Graph Pruning for Enumeration of Minimal Unsatisfiable Subsets

Panagiotis Lymperopoulos
Tufts University

Liping Liu
Tufts University

Abstract

Finding Minimal Unsatisfiable Subsets (MUSes) of boolean constraints is a common problem in infeasibility analysis of over-constrained systems. However, because of the exponential search space of the problem, enumerating MUSes is extremely time-consuming in real applications. In this work, we propose to prune formulas using a learned model to speed up MUS enumeration. We represent formulas as graphs and then develop a graph-based learning model to predict which part of the formula should be pruned. Importantly, the training of our model does not require labeled data. It does not even require training data from the target application because it extrapolates to data with different distributions. In our experiments we combine our model with existing MUS enumerators and validate its effectiveness in multiple benchmarks including a set of real-world problems outside our training distribution. The experiment results show that our method significantly accelerates MUS enumeration on average on these benchmark problems.

1 INTRODUCTION

Many problems in computer science and operations research are often formulated as constraint satisfaction problems. When a system is over-constrained and has no satisfying solutions, identifying and enumerating Minimal Unsatisfiable Subsets (MUSes) of the constraint set is one way to analyze the system and understand the unsatisfiability. Example applications include circuit error diagnosis (Han and Lee, 1999), symbolic bounded model checking (Clarke et al., 2000; Ghassabani et al., 2017) and formal equivalence checking (Cohen et al., 2010). Additionally, MUSes may also be used to identify inconsistencies between the environment and domain knowledge (Goel et al., 2022) in planning tasks. A wide range of constraint satisfaction problems are represented as boolean formulas, which often take the Conjunctive Normal Form (CNF). Then, a MUS of a formula is an unsatisfiable subset of the clauses in the formula, with the added property that removing any single clause from this subset renders it satisfiable. MUS enumeration aims to find all MUSes in a formula.

However, MUS enumeration poses itself as a problem harder than a satisfiability decision problem. In practice, the enumeration problem is mainly addressed by search-based algorithms, which face a large search space containing exponentially many combinations of clauses (Ignatiev et al., 2015). Given the large space, a search often requires many steps involving calls to satisfiability solvers (Liffiton et al., 2016). A promising direction of speeding up the enumeration is to reduce the search space (e.g. exploring subsets that are more likely to be unsatisfiable (Bendík et al., 2018; Belov et al., 2013)). However, manual design of search heuristics faces various difficulties.

Recently, neural methods have been proposed to address hard graph problems (Khalil et al., 2017; Li et al., 2018; Sato et al., 2019) such as the traveling salesperson problem (Shi and Zhang, 2022) and the maximum independent set problem (Schuetz et al., 2022). Often these problems are addressed using Graph Neural Networks (GNNs) (Zhou et al., 2020). At the same time, there has been increasing interests in applying neural networks to problems involving logic and reasoning (Hitzler et al., 2022). Previous work on neural SAT-solving (Selsam et al., 2018) shows promising results in extrapolation beyond the training distribution. We find that this parallel development presents an opportunity for MUS enumeration: we can represent a formula as a graph and then leverage the power of GNNs to accelerate MUS enumeration. As learning models, GNNs have the ability to identify patterns

Proceedings of the 27$^{\text{th}}$ International Conference on Artificial Intelligence and Statistics (AISTATS) 2024, Valencia, Spain. PMLR: Volume 238.

that are relevant to satisfiability and possibly MUSes.

In this work, we propose a learning-based pruning model, Graph Pruning for Enumeration of Minimal Unsatisfiable Subsets (GRAPE-MUST), to accelerate MUS enumeration. In particular, GRAPE-MUST represents a formula as a graph and then learns a GNN to prune the formula via its graph form. With this generic pruning procedure, it can be combined with most existing MUS enumeration algorithms and reduce their search spaces.

The training objective of GRAPE-MUST aims to reduce clauses in a formula while keeping it unsatisfiable. This design avoids the need of true labels in model training. We further train a GRAPE-MUST model with a large number of random formulas from our specially designed generative procedure. The trained model improves enumeration performance even in tasks without training formulas. During testing time, the pruning procedure only takes a small fraction of the overall running time that includes the enumeration time, but it speeds up the enumeration procedure significantly for a wide range of problems.

We validate the effectiveness of our approach by combining our method with existing MUS enumerators in multiple benchmarks including random formulas, graph coloring problems and problems from a logistics planning domain. We also find that GRAPE-MUST shows promising extrapolation performance on larger problems than ones it was trained on. This suggests that training GRAPE-MUST on smaller problems can be a viable strategy for accelerating MUS enumeration in larger problems. Finally, we demonstrate that GRAPE-MUST can also generalize across data distributions by improving enumeration performance in a collection of hard problems from the 2011 SAT competition MUS Track using a model trained on random formulas. This result has a strong implication: a trained GRAPE-MUST has the potential to accelerate MUS enumeration for a wide range of tasks without the need for training formulas or retraining for different tasks.

2 RELATED WORK

Finding MUSes is a well-studied problem in the field of constraint satisfaction. Even though the problem is fundamentally computationally hard (Ignatiev et al., 2015), its practical usefulness has motivated the development of a number of domain agnostic algorithms. These algorithms are concerned with either extracting a single MUS from a given constraint set (Belov and Marques-Silva, 2012; Nadel et al., 2014) or with enumerating multiple MUSes, with algorithms in the latter usually building on top of the former (Bendík and Černá, 2020). Since full enumeration of MUSes is often intractable, algorithms for *online* enumeration (Liffiton et al., 2016; Bendík et al., 2018, 2016) have been developed that promise to produce at least some MUSes in reasonable time. Our work aims to improve enumeration speed of these online algorithms in problems with boolean constraints and variables.

Recently neural methods have been applied in logical reasoning (Hitzler et al., 2022). Such examples include neural satisfiability solvers (Guo et al., 2022; Li and Si, 2022), neural theorem provers (Paliwal et al., 2020), and neural model counters (Abboud et al., 2020). These methods often represent problems in graphs and apply GNNs (Welling and Kipf, 2016) to identify structural patterns in these problems.

GNNs have been shown to be successful in addressing hard graph problems (Ma et al., 2021). Examples include the traveling salesperson problem (Shi and Zhang, 2022), maximum cut and independent set (Schuetz et al., 2022; Toenshoff et al., 2021) and graph edit distance (Liu et al., 2022). In these problems, GNNs can learn common graph patterns from a large number of problems and assist a solver in finding good solutions. They achieve this by providing search heuristics (Shi and Zhang, 2022) or reducing the problem search space (Liu et al., 2022).

