1991; Bakker et al., 1993). In many practical scenarios, enumerating all MUSes is computationally infeasible, so online enumeration methods are commonly employed, where MUSes/MSSes are output one by one as they are found. The representative algorithm of this approach is MARCO (Previti and Marques-Silva, 2013; Liffiton and Malik, 2013; Liffiton et al., 2016), which grows satisfiable sets to MSSes and shrinks unsatisfiable sets to MUSes iteratively while mapping the explored search space using blocking conjunctive normal form (CNF) clauses. Other notable algorithms include TOME (Bendík et al., 2016) and ReMUS (Bendík et al., 2018), which further improve the efficiency of MUS enumeration by refining the heuristics for selecting subsets to shrink.

Our proposed method can be integrated with these existing algorithms by replacing part of the shrink/grow operations with actions performed by machine learning models.

2.2 Machine Learning for MUS Enumeration

Recent advances in machine learning have led to the development of methods that use machine learning models to improve CSP solving, including search heuristics, variable assignment, and satisfiability prediction (Popescu et al., 2022; Guo et al., 2023). Many of these methods employ graph neural networks (GNNs) to capture the structure of variable-constraint relationships in CSPs (Bünz and Lamm, 2017; Selsam et al., 2018; Selsam and Bjørner, 2019; Wang et al., 2021; Li and Si, 2022; Tönshoff et al., 2022). For MUS enumeration, GRAPE-MUST (Lymperopoulos and Liu, 2024) represents literal-clause relationships in CNFs as graphs and utilizes GNNs to narrow down the search space of MUS enumeration. NeuroMUSX (Moriyama et al., 2023) also employs GNNs to reduce the number of satisfiability checks in deletion operations of MUS extraction by trimming unsatisfiable cores. These methods successfully accelerate MUS enumeration for Boolean satisfiability problems. However, for many types of CSPs, it is difficult to construct variable-constraint graphs, making it challenging to apply existing machine learning-based methods to other domains.

In contrast, our proposed method is domain-agnostic and can be applied to any CSPs, as it does not rely on explicit variable-constraint relationships.

2.3 Hypergraph Neural Networks

Hypergraph Neural Networks (HGNNs) are a class of neural networks designed to operate on hypergraphs. Hypergraphs are generalizations of graphs where edges, called hyperedges, are allowed to connect any number of vertices, that is, a hyperedge is a subset of vertices. Many real-world data can be naturally represented as hypergraphs, such as social networks, biological networks, and recommendation systems, so HGNNs have been widely studied and applied in various domains (Kim et al., 2024; Yang and Xu, 2025).

There have been several types of HGNNs proposed, including methods of expanding hypergraphs to corresponding ordinary graphs and applying GNNs (Yadati et al., 2019; Huang and Yang, 2021), spectral convolutional methods that generalize graph convolution to hypergraphs (Feng et al., 2019; Bai et al., 2021), message-passing methods that directly define message-passing operations on hypergraphs (Dong et al., 2020; Arya et al., 2020), and methods that utilize multi-head attention (Zhang et al., 2019; Bai et al., 2021; Arya et al., 2020; Chien et al., 2021). Our proposed method employs multi-head attention-based HGNNs following AllSetTransformer (Chien et al., 2021).

2.4 Reinforcement Learning for CSP

Reinforcement learning (RL) is a framework for sequential decision-making in which an agent learns a policy to maximize rewards through interactions with an environment. This paradigm has recently been applied to learn search heuristics for solving CSPs (Yolcu and Póczos, 2019; Kurin et al., 2020; Tönshoff et al., 2022; Li et al., 2024; Zhai and Ge, 2025).

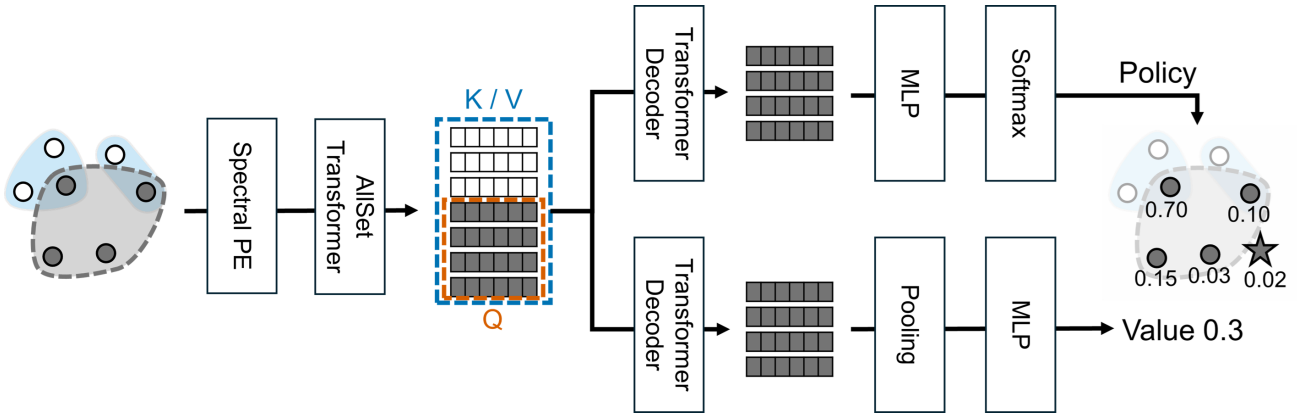

Figure 2: Architecture of Hypergraph Neural Networks. The input is MUS/MCS hypergraph and the subset of constraints to be shrink/grow shown as the gray colored nodes. The feature vectors of vertices are initialized with spectral embedding based on hypergraph Laplacian and updated by AllSetTransformer layers. Then, the feature vectors of constraints in the action candidates are updated separately by two transformer decoders to output policy and value.

In the context of MUS/MSS enumeration, we formulate the selection of constraints to delete/add in shrink/grow operations as a Markov Decision Process (MDP) and train an HGNN-based agent to improve enumeration efficiency. The agent is trained using Proximal Policy Optimization (PPO) (Schulman et al., 2017), a widely used RL algorithm, which has shown strong performance in various applications.

3 PRELIMINARIES

We first formalize the problem setting and key concepts used throughout this paper.

Consider a general constraint satisfaction problem (CSP) defined over a finite set of constraints $C = \{c_1, c_2, \ldots, c_m\}$. For any subset $S \subseteq C$, its satisfiability can be checked by an oracle solver $Oracle : 2^C \to \{satisfiable, unsatisfiable\}$.

The definitions of MUS, MSS, and MCS are as follows.

