A Machine Learning Approach to Model Counting

Marco Dalla
SFI Centre for Research Training in AI
School of Computer Science & IT
University College Cork, Ireland
m.dalla@cs.ucc.ie

Andrea Visentin
Insight Centre for Data Analytics
School of Computer Science & IT
University College Cork, Ireland
andrea.visentin@ucc.ie

Barry O'Sullivan
Insight Centre for Data Analytics
School of Computer Science & IT
University College Cork, Ireland
b.osullivan@cs.ucc.ie

***Abstract*—Model counting (#SAT) is the problem of computing the number of satisfying assignments for a given Boolean formula. It has a significant theoretical and practical interest. Tackling it can be challenging since the number of potential solution grows exponentially with the number of variables. Due to the inherent complexity of the problem, approaches to approximate model counting have been developed as a practical alternative. These methods extract the number of solutions within user-specified tolerance and confidence levels and in a fraction of the time required by exact model counters. However, even these methods require extensive computations, restricting their applicability to relatively small instances. In this paper, we propose a new approximate machine learning model counter that overcome this limitation. Predicting the number of solutions can be seen as a regression problem. We deploy an array of machine learning techniques trained to infer the approximate number of solutions based on statistical features extracted from a SAT propositional formula. Extensive numerical experiments performed on synthetic crafted and benchmark datasets show that learning approaches can provide a good approximation of the number of solutions with a much lower computational time and resource cost than the state-of-the-art approximate and exact model counters. Making it possible to approximate the model count of instances previously out of reach. We then investigated the structural factors that lead to a high model count using AI explainability approaches.**

***Index Terms*—Model counting, Machine learning, SAT problem, Feature analysis**

I. Introduction

The *Boolean Satisfiability Problem (SAT)* is a fundamental problem in computer science and mathematics. It deals with determining whether there exists an assignment of truth values (true or false) to variables in a given Boolean formula such that the formula evaluates to true. A Boolean formula consists of variables, logical operators (such as AND, OR, NOT), and parentheses for grouping. The SAT problem is significant because it is the first problem proven to be NP-complete, meaning it is one of the hardest problems in computational complexity theory.

A generalization of the satisfiability problem is Propositional Model Counting or #SAT, the canonical #P-complete problem [1]. The #SAT problem asks for the number of satisfying assignments for a given Boolean formula. This problem is inherently harder than the SAT problem itself since it requires counting all possible satisfying assignments, which could be exponentially large in the worst case. The #SAT problem is also NP-hard, meaning that if there existed a polynomial-time algorithm for solving #SAT, then P = NP. Thus, #SAT is considered even more challenging than SAT.

Theoretical work on #SAT has focused on the worst-case complexity of counting problems. Results have shown that even instances that can be decided in polynomial-time, such as 2-SAT, are hard for model counting [2]. Therefore, practical techniques need to be developed to approach this problem efficiently. Many practical applications can be translated into model counting problems, like bounded-length adversarial and contingency planning [3], probabilistic reasoning [4], as well as the study of hard combinatorial problems, such as combinatorial designs [5], where the number of solutions provides further insights into the problem.

Two of the most effective SAT-solving techniques, the *Davis-Putnam-Logemann-Loveland (DPLL)* algorithm and local search, can be extended to solve #SAT problems. However, some issues arise with scalability as most common SAT heuristics are designed to narrow down the search to reach a single solution quickly. In contrast, most #SAT algorithms need to be mindful of all the solutions in the search space. The search for better model counting algorithms and challenging instances led to the creation of the *Model Counting Competition* in 2020 [6], where the most recent developments, e.g. [7], are benchmarked. Approximate model counters have been developed to tackle problems out of reach of the complete algorithms. While they generally require fractions of the computational cost of complete approaches, due to the exponential complexity of the problem their applicability is still limited.

In recent years, machine and deep learning methods have been introduced into the study of SAT problems. Machine learning techniques have been injected directly into SAT solvers for parameter settings [8], branching heuristics [9], and restart policies [10]. They have also been used in the prediction of satisfiability [11] as well as the development of a deep learning heuristic solver [12]. To the best of our knowledge, the only attempt at tackling the approximate model counting as a regression problem is [13], in which an architecture based on the graph neural network for belief propagation is used to predict the number of solutions.