3 BACKGROUND

Consider a constraint satisfaction problem in CNF over a set of boolean variables $U = \{u_i \in \{T, F\}, i = 1 \dots N\}$ with clause set $C = \{c_i : i = 1 \dots M\}$. Each clause $c \in C$ is a disjunction $c = \bigvee_{i=1}^{n} l_i$ where each literal l_i is either a variable u_i or the negation of a variable $\neg u_i$. The full formula $S = \bigwedge_{i=1}^{M} c_i$ is the conjunction of all clauses.

A boolean CNF formula is satisfiable if there exists an assignment to the variables in U such that the formula S evaluates to true. A boolean CNF formula is unsatisfiable if there is no such assignment to the variables.

Definition 1 *A Minimal Unsatisfiable Subset (MUS) of a set of clauses C is a clause subset $M \subseteq C$ s.t. M is unsatisfiable, and $M \backslash \{c\}$ is satisfiable $\forall c \in M$.*

A single MUS is often viewed as one minimal explanation of why C is unsatisfiable. It is often desirable to enumerate multiple MUSes to locate good explanations. In this work we focus on accelerating the enumeration of MUSes, aiming specifically to aid practical applications in which full enumeration is computationally infeasible.

Proposition 1 *Given an unsatisfiable subset $C' \subseteq C$, if M is a MUS of C' then M is a MUS of C.*

This proposition is well known (e.g. (Bendík and Černá, 2020)), and here we formalize it to facilitate discussion. In fact, it is commonly used in MUS enumeration algorithms that use a *seed-shrink* procedure (Bendík and Černá, 2020; Liffiton et al., 2016; Bendík et al., 2016, 2018). This procedure looks for an initial *seed* $C' \subseteq C$ that is unsatisfiable, and then repeatedly shrinks C' by removing clauses until it becomes a MUS.

Definition 2 *A clause $c \in C'$ is critical for an unsatisfiable subset $C' \subseteq C$ if $C' \backslash \{c\}$ is satisfiable.*

Critical clauses are important in MUS enumeration as they tend to be involved in many MUSes of C and some solvers use them in their search strategy (Bendík et al., 2018).

4 METHOD

In this section we develop a neural method to accelerate searching algorithms in the enumeration of MUSes. In particular, we will prune the clause set C to get a smaller clause set C' that is still unsatisfiable. From that point, searching algorithms can be applied to enumerate MUSes in C' according to Proposition 1. Our main strategy is to represent formulas as graphs and then treat the pruning problem as a node labeling problem.

4.1 CNF Formulas As Attributed Graphs

We represent CNF formulas as Literal Clause Graphs (Guo et al., 2022). Consider a propositional formula $S = (U, C)$ in CNF with variable set U and clause set C. To construct an attributed graph $G = (V, E, X)$ from a formula, we treat variables and their negations as one node type and clauses as another node type. We also have two types of edges: the first type connects variables to their negations, and the second type connects variables or their negations to a clause if they appear in that clause. Formally, we construct the graph $G = (V, E, X)$ from the formula $S = (U, C)$ with the node set:

$$\begin{aligned} &V = V_1 \cup V_2 \text{ with} \\ &V_1 = \{u_i | i = 1...N\} \cup \{\neg u_i | i = 1...N\}, V_2 = C \end{aligned} \tag{1}$$

and the edge set:

$$\begin{aligned} &E = E_1 \cup E_2 \text{ with} \\ &E_1 = \{(u_k, c_j) | u_k \in c_j, j = 1...M\}, \\ &E_2 = \{(u_i, \bar{u}_i) | i = 1...N\}. \end{aligned} \tag{2}$$

Each node and each edge are associated with one-hot vectors indicating their types. All node and edge types are recorded respectively in two matrices $X = (X_V \in \{0,1\}^{|V|\times 2}, X_E \in \{0,1\}^{|E|\times 2})$ that encode the node and edge types in a one-hot manner. The graph representation $G = (V, E, X)$ keeps all information of the formula since one can recover the formula from the attributed graph. We denote the procedure by $G = \mathrm{g}_{\mathrm{rep}}(S)$ for easy reference later.

4.2 CNF Pruning Via Graph Pruning

Using our graph formulation we can achieve formula pruning through graph pruning, that is, pruning nodes in V_2 corresponds to removing clauses from C.

We formulate the pruning problem as a node labeling problem. We learn a model $p_\theta(\mathbf{y}|G)$ parameterized by θ, and the model predicts a vector $\mathbf{y} \in \{0,1\}^M$ of labels for nodes only in V_2, then $\mathbf{y}$ indicates *which nodes to keep* after pruning. Formally, $\mathbf{y}$ decides the pruned subset $C' \subseteq C$.

$$C' = \{c_i | c_i \in C, y_i = 1\} \tag{3}$$

From the pruned clause set C', we obtain the pruned formula $S' = (U', C')$. Here U' only contains variables involved in C'.

To get a differentiable training objective later, we treat the model $p_\theta(\mathbf{y}|G)$ as a distribution of $\mathbf{y}$. Here we use a simple model that treats y_i-s as independent Bernoulli random variables. The probabilities of $\mathbf{y}$ are computed from the input G by a neural network, and θ denotes its learnable parameters. More complex $\mathbf{y}$ may better capture patterns in the graph but usually require more complex parameterizations and more computations.

With the pruning procedure above, the model $p_\theta(\mathbf{y}|G)$ essentially defines a distribution $p_\theta(S'|S)$.

4.3 Optimization

We now need to form a training objective and learn parameters of the pruning model $p_\theta(S'|S)$. Since data labeling requires expensive searching procedures, we use a weak supervision scheme that does not need labeled data. A pruned formula S' should be small and unsatisfiable. We first design a loss function guided by this principle.

Loss function Given a formula $S = (U, C)$, the loss function $\mathrm{loss}(S'; S)$ computes a loss value for the pruned formula $S' = (U', C')$ by:

$$\mathrm{loss}(S'; S) = \begin{cases} 1 & \text{if SAT}(S') \\ \left(\frac{|C'|}{|C|}\right)^2 & \text{otherwise} \end{cases}. \tag{4}$$

Here $\mathrm{SAT}(S')$ corresponds to a query to a satisfiability solver on S' that returns true if a satisfying assignment is found and false otherwise.