- **Minimal Unsatisfiable Subset (MUS):** A subset $M \subseteq C$ such that $Oracle(M) = unsatisfiable$ and $\forall c \in M, Oracle(M \setminus \{c\}) = satisfiable$.
- **Maximal Satisfiable Subset (MSS):** A subset $M \subseteq C$ such that $Oracle(M) = satisfiable$ and $\forall c \in C \setminus M, Oracle(M \cup \{c\}) = unsatisfiable$.
- **Minimal Correction Subset (MCS):** A subset $M \subseteq C$ such that $Oracle(C \setminus M) = satisfiable$ and $\forall c \in M, Oracle(C \setminus (M \setminus \{c\})) = unsatisfiable$. Note that MCS is the complement of MSS and is the hitting set of all MUSes.

Deletion-based MUS/MSS extraction algorithms typically consist of two operations:

- **Shrink:** Given an unsatisfiable subset $S \subseteq C$, iteratively delete constraints from S while maintaining unsatisfiability and output an MUS.
- **Grow:** Given a satisfiable subset $S \subseteq C$, iteratively add constraints to S while maintaining satisfiability and output an MSS.

In online MUS/MSS enumeration, a seed subset is picked from the unexplored search space, and then if it is unsatisfiable, the shrink operation is applied to obtain an MUS; if it is satisfiable, the grow operation is applied to obtain an MSS.

4 PROPOSED METHOD

This work aims to accelerate MUS/MSS enumeration by reducing the number of satisfiability checks through a domain-agnostic machine learning-based approach. To achieve this, we propose HyMUSE, a method that represents the MUS/MCSes enumerated until the current step as hypergraphs and employs HGNN-based agent trained via reinforcement learning to minimize the number of checks required for correcting the outputs to valid MUSes/MSSes. Overview of the proposed method is shown in Figure 1.

In the proposed method, part of the conventional shrink/grow operations is replaced by actions performed by an HGNN-based agent. At each enumeration step t, when an unexplored subset $S \subseteq C$ is picked, the agent performs the shrink operation to obtain a tentative MUS if S is unsatisfiable, or the grow operation to obtain a tentative MSS if S is satisfiable.

Algorithm 1 Correction for Shrink

Input: Output subset $S' \subseteq C$ and ordered list of constraints selected to delete D of shrink operation with HGNN agent
Output: An MUS $M \subseteq C$

if $Oracle(S') = satisfiable$ **then**
  $M \leftarrow S'$
  while $Oracle(M) = satisfiable$ **do**
    $c \leftarrow D.pop()$
    $M \leftarrow M \cup \{c\}$
  end while
  $M \leftarrow \mathrm{Shrink}(M)$
else
  $M \leftarrow \mathrm{Shrink}(S')$
end if
return M

Algorithm 2 Correction for Grow

Input: Output subset $S' \subseteq C$ and ordered list of constraints selected to add A of grow operation with HGNN agent
Output: An MSS $M \subseteq C$

if $Oracle(S') = unsatisfiable$ **then**
  $M \leftarrow S'$
  while $Oracle(M) = unsatisfiable$ **do**
    $c \leftarrow A.pop()$
    $M \leftarrow M \setminus \{c\}$
  end while
  $M \leftarrow \mathrm{Grow}(M)$
else
  $M \leftarrow \mathrm{Grow}(S')$
end if
return M

Given the MUS/MCS hypergraph $H^{(t)}$ and the subset S to be shrunk/grown, the agent iteratively selects a constraint to delete/add or decides to finish the operation and outputs the resulting subset. Since the output subset is not guaranteed to be a valid MUS/MSS, a correction algorithm is applied to convert the tentative subset into valid MUS/MSS with a small number of satisfiability checks. The HGNN-based agent is trained via reinforcement learning to minimize the number of checks required for the correction procedure.

In the following sections, we describe the details of each component of HyMUSE. First, we define the MUS/MCS hypergraph. Then, we explain the architecture of the hypergraph neural network used in the agent and how it outputs the policy for selecting constraints to delete/add. Next, we describe the correction algorithms. Finally, we explain the training method of the HGNN-based agent via reinforcement learning.

4.1 MUS/MCS Hypergraph

Our method constructs hypergraphs where vertices represent constraints and hyperedges represent obtained MUSes/MCSes until the current step. In online MUS/MSS enumeration, we can obtain an MUS or an MSS at each enumeration step. In our method, let $MUS^{(t)} = \{M_1, M_2, \ldots, M_k\}$ be the set of MUSes enumerated until step t, and $MCS^{(t)} = \{M'_1, M'_2, \ldots, M'_l\}$ be the set of MCSes obtained as complements of MSSes enumerated until step t. Considering each constraint $c \in C$ as a vertex and each MUS $M_i \in MUS^{(t)}$ and MCS $M'_j \in MCS^{(t)}$ as hyperedges, the exploration state at step t can be represented as a heterogeneous hypergraph $H^{(t)} = (\mathcal{V}, \mathcal{E}^{(t)}_{MUS}, \mathcal{E}^{(t)}_{MCS})$ where $\mathcal{V} = C$ is the set of vertices representing constraints, and $\mathcal{E}^{(t)}_{MUS} = MUS^{(t)}$ and $\mathcal{E}^{(t)}_{MCS} = MCS^{(t)}$ are the sets of hyperedges representing MUSes and MCSes, respectively. Since MUS/MCS hypergraph is available in any CSPs during MUS/MSS enumeration even when variable-constraint relationships are unknown, it is a domain-agnostic representation of CSP.

4.2 Hypergraph Neural Networks

The HGNN takes the MUS/MCS hypergraph $H^{(t)}$ and the subset $S \subseteq C$ of constraints to shrink/grow as input, and outputs the policy $\pi(a|H^{(t)}, S)$ for action a that selects a constraint to delete/add and the value $V(H^{(t)}, S)$ for training with reinforcement learning. Our model employs AllSetTransformer-style set-to-set attention (Chien et al., 2021) to embed the structure of MUS/MCS hypergraph. The architecture of our model is shown in Figure 2.

In the model, the feature vector of each vertex $v \in \mathcal{V}$ is initialized with spectral embedding derived from the normalized hypergraph Laplacian proposed by Zhou et al. (2006). Then, the vertex features are updated by AllSetTransformer layers. In each layer, the feature vectors are updated separately on MUS hyperedges and MCS hyperedges using different parameters and then combined by linear projection.

$$h_v^{(l+1)} = W^{(l)} \cdot \left(\left(f^{(l)}_{\mathcal{E}^{(t)}_{MUS} \to \mathcal{V}} \circ f^{(l)}_{\mathcal{V} \to \mathcal{E}^{(t)}_{MUS}} \right) (h_v^{(l)}) \right. \\ \left. \| \left(f^{(l)}_{\mathcal{E}^{(t)}_{MCS} \to \mathcal{V}} \circ f^{(l)}_{\mathcal{V} \to \mathcal{E}^{(t)}_{MCS}} \right) (h_v^{(l)}) \right) \quad (1)$$