In this paper we present an explainable and easily deployable #SAT regressor developed using existing open-source libraries. The goal is to extrapolate an explainable model that approximates the model count. We compute a set of statistical features from SAT instances and train a Random

Forest Regressor to predict the total number of solutions. A comparison is then performed between learning algorithm, the state-of-the-art approximate model counter, *ApproxMC* [14] and the scalable probabilistic exact model counter *GANAK* [15]. We train the algorithm both on synthetic datasets of hard model counting instances and publicly available benchmarks [14], [16]–[18]. Results show that these techniques can provide an approximation of the number of solutions that is within the confidence interval of modern approximate model counters, as well as deliver a model count in less than a millisecond when the state-of-the-art model counters are unable to do so within a 5000 seconds time-out. Moreover, after the initial training time, the time to process new instances is almost instantaneous, regardless of the difficulty of the #SAT problem. Finally, we compare these feature-based machine learning techniques with the state-of-the-art deep learning algorithm and show that our method outperforms them on the task of approximating the number of solutions. This approach can be used to extend the reach of approximate model counting to realistic size application, or to provide an approximate solution in case an exact solver times out.

II. Related Work

We present an overview of the state-of-the-art in model counting and the different approaches that have been applied to this particular task thus far. Then, we focus on machine learning methods applied to the Boolean Satisfiability problem, as well as feature extraction techniques.

Modern exact model counters use a set of different heuristics and techniques to deal with relatively complex formulas. Some of those include caching the results of solved subsets of problems [19], dynamically dividing the remaining formulas into sub-problems [20], caching their counts [4] and combining the caching of components with standard *Conflict-Driven Clause Learning (CDCL)* to prune the search [21]. Another viable approach is knowledge compilation, where the original propositional formula is converted into a different representation in which the model count can be more efficiently computed [22], [23]. While these techniques have been shown extremely successful, the memory usage and runtime of such algorithms often scale extremely poorly with problem size.

In order to deal with larger instances, new methods for obtaining an approximate evaluation of the number of solutions have been developed. These new model counters can be categorized into three different groups: (ϵ,δ) counters; lower (or upper) bounding counters; guarantee-less counters. The first category is the one that has been mostly investigated in the recent literature, with successful approaches such as Monte-Carlo approximations [24] and *ApproxMC* [14]. Building on the foundation of dynamic decomposition and knowledge compilation techniques that have dominated the model counting landscape, *GANAK* [15] introduces a groundbreaking approach by integrating probabilistic component caching and several novel heuristics. This innovative architecture allows *GANAK* to significantly outperform state-of-the-art exact model counters not only in terms of efficiency and scalability but also in delivering exact model counts across a wide range of benchmarks.

The lower (or upper) bound counters provide a number of solution that is higher (or lower) than c with a probability of $1 - \delta$. Examples are [5] and [25]. The final category features counters that do not provide any guarantees that can be very efficient and deliver practical good approximations [26], [27]. These algorithms use different sampling techniques in order to reduce the amount of space searched and therefore scale much better than their exact counterpart.

The method to calculate the number of favourable assignments to propositional formulas described in this paper does not feature counting algorithms with different heuristics but rather a learning approach. As mentioned before, Boolean Satisfiability has been shown to benefit from machine learning and deep learning methods. For example, the successful portfolio SAT solver *SATzilla* [28] leverages a manually extracted dataset of 48 structural and statistical features to perform ridge regression and fit a runtime prediction function to perform algorithm selection. Another example is demonstrated in a 2008 paper by Devlin and O'Sullivan [11]. They treated the SAT problem as a classification task and used machine learning algorithms trained on the same feature set used by SATZilla. An attempt to perform the same classification task with automatically extracted features was performed in [29]. In 2014 Grozea et al. [30] employed machine learning models to correctly predict branches while searching for a solution for 3-SAT instances. They find that in over 90% of the test cases using a machine learning heuristic-enhanced SAT solver reduces by at most 1/3 the number of branchings that the algorithm performed during the search. A 2018 seminal paper [31] by Selsam et al. introduces *Neurosat*, a message-parsing graph neural network that learns to provide a solution to SAT instances by only being trained to predict their satisfiability. While this approach has yet to outperform state-of-the-art SAT solvers, it can scale efficiently with the size of the problem and handle instances with completely different structures from the ones it is trained on. A similar model has been used by [13]; they present an architecture based on the graph neural network framework for belief propagation to predict the number of solutions tacking as input from a graph representation of the SAT instance. Their approach is trained on generated instances and outperforms ApproxMC. However, it requires a tailor-made network and does not provide any explanation.

III. Method

In this section, we describe the experimental setup that was employed. The following two subsections will discuss in-depth, respectively, the dataset and the algorithms employed.