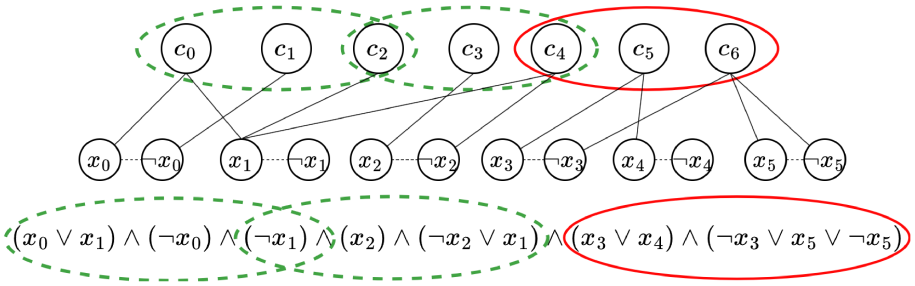


Figure 1: Graph pruning for MUS enumeration. CNF formulas are represented as Literal Clause Graphs and the clause nodes are pruned. The resulting graph is converted back into a formula and MUSes are enumerated. Green denotes two MUSes in the formula. Red denotes clauses that can be pruned without affecting the MUSes. c_2 is a critical constraint

The loss function equally weighs two types of undesirable pruned formulas: if S' is satisfiable, then the prediction is not usable and receives a penalty of 1; and if there is no pruning and $S' = S$, again it receives a penalty of 1. Otherwise, the penalty is a function of the ratio of the number of clauses in the pruned formula to the original formula. This encourages the model to prune as many clauses as possible, while maintaining unsatisfiability. This loss function will consider critical clauses (Definition 2) automatically: removing critical clauses of C produces satisfiable formulas and thus incurs high penalties. As a result, it encourages the learning model to shrink the formulas while keeping such critical clauses intact. While this loss does not penalize destroying MUSes, it avoids searching for MUSes during training and thus enables better scalability. We further discuss this issue in later sections.

Learning Objective The loss function $\text{loss}(\text{S}'; \text{S})$ is not differentiable with respect to S', and there is not a straightforward continuous relaxation of it. To get a differentiable learning objective, we take the expectation of the loss using the distribution $p_\theta(S'|S)$.

$$L(\theta; S) = \mathbb{E}_{S' \sim p_\theta(S'|S)}[\text{loss}(S'; S)]. \tag{5}$$

Then we can draw a few Monte Carlo samples of S' and estimate the gradient with respect to θ through the score function estimator (Williams, 1992).

$$\nabla_\theta L(\theta; S) = \mathbb{E}_{S' \sim p_\theta(S'|S)}[\text{loss}(S'; S) \cdot \nabla_\theta \log p_\theta(S'|S)].$$

Though the score function estimator often has large variance, it works well in our experiments. We will explore techniques to reduce the variance in the future.

4.4 Model Architecture

We now present our implementation of the graph pruning model $p_\theta(\mathbf{y}|G)$ with GNNs. To facilitate the scalability of our method, we use a lightweight architecture with a relatively small memory footprint. Along with node features X_V indicating node types, we also append random node features to improve the expressiveness of the network (Abboud et al., 2021; Sato et al., 2021). So the input to the GNN is:

$$H^0 = [X_V, R], \quad R \in \mathbb{R}^{N \times d_r} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}). \tag{6}$$

Then we use an L-layer GNN with heterogeneous message passing layers (Wang et al., 2022) to compute node representations $Z \in \mathbb{R}^{N \times d_o}$ for each node type, where d_o is the output dimension of the GNN.

$$Z = \text{gnn}(G, H_0). \tag{7}$$

Here $\text{gnn}(\cdot)$ denotes the function of the GNN. Finally, we use a simple MLP to predict probabilities of node labels $\mathbf{y}$, which indicate which clauses in C to *keep* in the pruned formula.

$$\boldsymbol{\mu} = \text{mlp}(Z), \quad \mathbf{y} \sim \text{Bernoulli}(\boldsymbol{\mu}). \tag{8}$$

Here the MLP applies to each node representations to compute a probability value.

We take a "conservative-to-aggressive" strategy to train the model. We initialize the bias in the last layer of the MLP to a moderate negative value (e.g. -3) and network weights to small values. This results in the model initially assigning a pruning probability around 0.05 to all nodes, which means the initial model does little pruning to all formulas. Then as the model learns to minimize the loss, it becomes more aggressive and prunes formulas to get smaller unsatisfiable formulas. This strategy avoids initial models that are too aggressive and cannot get unsatisfiable formulas as such models cannot improve through small updates and are hard to optimize.

4.5 Randomized Formula Generation From Problem Statistics

For tasks without training formulas, we generate random formulas as the training data to train our model.

However, it is non-trivial to generate formulas that are like real problems. Pure random formulas that are unsatisfiable tend to have small MUSes.

In this work, we devise a randomized procedure that generates formulas with similar clause lengths and clause-to-variable ratio with that from target tasks. We also consciously try to control sizes of MUSes in these formulas. According to the target clause-to-variable ratio, we first decide the number of variables and a lower bound of the number of clauses. The generation procedure then proceeds by sampling one clause at a time and adding it to the formula only if the resulting clause-set is satisfiable.

Algorithm 1 Prune a CNF formula

Input $S = (U, C)$, $\boldsymbol{\mu}$, k
Output $S' = (U', C')$

$t_{max} \leftarrow \max(\boldsymbol{\mu})$
$t_{min} \leftarrow \min(\boldsymbol{\mu})$
$T \leftarrow \{t_{min} + i \times \frac{t_{max} - t_{min}}{k} | i = 0 \dots k\}$
$l = 0$, $r = k$
while $l < r$ **do**
    $t \leftarrow T[\lfloor \frac{r-l}{2} \rfloor]$
    $\mathbf{y} \leftarrow \boldsymbol{\mu} \leq t$
    $S' \leftarrow Prune(S, \mathbf{y})$ ▷ using eq. 3
    if $SAT(S')$ **then**
        $l \leftarrow \lfloor \frac{r-l}{2} \rfloor$
    else
        $r \leftarrow \lfloor \frac{r-l}{2} \rfloor$
    end if
end while
return S'

This procedure is repeated until the lower bound is reached. After that point, we continue to add clauses until the formula becomes unsatisfiable. The literals in each clause are uniformly sampled, and the length of the clause is randomly decided according to the target clause length distribution.