Here, $h_v^{(l)} \in \mathbb{R}^d$ is the feature vector of vertex v in the l-th layer, $W^{(l)} \in \mathbb{R}^{2d \times d}$ is a learnable weight matrix, $\circ$ is the function composition operation, and $\|$ is the concatenation operation. $f^{(l)}_{\mathcal{V} \to \mathcal{E}}$ and $f^{(l)}_{\mathcal{E} \to \mathcal{V}}$ are AllSet-

Algorithm 3 Shrink with Agent

Input: An unsatisfiable set of constraints $S \subseteq C$ and MUS/MCS hypergraph $H^{(t)}$
Output: An MUS $M \subseteq S$

$S_0^{(t)} \leftarrow S$
$D \leftarrow [\,]$
for $\tau = 0, 1, 2, \ldots$ **do**
  $a_\tau^{(t)} \sim \pi(a|H^{(t)}, S_\tau^{(t)})$ from HGNN model
  if $a_\tau^{(t)}$ is $finish$ **then**
    break
  else
    $S_{\tau+1}^{(t)} \leftarrow S_\tau^{(t)} \setminus \{a_\tau^{(t)}\}$
    $D.push(a_\tau^{(t)})$
  end if
end for
$M \leftarrow \text{CorrectionForShrink}(S_\tau^{(t)}, D)$
return M

Algorithm 4 Grow with Agent

Input: A satisfiable set of constraints $S \subseteq C$ and MUS/MCS hypergraph $H^{(t)}$
Output: An MSS $M \supseteq S$

$S_0^{(t)} \leftarrow S$
$A \leftarrow [\,]$
for $\tau = 0, 1, 2, \ldots$ **do**
  $a_\tau^{(t)} \sim \pi(a|H^{(t)}, S_\tau^{(t)})$ from HGNN model
  if $a_\tau^{(t)}$ is $finish$ **then**
    break
  else
    $S_{\tau+1}^{(t)} \leftarrow S_\tau^{(t)} \cup \{a_\tau^{(t)}\}$
    $A.push(a_\tau^{(t)})$
  end if
end for
$M \leftarrow \text{CorrectionForGrow}(S_\tau^{(t)}, A)$
return M

Transformer operations defined by Chien et al. (2021), which are multi-head attention-based set-to-set functions that aggregate feature vectors of vertices $\mathcal{V}$ to hyperedges $\mathcal{E}$ and vice versa.

After AllSetTransformer layers, the candidate set is extracted and applied two separate transformer decoders (Vaswani et al., 2017) to output policy and value. When performing shrink, the candidate set is the input subset S, and when performing grow, the candidate set is the complement of the input subset $C \setminus S$. In both transformer decoders, the embeddings of vertices in the candidate set are used as queries, while all vertex embeddings are used as keys/values in cross-attention, enabling the model to consider the similarities of substructures in the MUS/MCS hypergraph.

To compute the policy $\pi(a|H^{(t)}, S)$, the output feature vectors of the candidate constraints from the policy decoder are passed through several feed-forward layers, producing a logit for each constraint in the candidate set. A virtual $finish$ action with a fixed logit of zero is appended, and softmax is applied over all logits to obtain the action probability distribution. To compute the value $V(H^{(t)}, S)$, the output feature vectors from the value decoder are aggregated via average and max pooling, concatenated, and passed through several feed-forward layers to produce a scalar estimate.

Because HGNN-based agent acts based on domain-agnostic MUS/MCS hypergraph, our method can be applied regardless of the types of variables and constraints, as long as the satisfiability can be checked by an oracle solver.

4.3 Correction Algorithm

The output subset of shrink/grow operations with the HGNN agent is not guaranteed to be a valid MUS/MSS due to prediction errors. The correction algorithm converts the tentative subsets into valid MUSes/MSSes with satisfiability check oracle. The algorithms for shrink/grow operations are shown in Algorithm 1 and Algorithm 2.

For shrink correction, if the output subset S is satisfiable, the deleted constraints are restored one by one in reverse order of their deletion until S becomes unsatisfiable again, and then the standard shrink algorithm is applied to obtain an MUS. If the output subset S is already unsatisfiable, the standard shrink algorithm is directly applied to ensure minimality.

For grow correction, if the output subset S is unsatisfiable, the added constraints are removed one by one in reverse order of their addition until S becomes satisfiable again, and then the standard grow algorithm is applied to obtain an MSS. If the output subset S is already satisfiable, the standard grow algorithm is directly applied to ensure maximality.

4.4 Training

The HGNN agent is trained via reinforcement learning to minimize the number of satisfiability checks required for the correction procedure. We formulate the iterative selection of constraints to delete/add in shrink/grow operations as a Markov Decision Process (MDP) and train the model using Proximal Policy Optimization (PPO) (Schulman et al., 2017). The shrink and grow operation with the agent are shown in Algorithm 3 and Algorithm 4.

The state $s_\tau^{(t)}$ at step τ in the shrink/grow operation at enumeration step t is defined as the pair of the MUS/MCS hypergraph $H^{(t)}$ and the subset $S_\tau^{(t)} \subseteq C$. For the shrink operation, $S_0^{(t)}$ is initialized to an unexplored unsatisfiable subset of constraints. At each step τ, the agent chooses an action $a_\tau^{(t)} \in S_\tau^{(t)} \cup \{finish\}$: either selecting a constraint $c \in S_\tau^{(t)}$ to remove from the subset, or terminating the operation by choosing $finish$. For the grow operation, $S_0^{(t)}$ is initialized to an unexplored satisfiable subset of constraints. At each step τ, the agent chooses an action $a_\tau^{(t)} \in (C \setminus S_\tau^{(t)}) \cup \{finish\}$: either selecting a constraint $c \in C \setminus S_\tau^{(t)}$ to add to the subset, or terminating the operation by choosing $finish$.

The transition is deterministic, and the next state $s_{\tau+1}^{(t)}$ is obtained by deleting/adding the selected constraint c from/to $S_\tau^{(t)}$. The episode ends when the action $finish$ is selected, and the reward $r^{(t)}$ is given at the end of the episode. The reward $r^{(t)}$ for the shrink/grow operation is calculated as follows:

$$r^{(t)} = \begin{cases} 1 - \dfrac{N_{correction} - |MUS|}{|S_0^{(t)}|} & \text{shrink} \\ 1 - \dfrac{N_{correction} - |C \setminus MSS|}{|C \setminus S_0^{(t)}|} & \text{grow} \end{cases} \tag{2}$$

Here, $N_{correction}$ is the number of satisfiability checks required for the correction procedure after the shrink/grow operation. $|MUS|$ and $|MSS|$ are the sizes of the valid MUS and MSS obtained after the correction, respectively. The minimum of $N_{correction}$ is $|MUS|$ when shrinking and $|C \setminus MSS|$ when growing, which is achieved when the output of the agent is already a valid MUS/MSS and no correction is needed. The reward is designed to encourage the agent to minimize the number of checks required for the correction.