A. Dataset

For the empirical section of this paper, we used three datasets. The first one, which we will later refer to as **Dataset 1**, was generated using the tool described in [32]. The main reason for generating the dataset is the necessity for a sizeable

number of instances with a controlled number of variables and clauses. Machine learning algorithms depend strongly on the training dataset's quality and representativeness. We opted for using small instances regarding the number of variables and clauses since the approximate model counter requires a considerable amount of time to compute the solution, and the graphs needed to compute some statistical features have large memory footprints. Finally, we required a smooth distribution for the decimal logarithm of the number of solutions of the instances. The logarithm is considered as it is the target for the different model counters in the Model Counting competitions.

The tool employed for generating this dataset is capable of producing propositional formulas in CNF DIMACS format, which have been benchmarked during the International Model Counting Competition [6]. The generator is highly parameterisable: the number of variables of the instances it produces, their number of clauses and the number of literals in each clause can all be set. Furthermore, the generated instances have been tested against other difficult model counting instances on state-of-the-art model counters. Results showed that the smallest unsolved instances of the competition, both in terms of the number of variables and clauses, were generated using this procedure. The tool allows for three types of instances to be generated: completely random k-CNF instances, random CNF instances into which exactly one solution is placed and random instances with an added cluster of solutions. An in-depth description of the algorithms used can be found at [32].

The second dataset, identified as **Dataset 2**, was generated using the procedure explained in [13]. We decided to employ this dataset to compare our machine learning algorithms and the deep learning procedure employed by the authors. For each clause, then an average of 5 variables are chosen and negated with a 50% probability. We refer to the original paper for a complete description of the procedure needed to assemble the dataset. The number of solutions was provided as a target for the learning algorithm and was extracted using *Cachet* [21] and *Sharpsat* [33] model counters.

The third dataset (**Dataset 3**) consists of 2031 publicly available instances from diverse applications of model counting, including probabilistic reasoning, plan recognition, DQMR networks, ISCAS89 combinatorial circuits, quantified information flow, logistics, and more [14], [16]–[18]. Out of the 2031 we have been able to provide an exact model count only for 1597, using *d4*, *gpmc* and *Cachet* [6]. This was due to our limited computing resources. These benchmarks have previously been utilized in model counting studies, ensuring a comprehensive and challenging testbed for assessing our learning algorithm scalability and efficiency performances. The main statistical properties of the three datasets are shown in Tables I.

B. Machine Learning Algorithm

The instances were processed using the *SATFeatPy* tool [34]. This library implements three widely common feature extractors and provides a wide range of statistical, structural and probing features of a CNF that help describe the complexity and structure of the instance. Multiple graph representations of the instances are also extracted (Variable-Clause Graph, Binary Implication Graph, Resolution Graph, etc.), yielding further insight into the properties of each instance. For each instance, a total amount of 331 features have been extracted. After the feature extraction, a collinearity test was performed to select only independent features. Two tests were used: *Pearson's Correlation Coefficient (PCC)* and *Variance Inflation Factor (VIF)*. For the sake of explainability, a further step is applied to select the variables that would contribute to the decision. For this purpose, the *Boruta* Feature Selection Algorithm [35] is applied, reducing the total features from 331 to 47. The retained features are computed from the variable-clause graph and the variable graph, as well as DPLL and local search probing features. A reduced set of features reduce the risk of overfitting and provides better explanations.

The features selected from the training set of instances were then fed to a *Random Forest Regressor algorithm (RFR)*. The hyperparameter for the *RFR* have been chosen after a cross-validated grid search. *RFR* has been choosen after evaluating a variety of regressors. It had the highest accuracy and provides a good level of explainability.

For the experiments performed on **Dataset 3**, only the features belonging to [28] have been extracted, as the remaining ones have proven to be too computationally heavy for this dataset. The training target is the decimal logarithm of the number of solutions.

C. Deep Learning: Graph Neural Network

To compare our feature-based machine learning approaches with the existing state-of-the-art learning methods applied to model counting, we leverage the architecture implemented in [13]. Here, a Graph Neural Network (GNN) framework for belief propagation (BP) is combined with a self-attentive mechanism to create a network that can approximate the solution to #SAT problems. This network will be referred to as *BPGAT*. The key aspects of the proposed approach are:

- Belief Propagation (BP) based GNN framework: The authors leverage a GNN framework that is designed for belief propagation, which is a popular technique for probabilistic inference in graphical models. The GNN framework is adapted to handle the #SAT problem, allowing for efficient message passing and information sharing between nodes in the graph.
- Self-attentive GNN: The architecture is extended with a self-attentive mechanism that allows nodes in the graph to weigh the importance of their neighbours' information when updating their own states. This self-attention mechanism helps the GNN capture more complex relationships between variables and clauses in the Boolean formula.
- The model is trained on a small set of random Boolean formulae, which enables the GNN to learn a general representation of the problem space.