This procedure yields problems that resemble the data distribution, specifically in clause lengths and clause-to-variable ratios. Without satisfiability checking, random clauses tend to make MUSes smaller. Our procedure guarantees satisfiability initially and thus tends to generate formulas with larger MUSes than pure random formulas of the same lengths. The procedure does require a large number of calls to a SAT solver, but many problems can be generated in parallel and we only need to generate a training set once for a wide range of problems.

4.6 Test-time Pruning

In testing time, we use a deterministic procedure to compute a valid pruning vector $\mathbf{y}$ to avoid randomness. We apply a threshold t to truncate the probability vector $\boldsymbol{\mu}$ to get $\mathbf{y}$. Then we check whether the pruned formula S' from $\mathbf{y}$ is satisfiable or not. We then proceed to search for the smallest threshold value (most aggressive pruning) that yields an unsatisfiable formula using binary search. The search is conducted over threshold values from $\min(\boldsymbol{\mu})$ to $\max(\boldsymbol{\mu})$ over k equally sized steps, with k being a hyperparameter not related to the size of S. This procedure is formally described by Algorithm 1. In the worst case, $t = \max(\boldsymbol{\mu})$ will give $S' = S$ without pruning. There are $O(\log k)$ SAT calls, which is typically much less than SAT calls in MUS searching algorithms.

After we have obtained the pruned formula S', we run a MUS enumeration algorithm on S' to enumerate MUSes of S. The main gain is that time saved by running the enumeration algorithm on a smaller formula S'. While some MUSes may be destroyed during pruning, we deem this a reasonable compromise as in practical problems enumerating all MUSes is already prohibitively expensive. Our experiments show that our pruning allows for more MUSes to be found within the same time-limit in both synthetic and real-life problems.

5 EXPERIMENTS

In this section, we evaluate the effectiveness of the proposed pruning strategy in MUS enumeration tasks. We first check whether applying a pruning model before an enumeration algorithm improves the algorithm's performance in enumerating MUSes. We also investigate the generalization ability of our model by evaluating a trained pruning model on formulas from a distribution different from the training distribution. Finally, we evaluate the feasibility of using a model trained on random formulas on a benchmark of challenging MUS enumeration problems from the literature, thus reducing the need to train a model on each new problem distribution.

Datasets We evaluate GRAPE-MUST on four datasets including both randomly generated and real-world problems. We use the following datasets:

Random Formulas. We generate formulas using the procedure described in (Selsam et al., 2018), resulting in formulas with about 700 clauses. Exact generation parameters are available in the appendix.

Logistics Planning. We use a standard logistics planning problem with variable numbers of cities, addresses, airplanes, airports and trucks. We create random initial and goal states and also vary the number of deliveries in a given timeframe. We only keep infeasible problems and then use MADAGASCAR (Rin-

tanen, 2014) to translate our planning problems into boolean CNF formulas. The derived formulas have about 800-1200 variables and 8000-15000 clauses. Exact generation parameters, domain file and conversion parameters are available in the appendix.

Graph Coloring. To generate random graph coloring problems, we first sample a random graph with 10 to 30 nodes from the Erdős-Rényi model with edge probability 0.8, and then randomly choose a color number between 4 and 7. Then, we convert formulas to SAT using a standard translation procedure described in the appendix. We only sample unsatisfiable formulas by discarding any satisfiable ones. The resulting formulas have up to 210 variables and up to about 2500 clauses.

Hard problems from SAT Competition 2011 MUS track.[1]. This benchmark contains problems from various applications such as planning, software and hardware verification that vary in size from a few hundred to millions of clauses and variables. We limit our investigation to problems in the benchmark that we deem hard: They contain at least 10^5 clauses and a state-of-the-art enumerator identifies at most 10^3 MUSes within a 2 hour time limit without exhausting the number of MUSes in the formulas. Given these criteria, we evaluate GRAPE-MUST on 63 problems from this benchmark.

For each of the first three datasets, we test enumeration algorithms on 500 randomly generated problems and repeat each experiment 5 times. As a separate note, these problems do not pose significant difficulties to modern SAT solvers such as Glucose-3.0 (Audemard and Simon, 2009), which can evaluate their satisfiability within 10 milliseconds.

Enumeration Algorithms We apply the pruning strategy to three contemporary online MUS enumeration algorithms: *MARCO* (Liffiton et al., 2016), *TOME* (Bendík et al., 2016), and *REMUS* (Bendík et al., 2018). All three algorithms are available as part of the MUST (Bendík and Černá, 2020) toolbox.

Model Hyperparameters We use the GraphConv operator (Morris et al., 2019) to implement message passing in heterogeneous graphs formed by formulas. In all experiments we use five message passing layers with 64 hidden units. We use two layers for the MLP with ReLU activations. We train the models for a maximum of 2 million formulas with early stopping and use the Adam optimizer with a learning rate of 0.0001 and a batch size of 32. At test time, we set $k = 10$ in algorithm 1 to compute the pruning. All our experiments are carried out on a server with 4 NVIDIA RTX 2080Ti GPUs, and an Intel(R) Core(TM) i9-9940X processor with 130 GB of memory.

[1] http://www.cril.univ-artois.fr/SAT11/

5.1 Improving Enumeration Performance

In this section we present the results of our first experiment. We run three existing MUS enumeration algorithms and compare performances of each algorithm with and without our pruning step. We run all three algorithms with the same default parameters in all problems. We evaluate the algorithms on the three aforementioned datasets. The evaluation metric is the average number of MUSes enumerated within a fixed time budget: the larger, the better. The running time of GRAPE-MUST on GPUs is included within the time-budget. In our experiments, we record three average numbers for three time budgets: 1, 2, and 5 seconds. We also run smaller scale experiments for a 30 minute time-budget and present the results as well as pruning statistics in the appendix. We note that these classes of problems are not very challenging to MUS enumerators at the problem size and time limit used in this experiment. Still, we believe that the results are indicative of the effectiveness of pruning in accelerating MUS enumeration. Performance improvement in more challenging problems is shown in following sections.

The experiment results from the three datasets are tabulated in Table 1, 2, and 3. These results show that GRAPE-MUST allows MARCO and REMUS to find more MUSes on all three datasets. On the *Graph Coloring* dataset, REMUS with pruning nearly doubles the number of MUSes found by REMUS alone across all timeouts. It also helps TOME to find more MUSes on two datasets, *Random Formulas* and *Logistics Planning*. Only on the *Graph Coloring* dataset, pruning actually harms the performance of TOME. Importantly REMUS is the strongest algorithm among the three (Bendík and Cerná, 2018), and our pruning model improves REMUS on all three datasets, making GRAPE-MUST+REMUS the strongest configuration.