We train the model by collecting episodes of shrink/grow operations with the current policy and updating the model parameters to maximize the expected reward via PPO.

5 EXPERIMENTS

We evaluated HyMUSE to address the following research questions:

RQ1: Does HyMUSE accelerate MUS/MSS enumeration compared to conventional methods?

RQ2: Does HyMUSE generalize to different problem distributions from those seen during training?

RQ3: How does HyMUSE perform when integrated with different enumeration algorithms?

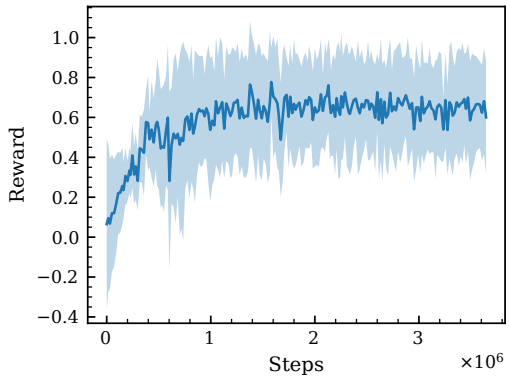

Figure 3: Reward Transition during Training. The averaged reward and the standard deviation are plotted.

5.1 Training Settings

The agent was trained on CNF instances drawn from $\mathbf{SR}(\mathbf{U}(5, 20))$, which is the distribution of random CNF instances introduced by Selsam et al. (2018). In this distribution, the number of variables is uniformly sampled from the range of 5 to 20, and then each constraint is generated by randomly sampling a subset of variables and negating each variable with a probability of 0.5. The training steps were performed in combination with MARCO (Previti and Marques-Silva, 2013; Liffiton and Malik, 2013; Liffiton et al., 2016) to select unexplored sets to shrink/grow. In total, training comprised 3.6M steps. Figure 3 shows the reward curve during training, indicating that the reward increased as training progressed and eventually converged.

5.2 Evaluation Settings

We evaluated the performance of the proposed method with the trained agent on four datasets of different problem distributions.

- $\boldsymbol{SAT_{small}}$: 500 CNF instances drawn from $\mathbf{SR}(\mathbf{U}(5, 20))$, the same distribution as in training.
- $\boldsymbol{SAT_{large}}$: 500 CNF instances drawn from $\mathbf{SR}(\mathbf{U}(20, 40))$ with larger number of variables and constraints than SAT_{small}.
- ***GC***: 500 CNF instances derived from graph coloring problems randomly generated with the method described in Lymperopoulos and Liu (2024).
- ***SMT***: 495 Satisfiability Modulo Theories (SMT) instances from the benchmark in SMT-LIB (Barrett et al., 2016; Preiner et al., 2025), referring to Bendík et al. (2018). Note that these SMT instances include problems that cannot be converted to CNF, making it impossible to represent variable-constraint relationships as graphs and apply existing GNN-based methods.

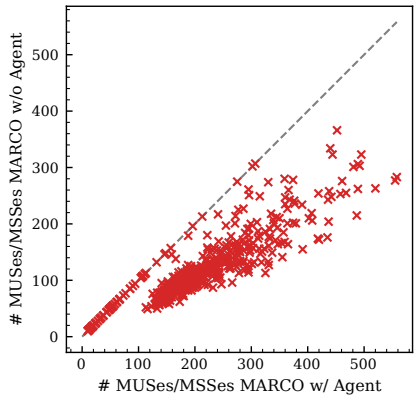

Figure 4: Comparison of MARCO with and without the agent on SAT_{small}. The x-axis shows the number of MUSes/MSSes found with the agent, and the y-axis shows that without the agent (within 10,000 checks). Points below the dashed $y = x$ line indicate improvement by the agent. Points on the line are instances where both methods found all MUSes/MSSes.

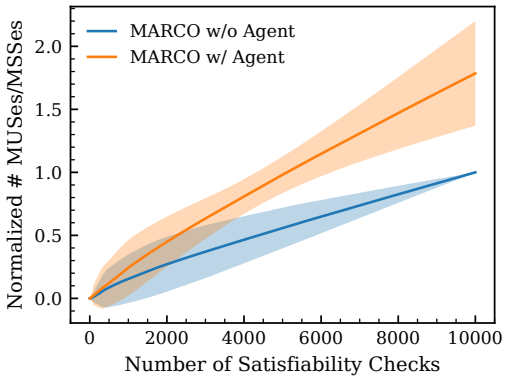

Figure 5: Relation of Number of MUSes/MSSes and Number of Satisfiability Checks. The number of MUSes/MSSes is plotted against the number of satisfiability checks for both MARCO with and without the agent.

For each dataset, we integrated the trained agent with three different enumeration strategies MARCO (Previti and Marques-Silva, 2013; Liffiton and Malik, 2013; Liffiton et al., 2016), TOME (Bendík et al., 2016) and ReMUS (Bendík et al., 2018), and compared performance with and without the agent. The main evaluation metric is the total number of MUSes and MSSes enumerated within a fixed number of satisfiability checks. This metric is appropriate for evaluating the efficiency of enumeration, as the cost of satisfiability checks dominates the overall computational cost of enumeration and is independent of the implementation and hardware. Only those instances that completed the fixed number of checks within 600 seconds and without error were included in the evaluation.

5.3 Results

The results corresponding to each research question are presented below.

5.3.1 Acceleration of MUS Enumeration

Figure 4 compares MARCO with and without the agent on SAT_{small} under a limit of 10,000 satisfiability checks. As shown in the figure, MARCO with the agent enumerated more MUSes/MSSes than MARCO without agent on all instances except for the instances where both methods completed the enumeration of all MUSes/MSSes.

Figure 5 shows the cumulative number of enumerated MUSes/MSSes as a function of the number of satisfiability checks. The numbers of MUSes/MSSes are normalized by that of MARCO without agent at 10,000 checks at each instance. Table 1 shows the improvement ratio in the number of MUSes/MSSes when integrating the trained agent with MARCO on SAT_{small}. The improvement ratio is calculated as the number of MUSes/MSSes enumerated with the agent divided by that without the agent at each instance and averaged across instances. The improvement ratio consistently exceeds 1 across different numbers of checks and quartiles.

Table 1: Improvement Ratio of MARCO with the Agent on SAT_{small}. The improvement ratio is calculated as the number of MUSes/MSSes with the agent divided by that without the agent at each instance and averaged across instances in each quartile group and overall. The quartiles are based on the number of MUSes/MSSes enumerated without the agent.