Further information about the specific architecture of the network is available at [13]. We deploy this architecture on **Dataset 2** and compare the results on the test set with *RFR*,

	Dataset 1		Dataset 2		Dataset 3
	Train	Test	Train	Test	
# of instances	3000	450	1300	300	1597
# of clauses	[40−500]	[40−500]	[20−50]	[20−50]	[10−2.06·10^6]
# of variables	[40−100]	[40−100]	[10−30]	[10−30]	[15−3.31·10^5]
Mean(log10(c))	10.00	10.05	4.473	4.325	1971
Range(log10(c))	[-1−20.8]	[-1−20.2]	[0−7.86]	[0−7.60]	[0.30−66136]

TABLE I: Datasets' statistics for training and testing. c is defined as the total number of solutions. The value -1 has been chosen to define a #SAT problem without solutions, i.e. an unsatisfiable instance

ApproxMC and *GANAK*. To run the experiments, we used the code published by the authors at [1]. Because of the design of the *BPGAT* and its limited scalability, the network can only be tested on this dataset. We were not able to produce further datasets using the code included in the paper [13], and the authors did not provide the dataset when requested.

D. Approximate Model Counter

A comparison with the state-of-the-art approximate model counter is performed to assess the learning algorithm's performance. *ApproxMC* [14] is a hashing-based algorithm for approximate discrete integration over finite domains and provides (ϵ,δ) guarantees. The probability that the approximate count is within a certain distance of the true count is evaluated by the following equation:

$$P(\frac{S}{1+\epsilon} \leq c \leq S(1+\epsilon)) \geq 1-\delta \tag{1}$$

with S the actual number of solution, c the approximated number of solutions and $\epsilon, \delta \in [0,1]$

In other words, it provides a count of solutions with a certain probability of correctness within a given threshold. For our experiments, we set the configurations of guarantees to ϵ = 0.8 and δ = 0.2, i.e. the default configuration used by the authors as well as δ = 1. This second parameter configuration effectively transforms *ApproxMC* into a guarantee-less model counter.

The decimal logarithm is then calculated from the estimated number of solutions that are obtained by running *ApproxMC*. Finally, this is compared using the regression metrics with the decimal logarithm of the actual number of solutions.

E. Probabilistic Exact Model Counter

GANAK stands out as a state-of-the-art tool in the field of probabilistic exact model counting, harnessing the power of universal hash functions and a novel probabilistic component caching system. Unlike traditional model counters that may rely on deterministic methods, *GANAK* introduces a degree of probabilistic reasoning into the exact counting process, allowing it to provide counts with a specified level of confidence. This is made possible by the delta parameter, δ, which in the context of *GANAK*, dictates the confidence with which the model count is asserted to be exact. The innovative use of such probabilistic techniques enables *GANAK* to tackle complex counting problems with remarkable efficiency and scalability. For our experimental setup, *GANAK* was configured to run with two distinct delta values: δ = 0.2 and δ = 1.

The evaluation involves computing the decimal logarithm of *GANAK*'s model counts and comparing these against the decimal logarithms of the known true counts using standard regression metrics, thereby quantifying *GANAK*'s accuracy and reliability in model counting tasks.

IV. Experimental Results

This section empirically compares the machine learning and deep learning algorithm to a state-of-the-art approximate model counter and probabilistic exact model counter. We evaluate them in terms of regression accuracy and computational effort required to calculate the prediction. Using the trained learning models, we investigate the statistical features that mainly affect the number of solutions and the possibility of approximating the problem with a close formula. All the experiments have been performed using a single core on a *Intel(R) Xeon(R) Gold 6240 CPU 2.60GHz*. The machine learning algorithm is trained on statistical features extracted from the instances of the training sets and evaluated on the test sets for **Dataset 1** and **Dataset 2**. For **Dataset 3** a 10-fold cross-validation for the *RFR* has been performed on the total number of instances. This decision is due to the large diversity of instances in this set. The graph neural network was trained for 600 epochs using a *NVIDIA Quadro RTX P6000* graphic card on the **Dataset 2** training set. The set was split into 1000 training instances and 300 validation instances. This was done to replicate the training procedure of [13]. *ApproxMC* and *GANAK* are run with both configuration on **Dataset 1** and **Dataset 2** testing set and on the whole **Dataset 3**. A timeout of 5000 seconds per instance is enforced for the model counters.