Further analysis shows how the three algorithms benefit differently from a prior pruning step. REMUS invests significant computation in finding small unsatisfiable subsets and heavily depends on critical constraints. GRAPE-MUST learns to prune non-critical constraints, which makes finding and using critical constraints easier for REMUS. Compared to REMUS, MARCO benefits less from pruning in all three tasks. MARCO's search strategy tends to start from large unsatisfiable subsets and shrinks them to find MUSes with many SAT calls. Since single SAT calls are in practice not as expensive in boolean problems, MARCO's less aggressive seed-searching strat-

Solver	1 (s)	2 (s)	5 (s)
MARCO	230.85± 8.79	465.24± 16.41	1145.92± 36.0
GRAPE-MUST + MARCO	**231.35± 8.32**	**469.24± 15.5**	**1157.69± 34.03**
REMUS	891.28± 29.83	1933.27± 60.64	5188.03± 149.6
GRAPE-MUST + REMUS	**1171.74± 33.1**	**2400.03± 64.73**	**6055.5± 152.93**
TOME	156.08± 5.25	303.58± 9.89	723.29± 23.32
GRAPE-MUST + TOME	**175.9± 6.0**	**347.56± 11.74**	**848.8± 27.96**

Table 1: Average number of MUSes enumerated for random problems in 1, 2 and 5 seconds for different solvers with and without pruning. Bold indicates higher average.

Solver	1 (s)	2 (s)	5 (s)
MARCO	148.45± 16.72	288.07± 31.17	648.67± 65.36
GRAPE-MUST + MARCO	**163.58± 20.34**	**312.88± 37.33**	**680.44± 75.44**
REMUS	151.42± 14.19	322.12± 31.43	814.14± 77.16
GRAPE-MUST + REMUS	**245.25± 32.96**	**535.82± 72.2**	**1321.94± 160.1**
TOME	56.25± 3.68	109.92± 6.76	253.19± 13.64
GRAPE-MUST + TOME	**67.85± 5.73**	**130.8± 10.77**	**313.76± 24.07**

Table 2: Average number of MUSes enumerated for logistics planning problems in 1, 2 and 5 seconds for different solvers with and without pruning. Bold indicates higher average.

egy means it does not benefit as much from pruning as REMUS. TOME builds chains of subsets of the formula and looks for the smallest unsatisfiable one in the chain (Bendík et al., 2016). The sparsity induced by pruning may make it harder for TOME to find fully unsatisfiable chains, requiring it to perform binary search or build new chains more often. It is also important to note that TOME enjoys the weakest negative correlation between number of constraints and MUSes enumerated (Bendík and Cerná, 2018), making positive effects of pruning easier to mask.

These results, consistent with the 30-minute runs shown in the appendix, indicate that while GRAPE-MUST is beneficial to MUS enumeration in many cases, the choice of underlying solver matters and REMUS is consistently the best choice in our problems.

5.2 Extrapolation To Larger Problems

In this experiment we investigate the extrapolation capability of our model to see if it can successfully prune formulas of larger sizes than the ones used for training.

Experiment Settings We use the same model trained on randomly generated formulas with 100 variables as in section 5.1. We evaluate the model on formulas with 100, 150, 200, 250 and 300 variables. We evaluate the solvers on 500 formulas of each size and measure the absolute and relative improvement in MUS enumeration performance.

Results Figure 2 shows the relative and absolute improvement in MUS enumeration for each solver using GRAPE-MUST. Pruning generalizes well to larger problems, with the average problem size reduction only decreasing by about 7 percentage points from 100 to 300 variables. A table with size reduction figures and enumeration results is available in the appendix.

Consistent with previous results, pruning benefits REMUS significantly more than the other solvers, and here, MARCO benefits the least. Particularly, figure 2 (left) indicates that for REMUS the improvement is largest in formulas of around 200 clauses. However, in very large formulas the effect of pruning diminishes as even pruned problems become too large for the 5-second timeout. As shown in 2 (right), the number of MUSes found by all solvers in the largest formulas within the time limit is very small, resulting in large variance in the effect of pruning. Nevertheless, the results up to 200-250 formulas suggest that training GRAPE-MUST on smaller problems can be a viable strategy for improving the MUS enumeration performance of REMUS even in larger instances.

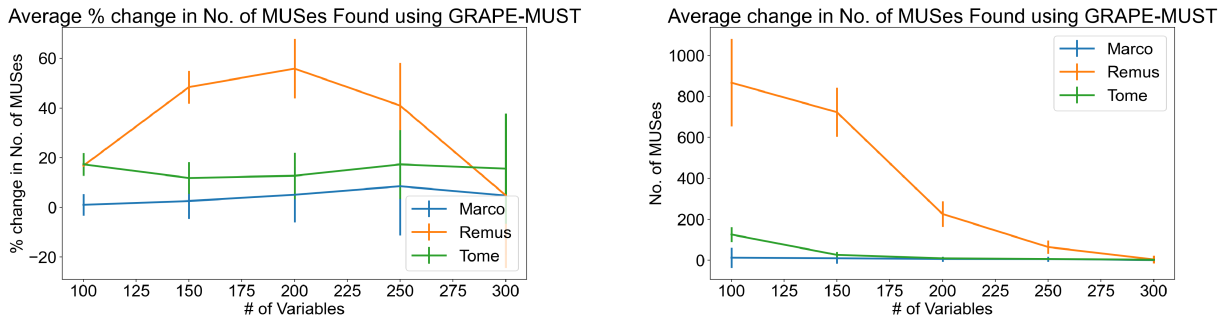
5.3 Performance Improvement In Benchmark Problems

In this experiment we evaluate a model trained on randomly generated formulas on a collection of hard problems from the 2011 SAT competition MUS enumeration benchmark problems.

Experiment Settings We scale up our model to use 6 hidden layers and a latent dimension of 128 units. We train the model on random formulas generated as described in section 4.5. We obtain the dataset statistics from the entire SAT Competition 2011 MUS track problem set and no additional information from the dataset is used for training. We train the model on 2 million formulas with 50 to 10000 variables and use $k = 100$ to compute the pruning at test time. We evaluate the model on 63 problems as described in

Solver	1 (s)	2 (s)	5 (s)
MARCO	113.97± 13.59	196.27± 24.27	439.55± 56.51
GRAPE-MUST + MARCO	**119.62± 16.92**	**231.34± 33.39**	**514.68± 74.22**
REMUS	216.13± 24.05	423.62± 50.3	971.27± 111.76
GRAPE-MUST + REMUS	**428.55± 65.89**	**958.15± 145.02**	**2371.07± 358.77**
TOME	**115.96± 13.99**	**195.96± 22.96**	**385.97± 42.59**
GRAPE-MUST + TOME	81.39± 11.35	154.72± 21.89	339.01± 46.81

Table 3: Average number of MUSes enumerated for graph coloring problems in 1, 2 and 5 seconds for different solvers with and without pruning. Bold indicates higher average.