Quartile	1k checks	5k checks	10k checks
1Q	1.80 ± 0.57	2.05 ± 0.55	1.73 ± 0.58
2Q	1.78 ± 0.47	1.90 ± 0.40	1.89 ± 0.33
3Q	1.77 ± 0.36	1.82 ± 0.31	1.86 ± 0.33
4Q	1.57 ± 0.29	1.61 ± 0.31	1.67 ± 0.33
Overall	1.74 ± 0.45	1.85 ± 0.43	1.79 ± 0.42

5.3.2 Performance on Different Distributions

Table 2 shows the improvement ratio when integrating the trained agent with MARCO on datasets with different distributions from the training distribution. Integrating the agent improved the number of MUSes/MSSes enumerated on all datasets. A large variation in performance is observed on GC and SMT.

Table 2: The Number of MUSes/MSSes with and without the Agent and the Improvement Ratio on Different Datasets at 10k checks. The number of MUSes/MSSes and the improvement ratio are averaged across instances in each dataset and shown with standard deviation.

Dataset	# MUSes/MSSes		Ratio
SAT_{small}	w/o Agent	119.5 ± 60.3	1.79 ± 0.42
	w/ Agent	**211.3 ± 99.4**	
SAT_{large}	w/o Agent	54.1 ± 15.3	2.46 ± 0.42
	w/ Agent	**130.6 ± 31.6**	
GC	w/o Agent	15.1 ± 16.1	2.32 ± 0.96
	w/ Agent	**31.6 ± 28.4**	
SMT	w/o Agent	43.3 ± 100.7	1.30 ± 0.62
	w/ Agent	**58.0 ± 103.2**	

Table 3: Improvement Ratio by Integrating the Agent with Different Enumeration Method on SAT_{small}. The improvement ratio is calculated as the number of MUSes/MSSes found by each enumeration method with the agent divided by that without the agent at each instance and averaged across instances.

Method	1k checks	5k checks	10k checks
TOME	0.92 ± 0.14	1.08 ± 0.18	1.11 ± 0.16
ReMUS	1.31 ± 0.29	1.05 ± 0.37	0.75 ± 0.29

5.3.3 Performance with Different Enumeration Algorithms

Table 3 shows the improvement ratio in the number of MUSes/MSSes when integrating the trained agent with different enumeration algorithms from the algorithms used during training. As shown in the table, the improvement is marginal when integrating the agent with TOME and ReMUS, in contrast to the significant improvement observed with MARCO.

6 DISCUSSION

The results demonstrate that our method successfully accelerates MUS enumeration, producing significant improvements in the number of MUSes/MSSes enumerated within a fixed number of satisfiability checks.

The results on different distributions indicate that our method generalizes well to different problem distributions, which is a significant advantage for practical use. The results on SMT indicate that our method is domain-agnostic and can be applied to various types of CSPs, including those where explicit variable-constraint relationships are not available. The large variation in performance on different distributions from the training distribution suggests that the agent learned both domain-agnostic heuristics effective across different distributions and some heuristics specific to particular problem distributions.

The results on different enumeration algorithms suggest that the agent trained with MARCO specializes for optimizing MARCO operations and does not generalize well to other methods. This is presumably because the TOME and ReMUS have different strategies for efficient selection of unexplored sets to shrink/grow compared to MARCO and thus the agent has little margin to improve the performance. Training the agent in combination with each enumeration algorithm would yield better performance. The result of training with ReMUS is provided in Appendix E.

It would be interesting as future work to induce the acquisition of more general heuristics by training the agent on more diverse problem distributions and with different enumeration algorithms. Though MUS and MCS are used as hyperedges in this work, it is important to compare the performance among different combinations of subsets used as hyperedges, such as using only MUS, only MCS, or using other subsets of constraints including MSS and complements of MUS. It is also expected that observing the behavior of the trained agent and analyzing the HGNN using XAI methods will lead to the discovery of new rule-based heuristics.

7 CONCLUSION

In this paper, to accelerate MUS enumeration in domain-agnostic CSPs, we proposed a hypergraph neural network-based method HyMUSE. Our method constructs hypergraphs where vertices represent constraints and hyperedges represent MUSes/MCSes enumerated until the current step and employs HGNN-based agents to obtain tentative MUS/MSS candidates. The HGNN-based agents are trained via reinforcement learning to minimize the number of satisfiability checks required to correct the candidates to valid MUSes/MSSes. Experimental results demonstrate the effectiveness of our method in accelerating MUS enumeration, producing significant improvements in the number of MUSes/MSSes enumerated within a fixed number of satisfiability checks. The results also show that our method generalizes well to different problem distributions, indicating its potential for practical applications.

References

Arya, D., Gupta, D. K., Rudinac, S., and Worring, M. (2020). Hypersage: Generalizing inductive representation learning on hypergraphs. *arXiv preprint arXiv:2010.04558*.

Audemard, G. and Simon, L. (2009). Glucose: a solver that predicts learnt clauses quality. *SAT Competition*, pages 7–8.

Bai, S., Zhang, F., and Torr, P. H. (2021). Hypergraph convolution and hypergraph attention. *Pattern Recognition*, 110:107637.

Bakker, R. R., Dikker, F., Tempelman, F., and Wognum, P. M. (1993). Diagnosing and solving over-determined constraint satisfaction problems. In *IJCAI'93: Proceedings of the 13th international joint conference on Artifical intelligence*, pages 276–281. Morgan Kaufmann.

Barrett, C., Fontaine, P., and Tinelli, C. (2016). The Satisfiability Modulo Theories Library (SMT-LIB). `www.SMT-LIB.org`.

Bendík, J. (2017). Consistency checking in requirements analysis. In *Proceedings of the 26th ACM SIGSOFT international symposium on software testing and analysis*, pages 408–411.

Bendík, J., Benes, N., Cerná, I., and Barnat, J. (2016). Tunable online mus/mss enumeration. *arXiv preprint arXiv:1606.03289*.

Bendík, J. and Cerná, I. (2018). Evaluation of domain agnostic approaches for enumeration of minimal unsatisfiable subsets. In *LPAR*, pages 131–142.

Bendík, J., Černá, I., and Beneš, N. (2018). Recursive online enumeration of all minimal unsatisfiable subsets. In *International symposium on automated technology for verification and analysis*, pages 143–159. Springer.

Biere, A., Faller, T., Fazekas, K., Fleury, M., Froleyks, N., and Pollitt, F. (2024). CaDiCaL 2.0. In Gurfinkel, A. and Ganesh, V., editors, *Computer Aided Verification - 36th International Conference, CAV 2024, Montreal, QC, Canada, July 24-27, 2024, Proceedings, Part I*, volume 14681 of *Lecture Notes in Computer Science*, pages 133–152. Springer.

Bünz, B. and Lamm, M. (2017). Graph neural networks and boolean satisfiability. *arXiv preprint arXiv:1702.03592*.

Chien, E., Pan, C., Peng, J., and Milenkovic, O. (2021). You are allset: A multiset function framework for hypergraph neural networks. *arXiv preprint arXiv:2106.13264*.

Chinneck, J. W. and Dravnieks, E. W. (1991). Locating minimal infeasible constraint sets in linear programs. *ORSA Journal on Computing*, 3(2):157–168.

De Moura, L. and Bjørner, N. (2008). Z3: An efficient smt solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340. Springer.

Dong, Y., Sawin, W., and Bengio, Y. (2020). Hnhn: Hypergraph networks with hyperedge neurons. *arXiv preprint arXiv:2006.12278*.

Feng, Y., You, H., Zhang, Z., Ji, R., and Gao, Y. (2019). Hypergraph neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 3558–3565.

Fey, M. and Lenssen, J. E. (2019). Fast graph representation learning with pytorch geometric. *arXiv preprint arXiv:1903.02428*.

Gario, M. and Micheli, A. (2015). Pysmt: a solver-agnostic library for fast prototyping of smt-based algorithms. In *SMT Workshop 2015*.

Guo, W., Zhen, H.-L., Li, X., Luo, W., Yuan, M., Jin, Y., and Yan, J. (2023). Machine learning methods in solving the boolean satisfiability problem. *Machine Intelligence Research*, 20(5):640–655.

Herud, K., Baumeister, J., Sabuncu, O., and Schaub, T. (2022). Conflict handling in product configuration using answer set programming. In *ICLP Workshops*.

Huang, J. and Yang, J. (2021). Unignn: a unified framework for graph and hypergraph neural networks. *arXiv preprint arXiv:2105.00956*.

Ignatiev, A., Morgado, A., and Marques-Silva, J. (2018). PySAT: A Python toolkit for prototyping with SAT oracles. In *SAT*, pages 428–437.

Ignatiev, A., Narodytska, N., Asher, N., and Marques-Silva, J. (2020). From contrastive to abductive explanations and back again. In *International Conference of the Italian Association for Artificial Intelligence*, pages 335–355. Springer.

Ignatiev, A., Narodytska, N., and Marques-Silva, J. (2019). Abduction-based explanations for machine learning models. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 1511–1519.

Ignatiev, A., Tan, Z. L., and Karamanos, C. (2024). Towards universally accessible SAT technology. In *SAT*, pages 4:1–4:11.

Kim, S., Lee, S. Y., Gao, Y., Antelmi, A., Polato, M., and Shin, K. (2024). A survey on hypergraph neural networks: an in-depth and step-by-step guide. In *Proceedings of the 30th ACM SIGKDD Conference*

on Knowledge Discovery and Data Mining, pages 6534–6544.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kurin, V., Godil, S., Whiteson, S., and Catanzaro, B. (2020). Can q-learning with graph networks learn a generalizable branching heuristic for a sat solver? *Advances in Neural Information Processing Systems*, 33:9608–9621.

Li, C., Liu, C., Chung, J., Lu, Z., Jha, P., and Ganesh, V. (2024). A reinforcement learning based reset policy for cdcl sat solvers. *arXiv preprint arXiv:2404.03753*.

Li, Z. and Si, X. (2022). Nsnet: A general neural probabilistic framework for satisfiability problems. *Advances in Neural Information Processing Systems*, 35:25573–25585.

Liffiton, M. H. and Malik, A. (2013). Enumerating infeasibility: Finding multiple muses quickly. In *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 160–175. Springer.

Liffiton, M. H., Previti, A., Malik, A., and Marques-Silva, J. (2016). Fast, flexible mus enumeration. *Constraints*, 21(2):223–250.

Lymperopoulos, P. and Liu, L. (2024). Graph pruning for enumeration of minimal unsatisfiable subsets. In *International Conference on Artificial Intelligence and Statistics*, pages 2647–2655. PMLR.

Moriyama, S., Watanabe, K., and Inoue, K. (2023). Gnn based extraction of minimal unsatisfiable subsets. In *International Conference on Inductive Logic Programming*, pages 77–92. Springer.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. (2019). Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32.

Popescu, A., Polat-Erdeniz, S., Felfernig, A., Uta, M., Atas, M., Le, V.-M., Pilsl, K., Enzelsberger, M., and Tran, T. N. T. (2022). An overview of machine learning techniques in constraint solving. *Journal of Intelligent Information Systems*, 58(1):91–118.

Preiner, M., Schurr, H.-J., Barrett, C., Fontaine, P., Niemetz, A., and Tinelli, C. (2025). Smt-lib release 2025 (non-incremental benchmarks) (2025.08.04). Data set.

Previti, A. and Marques-Silva, J. (2013). Partial mus enumeration. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 27, pages 818–825.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Selsam, D. and Bjørner, N. (2019). Guiding high-performance sat solvers with unsat-core predictions. In *International conference on theory and applications of satisfiability testing*, pages 336–353. Springer.

Selsam, D., Lamm, M., Bünz, B., Liang, P., de Moura, L., and Dill, D. L. (2018). Learning a sat solver from single-bit supervision. *arXiv preprint arXiv:1802.03685*.

Tönshoff, J., Kisin, B., Lindner, J., and Grohe, M. (2022). One model, any csp: graph neural networks as fast global search heuristics for constraint satisfaction. *arXiv preprint arXiv:2208.10227*.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, 30.

Wang, W., Hu, Y., Tiwari, M., Khurshid, S., McMillan, K., and Miikkulainen, R. (2021). Neuroback: Improving cdcl sat solving using graph neural networks. *arXiv preprint arXiv:2110.14053*.

Yadati, N., Nimishakavi, M., Yadav, P., Nitin, V., Louis, A., and Talukdar, P. (2019). Hypergcn: A new method for training graph convolutional networks on hypergraphs. *Advances in neural information processing systems*, 32.

Yang, M. and Xu, X.-J. (2025). Recent advances in hypergraph neural networks. *arXiv preprint arXiv:2503.07959*.

Yolcu, E. and Póczos, B. (2019). Learning local search heuristics for boolean satisfiability. *Advances in Neural Information Processing Systems*, 32.

Zhai, S. and Ge, N. (2025). Learning splitting heuristics in divide-and-conquer sat solvers with reinforcement learning. In *The Thirteenth International Conference on Learning Representations*.

Zhang, R., Zou, Y., and Ma, J. (2019). Hyper-sagnn: a self-attention based graph neural network for hypergraphs. *arXiv preprint arXiv:1911.02613*.

Zhou, D., Huang, J., and Schölkopf, B. (2006). Learning with hypergraphs: Clustering, classification, and embedding. *Advances in neural information processing systems*, 19.