(a) Percent improvement in MUS enumeration

(b) Absolute improvement in MUS enumeration

Figure 2: Extrapolation to larger formulas. Relative (a) and absolute (b) improvement over the base solvers using GRAPE-MUST on formulas of increasing size at a 5-second timeout. Despite the distribution shift, REMUS benefits proportionally more from pruning in formulas up to 200 variables as the smaller size of the pruned formulas accelerates enumeration. MARCO and REMUS maintain a small improvement across all formulas. As indicated by (b) the variance in (a) increases rapidly with formula size as the actual number of MUSes found becomes very small.

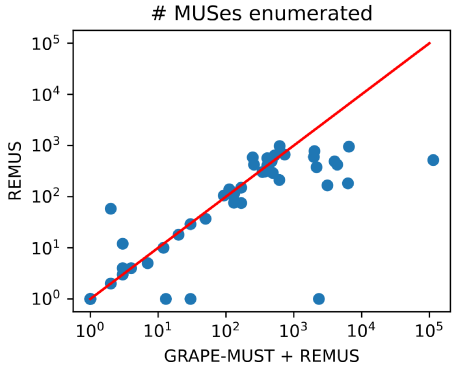


Figure 3: Comparison between REMUS and GRAPE-MUST+REMUS on hard benchmark problems. Red line indicates equal number of MUSes. Overall GRAPE-MUST enables the discovery of more MUSes.

the beginning of this section. We compare the performance of the highest performing enumerator (Bendík and Černá, 2020) REMUS with and without pruning using a timeout of 2 hours.

Results Figure 3 summarizes the performance of the two methods in the benchmark. Out of 63 problems, GRAPE-MUST enables the discovery of more MUSes in 30 problems, results in no change in 21 and in performance decrease in 12 problems. Interestingly in 2 problems in which REMUS alone finds no MUSes within the time limit, GRAPE-MUST enables the enumeration of 2349 and 30 MUSes respectively.

On average, GRAPE-MUST removes 9444 clauses from the problems, which corresponds to about 1% of the problem size. Compared to previous experiments this is a small reduction (see appendix). However, pruning 1% of randomly chosen clauses from the benchmark problems invariably yields satisfiable problems. This suggests that GRAPE-MUST is able to identify non-critical constraints in the benchmark problems despite the difference from the training distribution. In almost all failure cases, the pruning removes less than 0.1% of the clauses, indicating that the effort put into pruning the problems had minimal effect, only taking time away from MUS enumeration. Furthermore, comparisons against naive pruning heuristics (see appendix), show that GRAPE-MUST is more general and robust in real-world problems.

Overall this experiment indicates that training GRAPE-MUST on random formulas is a viable strategy for accelerating MUS enumeration in difficult real-world problems. We are therefore releasing the trained

GRAPE-MUST model along with the training code[2].

6 CONCLUSION

In this work we have introduced a method that uses learning-based graph pruning to accelerate the enumeration of MUSes from unsatisfiable CNF formulas. The main approach is converting CNF formulas to graphs and then formulating the pruning problem as a node labeling problem. To achieve this, we have designed a loss function and a differentiable objective to train a pruning model. Extensive experimental results show that MUS enumeration algorithms benefit from pruning in most cases, despite the possibility of destroying some MUSes. The learned model is also able to extrapolate to problems larger than ones in its training data. Interestingly, GRAPE-MUST trained on random formulas is able to generalize across data distributions, improving MUS enumeration performance on hard real-world problems without the need of large datasets.

This work opens up several possible research directions to further explore the use of learning models in accelerating MUS enumeration. First, future work can conduct a thorough investigation of the effect of destroying MUSes during pruning. We do not conduct such an investigation in this work due to the potentially exponential number of MUSes in unsatisfiable formulas and the challenge of finding even a small number of them in hard problems. However, as research in this area develops it is imperative that we better understand this trade-off and its effect on downstream tasks. For instance, recent work on approximate MUS counting (Bendík and Meel, 2020) can inform loss function design for pruning models to minimize the potential loss of MUSes.

Further investigation of the trade-off between pruning time and counting time may also be beneficial. In this work, we assume conditional independence between clauses when deciding on pruning, which limits the expressiveness of the model but enables more efficient training and inference. Future work can explore more complex models that capture the dependencies between clauses and evaluate their effectiveness in practice.

Another direction would be to generalize the proposed approach to other constraint satisfaction domains such as Satisfiability Modulo Theories (SMT). While solvers like REMUS, MARCO and TOME are agnostic to the underlying domain, our graph-pruning step relies on graph representations of boolean CNF formulas. Future work can devise compact representations for other domains, as well as models with appropriate inductive biases. Then, our training procedure can applied for problems in those domains.

[2]https://github.com/tufts-ml/GRAPE-MUST

Acknowledgments

We thank all anonymous reviewers for their constructive feedback. This work is partially supported by the NSF CAREER Award 2239869.

References

Abboud, R., Ceylan, I., and Lukasiewicz, T. (2020). Learning to reason: Leveraging neural networks for approximate dnf counting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 3097–3104.

Abboud, R., Ceylan, . ., Grohe, M., and Lukasiewicz, T. (2021). The surprising power of graph neural networks with random node initialization. In Zhou, Z.-H., editor, *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pages 2112–2118. International Joint Conferences on Artificial Intelligence Organization. Main Track.

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

Belov, A., Järvisalo, M., and Marques-Silva, J. (2013). Formula preprocessing in mus extraction. In *Tools and Algorithms for the Construction and Analysis of Systems: 19th International Conference, TACAS 2013, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2013, Rome, Italy, March 16-24, 2013. Proceedings 19*, pages 108–123. Springer.

Belov, A. and Marques-Silva, J. (2012). Muser2: An efficient mus extractor. *Journal on Satisfiability, Boolean Modeling and Computation*, 8(3-4):123–128.

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. and Černá, I. (2020). Must: minimal unsatisfiable subsets enumeration tool. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 135–152. Springer.

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.