Checklist

1. For all models and algorithms presented, check if you include:
   (a) A clear description of the mathematical setting, assumptions, algorithm, and/or model. [Yes]
   (b) An analysis of the properties and complexity (time, space, sample size) of any algorithm. [Yes]
   (c) (Optional) Anonymized source code, with specification of all dependencies, including external libraries. [Yes] We stated that the code would be released upon acceptance of the paper, and it is now available at `https://github.com/hitachi-ais/HGNN-MUSE`.

2. For any theoretical claim, check if you include:
   (a) Statements of the full set of assumptions of all theoretical results. [Yes]
   (b) Complete proofs of all theoretical results. [Not Applicable]
   (c) Clear explanations of any assumptions. [Yes]

3. For all figures and tables that present empirical results, check if you include:
   (a) The code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL). [Yes] The code is available at `https://github.com/hitachi-ais/HGNN-MUSE`.
   (b) All the training details (e.g., data splits, hyperparameters, how they were chosen). [Yes]
   (c) A clear definition of the specific measure or statistics and error bars (e.g., with respect to the random seed after running experiments multiple times). [Yes]
   (d) A description of the computing infrastructure used. (e.g., type of GPUs, internal cluster, or cloud provider). [Yes]

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets, check if you include:
   (a) Citations of the creator If your work uses existing assets. [Yes]
   (b) The license information of the assets, if applicable. [Yes]
   (c) New assets either in the supplemental material or as a URL, if applicable. [Yes] The code is available at `https://github.com/hitachi-ais/HGNN-MUSE`.
   (d) Information about consent from data providers/curators. [Not Applicable]
   (e) Discussion of sensible content if applicable, e.g., personally identifiable information or offensive content. [Not Applicable]

5. If you used crowdsourcing or conducted research with human subjects, check if you include:
   (a) The full text of instructions given to participants and screenshots. [Not Applicable]
   (b) Descriptions of potential participant risks, with links to Institutional Review Board (IRB) approvals if applicable. [Not Applicable]
   (c) The estimated hourly wage paid to participants and the total amount spent on participant compensation. [Not Applicable]

Appendix

A Theoretical Performance Analysis

A.1 Best and Worst Case Performance

The standard shrink and grow procedure described in Liffiton et al. (2016) are shown in Algorithm 5 and Algorithm 6. The number of satisfiability checks required for the standard shrink/grow algorithms is $|S|$ and $|C \setminus S|$, respectively, where S is the input subset of constraints to be shrink/grow.

Algorithm 5 Shrink

Input: An unsatisfiable subset of constraints $S \subseteq C$
Output: An MUS $M \subseteq S$

$M \leftarrow S$
for each constraint c in S **do**
  if $Oracle(M \setminus \{c\}) = satisfiable$ **then**
    continue
  else
    $M \leftarrow M \setminus \{c\}$
  end if
end for
return M

Algorithm 6 Grow

Input: A satisfiable subset of constraints $S \subseteq C$
Output: An MSS $M \supseteq S$

$M \leftarrow S$
for each constraint c in $C \setminus S$ **do**
  if $Oracle(M \cup \{c\}) = unsatisfiable$ **then**
    continue
  else
    $M \leftarrow M \cup \{c\}$
  end if
end for
return M

In our method, the minimum number of satisfiability checks required to obtain an MUS/MSS from an unsatisfiable/satisfiable set of constraints S is $|MUS|$ and $|C \setminus MSS|$, respectively, which is achieved when the output of the agent is already a valid MUS/MSS and no correction is needed.

In the worst case, the number of satisfiability checks required for the correction procedure is $2|S|$ when shrinking, which occurs when the agent deletes all constraints in S and the correction procedure needs to add back all constraints. When growing, the worst case is $2|C \setminus S|$, which occurs when the agent adds all constraints in $C \setminus S$ and the correction procedure needs to remove all added constraints.

A.2 Computational Cost

To assess the practical impact of the improvement, the comparison of the computational cost or time required to enumerate a certain number of MUSes/MSSes with and without the agent is also important. The effectiveness of the improvement in terms of computational cost depends on the balance between the cost of satisfiability checks and the cost of the agent's inference. Let C_{check} be the average cost of a satisfiability check, C_{infer} be the average cost of the agent's inference, N_{check} be the number of satisfiability checks required for extraction of an MUS/MSS without the agent, N'_{check} be the number of satisfiability checks required for extraction of an MUS/MSS with the agent, and N_{infer} be the number of times the agent is invoked during extraction. Our method is effective in terms of computational cost when the following inequality holds:

$$C_{check} \cdot N'_{check} + C_{infer} \cdot N_{infer} < C_{check} \cdot N_{check} \quad (3)$$

The left-hand side of the inequality represents the total cost of extraction with the agent, which consists of the cost of satisfiability checks required for correction and the cost of agent inference. The right-hand side represents the cost of extraction without the agent, which consists of the cost of satisfiability checks required for the standard shrink/grow procedure. This can be rewritten as:

$$\frac{C_{infer}}{C_{check}} < \frac{N_{check} - N'_{check}}{N_{infer}} \quad (4)$$

As shown in the equation, our method is effective when the ratio of the inference cost to the check cost is less than $r_{eff} = (N_{check} - N'_{check})/N_{infer}$, which represents the average number of checks saved per agent invocation. Table 4 shows the average number of checks required to extract an MUS/MSS with and without the agent, the average number of inference calls, and the effective ratio r_{eff} on each dataset.

Table 4: Average Number of Checks and Inference Calls for Extracting an MUS/MSS with and without the Agent and the Effective Ratio r_{eff} on Different Datasets. The numbers are averaged across instances in each dataset and shown with standard deviation.

Dataset	N_{check}	N'_{check}	N_{infer}	r_{eff}
SAT_{small}	85 ± 37	44 ± 17	48 ± 24	0.83 ± 0.11
SAT_{large}	196 ± 51	81 ± 22	127 ± 37	0.90 ± 0.08
GC	996 ± 540	478 ± 312	673 ± 422	0.77 ± 0.27
SMT	136 ± 313	117 ± 428	94 ± 243	0.25 ± 0.71

B Details of Experimental Settings

B.1 Training Settings

The agent was trained on randomly generated CNF instances drawn from $\mathbf{SR}(\mathbf{U}(5, 20))$ distribution, which was introduced by Selsam et al. (2018). The geometric distribution's probability parameter was set to 0.3 as used in Lymperopoulos and Liu (2024). At each parameter update, approximately 18k steps were collected through shrink/grow episodes across four different CNF instances using the current policy within 5000 satisfiability checks budget, and the model parameters were updated for four epochs. In total, training comprised 3.6M steps. The training steps were performed in combination with MARCO (Previti and Marques-Silva, 2013; Liffiton and Malik, 2013; Liffiton et al., 2016) for selecting unexplored sets to shrink/grow. The HGNN architecture used three AllSetTransformer layers and three transformer decoder layers, with feature dimension 64 and four attention heads. Optimization was performed using Adam (Kingma and Ba, 2014) with a learning rate of 2×10^{-5} and a batch size of 1024 via gradient accumulation.