Bendík, J. and Meel, K. S. (2020). Approximate counting of minimal unsatisfiable subsets. In *International Conference on Computer Aided Verification*, pages 439–462. Springer.

Clarke, E., Grumberg, O., Jha, S., Lu, Y., and Veith, H. (2000). Counterexample-guided abstraction refinement. In *International Conference on Computer Aided Verification*, pages 154–169. Springer.

Cohen, O., Gordon, M., Lifshits, M., Nadel, A., and Ryvchin, V. (2010). Designers work less with quality formal equivalence checking. In *Design and Verification Conference (DVCon)*.

Ghassabani, E., Whalen, M., and Gacek, A. (2017). Efficient generation of all minimal inductive validity cores. In *2017 Formal Methods in Computer Aided Design (FMCAD)*, pages 31–38. IEEE.

Goel, S., Shukla, Y., Sarathy, V., Scheutz, M., and Sinapov, J. (2022). Rapid-learn: A framework for learning to recover for handling novelties in open-world environments. In *2022 IEEE International Conference on Development and Learning (ICDL)*, pages 15–22. IEEE.

Guo, W., Yan, J., Zhen, H.-L., Li, X., Yuan, M., and Jin, Y. (2022). Machine learning methods in solving the boolean satisfiability problem. *arXiv preprint arXiv:2203.04755*.

Han, B. and Lee, S.-J. (1999). Deriving minimal conflict sets by cs-trees with mark set in diagnosis from first principles. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 29(2):281–286.

Hitzler, P., Eberhart, A., Ebrahimi, M., Sarker, M. K., and Zhou, L. (2022). Neuro-symbolic approaches in artificial intelligence. *National Science Review*, 9(6):nwac035.

Ignatiev, A., Previti, A., Liffiton, M., and Marques-Silva, J. (2015). Smallest mus extraction with minimal hitting set dualization. In *International Conference on Principles and Practice of Constraint Programming*, pages 173–182. Springer.

Khalil, E., Dai, H., Zhang, Y., Dilkina, B., and Song, L. (2017). Learning combinatorial optimization algorithms over graphs. *Advances in neural information processing systems*, 30.

Li, Z., Chen, Q., and Koltun, V. (2018). Combinatorial optimization with graph convolutional networks and guided tree search. *Advances in neural information processing systems*, 31.

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

Liu, L., Han, X., Zhou, D., and Liu, L.-P. (2022). Towards accurate subgraph similarity computation via neural graph pruning. *arXiv preprint arXiv:2210.10643*.

Ma, G., Ahmed, N. K., Willke, T. L., and Yu, P. S. (2021). Deep graph similarity learning: A survey. *Data Mining and Knowledge Discovery*, 35(3):688–725.

Morris, C., Ritzert, M., Fey, M., Hamilton, W. L., Lenssen, J. E., Rattan, G., and Grohe, M. (2019). Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 4602–4609.

Nadel, A., Ryvchin, V., and Strichman, O. (2014). Accelerated deletion-based extraction of minimal unsatisfiable cores. *Journal on Satisfiability, Boolean Modeling and Computation*, 9(1):27–51.

Paliwal, A., Loos, S., Rabe, M., Bansal, K., and Szegedy, C. (2020). Graph representations for higher-order logic and theorem proving. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 2967–2974.

Rintanen, J. (2014). Madagascar: Scalable planning with sat. *Proceedings of the 8th International Planning Competition (IPC-2014)*, 21:1–5.

Sato, R., Yamada, M., and Kashima, H. (2019). Approximation ratios of graph neural networks for combinatorial problems. *Advances in Neural Information Processing Systems*, 32.

Sato, R., Yamada, M., and Kashima, H. (2021). Random features strengthen graph neural networks. In *Proceedings of the 2021 SIAM International Conference on Data Mining (SDM)*, pages 333–341. SIAM.

Schuetz, M. J., Brubaker, J. K., and Katzgraber, H. G. (2022). Combinatorial optimization with physics-inspired graph neural networks. *Nature Machine Intelligence*, 4(4):367–377.

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*.

Shi, Y. and Zhang, Y. (2022). The neural network methods for solving traveling salesman problem. *Procedia Computer Science*, 199:681–686.

Toenshoff, J., Ritzert, M., Wolf, H., and Grohe, M. (2021). Graph neural networks for maximum constraint satisfaction. *Frontiers in artificial intelligence*, 3:580607.

Wang, X., Bo, D., Shi, C., Fan, S., Ye, Y., and Philip, S. Y. (2022). A survey on heterogeneous graph embedding: methods, techniques, applications and sources. *IEEE Transactions on Big Data*.

Welling, M. and Kipf, T. N. (2016). Semi-supervised classification with graph convolutional networks. In *J. International Conference on Learning Representations (ICLR 2017)*.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3):229–256.

Zhou, J., Cui, G., Hu, S., Zhang, Z., Yang, C., Liu, Z., Wang, L., Li, C., and Sun, M. (2020). Graph neural networks: A review of methods and applications. *AI Open*, 1:57–81.

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]

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

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]
   (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]
   (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

Problem Generation

In this section we provide details on the problem generation parameters for our experiments.

Problem	Pruning %
Random Formulas	32.22% ± 0.83%
Logistics Planning	36.27% ± 1.56%
Graph Coloring(4-7)	30.18% ± 1.00%

Table 4: Average percentage of the original formula pruned by GRAPE-MUST.

Random Problems We use the same generation procedure as (Selsam et al., 2018). As mentioned in the main text, we generate formulas with 30-100 variables for our first experiment and larger formulas for testing Extrapolation. In our problem generation we set the parameter of the geometric distribution to 0.3. This has the effect of generating formulas with slightly more literals per clause than in (Selsam et al., 2018). We find that this correlates with larger MUSes, which increases the difficulty of the search in our problem.

# Variables	Pruning %
100	30.22% ± 0.83%
150	29.16% ± 1.14%
200	27.68% ± 1.26%
250	24.28% ± 1.45%
300	23.48% ± 1.48%

Table 5: Average percentage of the original formula pruned by GRAPE-MUST.

Graph coloring The generation parameters are summarized in the text. We follow a standard SAT encoding for graph coloring problems: Each node is associated with at-least-1 and at-most-one constraints for the colors. Each edge is represented by K clauses for K colors, forbidding adjacent nodes to share that color. To plot pruned graph coloring formulas as graphs, the pruned formula is further processed for visualization using pure literal elimination to remove clauses that are trivially satisfiable.

Logistics planning The domain file for the logistics planning problem is available along-side our code. Unfortunately we could not obtain permission to distribute the planner with our code in time, however it is available online at https://research.ics.aalto.fi/software/sat/madagascar/ freely for academic purposes. We generate trajectories of up to 5 steps with 5 to 10 packages, 1 airplane, 2 airports, 2 cities, 2 locations, and 2 trucks, which are all matched randomly with each other in the initial and goal states. We use the following command line options to generate the problems: `-P 0 -1 -O -S 1 -F <steps>-1 -T <steps>`

Longer Timeout Experiment

In this section we show the results from a smaller scale experiment with a longer timeout. We use the same problem generation parameters as in the main text, but we use a 30 minute timeout instead. The results are shown in Table 7.

In random problems, pruned formulas still result in more MUSes found by Marco and Remus, but the difference is proportionally smaller than in shorter time limits. In fact, Tome finds almost the same number of MUSes in both formulas on average. Similarly, in logistics planning problems pruning still benefits all solvers but the effect is not as strong in the longer time limit. In graph coloring problems, pruning still offers significant benefit to Marco and Remus, while it seems to disrupt Tome's search strategy consistent with our experiment in section 5.1.

Overall, while even in longer time limits solvers with different search strategies interact with GRAPE-MUST differently, the combination GRAPE-MUST+REMUS consistently outperforms other methods in our experiments. Furthermore, the results from random and logistics planning problems indicate that given enough time, the speed-up due to pruning may diminish. However, as demonstrated in section 5.3, in realistic problems with a standard time limit of 2 hours GRAPE-MUST offers significant benefit to MUS enumeration in most cases.

Extrapolation Experiment Additional Tables

Here we show the reduction and enumeration results for the extrapolation experiment in additional detail.

Table 5 Shows that formula pruning performance is fairly consistent even in larger problem sizes, with the pruning percentage decreasing by about 7 percentage points when the problem, size is tripled. This indicates that the pruning method that the model has learned is able to generalize to larger problems and can therefore be used to accelerate MUS enumeration on larger problems.

Table 6 Shows the detailed MUS enumeration performance before and after pruning for the extrapolation experiment. As discussed in the main text, GRAPE-MUST is able to accelerate MUS enumeration especially for REMUS in larger problems. Interestingly, around 200 variables, the improvement of REMUS is

Solver	100 vars.	150 vars.	200 vars.	250 vars.	300 vars.
MARCO	1145.92± 36.0	348.56± 17.84	103.0± 8.13	55.6± 7.52	19.33± 3.31
GRAPE-MUST + MARCO	**1157.69± 34.03**	**357.43± 17.98**	**108.21± 8.47**	**60.31± 8.7**	**20.25± 3.31**
Remus	5188.03± 149.6	1493.73± 77.36	404.66± 35.97	156.41± 19.74	63.29± 14.78
GRAPE-MUST + REMUS	**1157.69± 34.03**	**2216.71± 91.92**	**630.75± 50.52**	**220.49± 25.72**	**66.38± 11.69**
Tome	723.29± 23.32	218.99± 9.81	68.02± 4.56	30.31± 3.03	11.13± 1.75
GRAPE-MUST + TOME	**848.8± 27.96**	**244.87± 11.11**	**76.67± 5.03**	**35.56± 3.42**	**12.86± 2.02**

Table 6: Enumeration results from the extrapolation experiment.

Solver (30 mins.)	Random Problems	Logistics Planning	Graph Coloring
Marco	96220 ± 7129	172933.72 ± 16285.18	128638.94 ± 23702.68
GRAPE-MUST + Marco	97254 ± 6822	177356.60 ± 16373.36	176703.96 ± 31061.28
Remus	484033 ± 25554	82774.54 ± 5690.20	8965.48 ± 1121.03
GRAPE-MUST + Remus	504056 ± 28010	87825.64 ± 6239.49	15525.12 ± 3201.24
Tome	73644 ± 3318	24403.12 ± 2843.69	72476.06 ± 16174.54
GRAPE-MUST + Tome	73442 ± 3719	25903.40 ± 2842.90	54981.82 ± 12340.48

Table 7: Average number of MUSes found by each solver in 30 minutes.

even higher than previous results, indicating a sweet-spot between extrapolation ability and problem size.

Pruning Statistics

Table 4 shows the average percentage of clauses pruned by our model. We can see that GRAPE-MUST actually prunes a significant fraction of the clauses in the original formulas. We have also tried a random pruning method that randomly removes the same fraction of clauses, and the resulting formulas are mostly satisfiable. This provides further evidence that our pruning model can effectively identify non-critical constraints.

Comparison With Naive Pruning

We devise the following two naive strategies and apply them to the collection of hard problems from the 2011 SAT competition MUS enumeration benchmark used in the main text. To make the comparison with our method fair, both strategies involve O(1) SAT calls with respect to the problem size. We use the following two strategies to rank clauses:

- Clause-length based: Given a problem $S = (U, C)$ Let $l_{max} = \max_{c \in C} |c|$ and $l_{min} = \min_{c \in C} |c|$. Then, taking K=100 equally sized integer steps between l_{max} and l_{min} search for the smallest clause length l^* such that $C' = c \in C, |c| \leq l^*$ and $S' = (C', U')$ is unsatisfiable. Return S'.
- Variable frequency based: Compute the frequency in which each variable (and its negation) appear in the clauses of S. Then, each clause is scored by the negative average frequency of the variables (and their negations) in it. As a result, clauses with variables appearing in few other clauses receive higher scores. Then, prune the formula as in the previous strategy, only using clause scores instead of length to decide pruning. This replaces scores produced by our GNN.

The first strategy produces on average formulas 0.08% smaller than the original formulas, with most problems not being successfully pruned. Most problems failed to be pruned after a single SAT call, taking little time from MUS enumeration. As a result, 55 of the benchmark problems have the same number of MUSes enumerated between the two methods. The clause-length strategy improves enumeration in 5 problems and reduces the number of MUSes found in 3 problems. Interestingly, in one of the problems where the first strategy succeeds, REMUS finds 1 MUS, GRAPE-MUST + REMUS finds 13 and Strategy 1 + REMUS finds 134. This may suggest that for some problems, naive heuristics like clause-length may be viable strategies, however it is not clear how to decide in which problems to use it and it does not generalize well. Since GRAPE-MUST is a learning model trained on difficult random problems, its learned heuristic is much more widely applicable, even if better problem-specific heuristics may exist. The second strategy prunes no formulas successfully, and so we do not run MUS enumeration.