B.2 Implementation Details

For implementation of HGNN model, we used PyTorch (Paszke et al., 2019) and PyTorch Geometric (Fey and Lenssen, 2019). For the oracle solver for CNF instances and the solver for mapping clauses in enumeration algorithms, we used PySAT (Ignatiev et al., 2018, 2024) with CaDiCaL (Biere et al., 2024) and Glucose (Audemard and Simon, 2009) backend. For the oracle solver for SMT instances, we used PySMT (Gario and Micheli, 2015) with Z3 (De Moura and Bjørner, 2008) backend. All experiments were conducted on a machine with an Intel Xeon Gold 6130 CPU with 256GB RAM and four NVIDIA Tesla V100 GPUs with 32GB memory.

C Effect of Training

To evaluate the effect of training, we compared the performance of the agent trained with MARCO and that of a randomly initialized agent without training when integrated with MARCO on SAT_{small}. Table 5 shows the improvement ratio in the number of MUSes/MSSes. The trained agent enumerated significantly more MUSes/MSSes than the untrained agent, demonstrating the effectiveness of training.

Table 5: The Improvement Ratio in the Number of MUSes/MSSes when Integrating the Trained Agent and the Untrained Agent with MARCO on SAT_{small}. The numbers are averaged across instances and shown with standard deviation.

Agent	1k checks	5k checks	10k checks
Trained	1.74 ± 0.45	1.85 ± 0.43	1.79 ± 0.42
Untrained	1.02 ± 0.22	0.93 ± 0.15	0.88 ± 0.11

D Direct Comparison with Other Enumeration Methods

To directly compare the performance of different enumeration methods, we compared the number of MUSes/MSSes enumerated within a fixed number of satisfiability checks with and without the agent trained with MARCO. Table 6 shows the normalized number of MUSes/MSSes with different enumeration methods, where the numbers are normalized by that of MARCO without agent at each instance and averaged across instances. While ReMUS without the agent performs the best among the all combinations of enumeration methods and agent integration, MARCO with the agent outperforms TOME. It is a significant result considering that TOME has an efficient strategy for selecting unexplored sets to shrink/grow.

Table 6: Normalized Number of MUSes/MSSes Enumerated with Different Enumeration Methods on SAT_{small}. The numbers are normalized by that of MARCO without agent at each instance and averaged across instances.

Method	MARCO	TOME	ReMUS
w/o Agent	(1.00)	1.52 ± 0.48	$\mathbf{3.29 \pm 1.54}$
w/ Agent	1.79 ± 0.42	1.69 ± 0.57	2.16 ± 0.58

E Training with Different Enumeration Algorithms

We also trained the HGNN-based agent with ReMUS. ReMUS recursively searches for smaller unexplored subsets to shrink minimizing the number of satisfiability checks required to obtain an MUS and achieves state-of-the-art efficiency on some CSP domains among domain-agnostic MUS enumeration algorithms (Bendík and Cerná, 2018). Table 7 and Table 8 shows the number of MUSes/MSSes enumerated within 10,000 satisfiability checks of each method with and without the agent trained with ReMUS on

SAT_{small} and SAT_{large}, respectively. The numbers are normalized by that of MARCO without agent at each instance. The results show that proposed method improves the performance of ReMUS on SAT_{large}, thus achieving state-of-the-art performance on this dataset. However, the performance deteriorates on SAT_{small}. It is assumed that ReMUS already employs a sophisticated heuristic to select small unexplored subsets to shrink, and therefore the agent has limited room for improvement when the scale of the problem is small.

Table 7: Normalized Number of MUSes/MSSes Enumerated within 10,000 Satisfiability Checks on SAT_{small} with and without the agent trained with ReMUS.

Method	MARCO	TOME	ReMUS
w/o Agent	(1.00)	1.51 ± 0.48	**3.26 ± 1.48**
w/ Agent	1.85 ± 0.44	1.67 ± 0.56	2.19 ± 0.59

Table 8: Normalized Number of MUSes/MSSes Enumerated within 10,000 Satisfiability Checks on SAT_{large} with and without the agent trained with ReMUS.

Method	MARCO	TOME	ReMUS
w/o Agent	(1.00)	2.17 ± 0.40	2.00 ± 0.73
w/ Agent	2.50 ± 0.43	2.47 ± 0.47	**3.01 ± 0.41**

F Performance Comparison on Different Distributions

The performance comparison of MARCO with and without the agent on SAT_{large}, GC and SMT is shown in Figure 6, Figure 7 and Figure 8, respectively. On SAT_{large}, MARCO with the agent enumerated more MUSes/MSSes than MARCO without agent on almost all instances except for one instance where both methods enumerated all MUSes/MSSes. Even on GC and SMT, where the performance improvement is relatively small compared to SAT_{small}, MARCO with the agent outperformed MARCO without the agent on most instances.

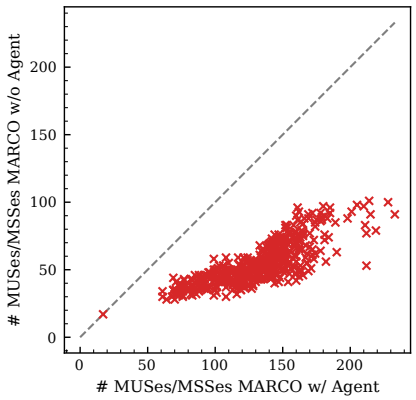

Figure 6: Comparison of MARCO with and without the Agent on SAT_{large}.

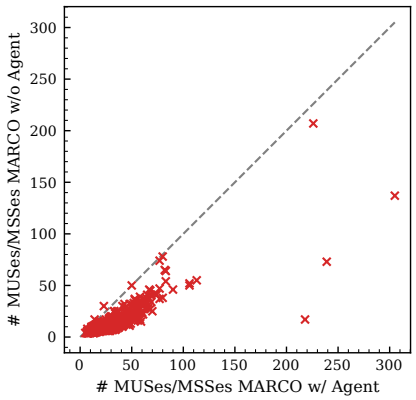

Figure 7: Comparison of MARCO with and without the Agent on GC.

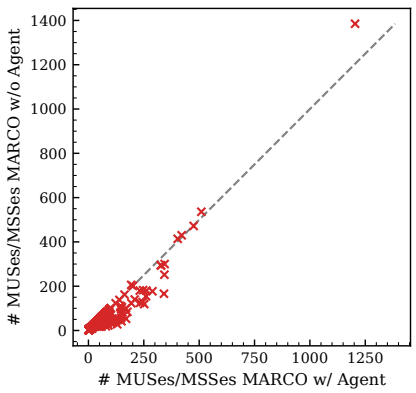

Figure 8: Comparison of MARCO with and without the Agent on SMT.