A. Accuracy of the Approximation

Tables II, III and IV shows the performance of the learning models versus the approximate model counter. The learning approaches are sorted by decreasing the models' complexity; simpler models are generally more transparent.

GANAK delivers the best overall performance in the three datasets. In **Dataset 1** and **Dataset 2** the probabilistic exact model counter provides the same exact number of solutions irrespective of the parameter δ. It is worth noting though that for the first set GANAK was not able to reach a solution within 5000 seconds for 21 instances in case δ = 0.2 and 15 for δ = 1. The machine learning methods show a MAE, RMSE and

[1]https://github.com/GaiaSaveri/GNN-sharpSAT

	Dataset 1 Test			
	MAE	RMSE	R2	MAPE
ApproxMC				
$\delta = 0.2$, $\epsilon = 0.8$	0.020	0.025	0.999	0.002
$\delta = 1$	0.054	0.068	0.999	0.006
GANAK				
$\delta = 0.2, 1$	$7.988{\cdot}10^{-9}$	$1.037{\cdot}10^{-8}$	1	$7.801{\cdot}10^{-10}$
RFR	0.097	0.208	0.996	0.020

TABLE II: Accuracy comparison for Dataset 1. For this table, only 429 instances out of 450 are considered, as those are the only one where ApproxMC and GANAK are able to provide a count within the 5000 seconds timeout.

	Dataset 2 Test			
	MAE	RMSE	R2	MAPE
ApproxMC				
$\delta = 0.2$, $\epsilon = 0.8$	0.034	0.046	0.999	0.003
$\delta = 1$	0.054	0.068	0.999	0.006
GANAK				
$\delta = 0.2, 1$	$1.853{\cdot}10^{-11}$	$1.208{\cdot}10^{-10}$	1	$5.376 \cdot 10^{-12}$
RFR	0.125	0.173	0.991	0.043
BPGAT	0.166	0.228	0.985	0.085

TABLE III: Accuracy comparison for Dataset 2.

MAPE of an order-of-magnitude greater than the state-of-the-art approximate model counter. The Random Forest Regressor, based on features extracted from the instances outperform the deep learning approach with a Graph Neural Network. Both *ApproxMC* and the learning algorithms perform better on **Dataset 1**. We observe that by relaxing the confidence parameter in the approximate model counter the result worsen by a 2-3 factor. The learning algorithms have metrics that are worse by factors between 2-10, with especially the RMSE and the MAPE that tend to magnify instances with larger errors. This behaviour is highlighted in Figures 1 and 2. Here, the plots show the distribution for the two testing datasets of the *GANAK* model count, with a grey interval representing the targeted guarantees for *ApproxMC*, the approximation of *ApproxMC*, and the prediction of the Random Forest Regressor and the Graph Neural Network Regressor. We can see that while not as accurate as the approximate model counter, the learning algorithms provide an estimate that is within the boundaries for most of the solutions. In fact, if we evaluate (1) using the adopted guarantee of $\epsilon = 0.8$, we obtain a probability for the predictions of the *RFR* to be inside the guarantee interval of respectively 0.91 for **Dataset1** and 0.85 for **Dataset 2**, which is consistent with the one expected for a $\delta = 0.2$.

Dataset 3 incorporates a large number difficult model counting instances, stemming from a wide range of applications. We observe here that *GANAK* still provides overall the best results. *ApproxMC* with guarantees actually has a better value of MAE, and very similar performance in terms of RMSE. The *RFR* reaches very close performances with the guarantee-less *ApproxMC*, providing a better value of RMSE. Most important, the learning algorithm is able to output a number of solutions for all of the 1597 instances, while the timeout of

	Dataset 3			
	MAE	RMSE	R2	MAPE
ApproxMC				
$\delta = 0.2$, $\epsilon = 0.8$	0.772	6.622	0.998	0.027
$\delta = 1$	1.937	23.876	0.979	0.043
GANAK				
$\delta = 0.2, 1$	1.568	5.682	0.998	0.005
RFR	2.884	23.357	0.980	0.134

TABLE IV: Accuracy comparison for Dataset 3. For this table, only 993 instances out of 1597 are considered, as those are the only one where ApproxMC and GANAK are able to provide a count within the 5000 seconds timeout.

5000 seconds is reached for over 204 and 201 instances for respectively *ApproxMC* with or without guarantees, and 476 for *GANAK*. The distribution of the model count values for the different techniques is shown in 3.

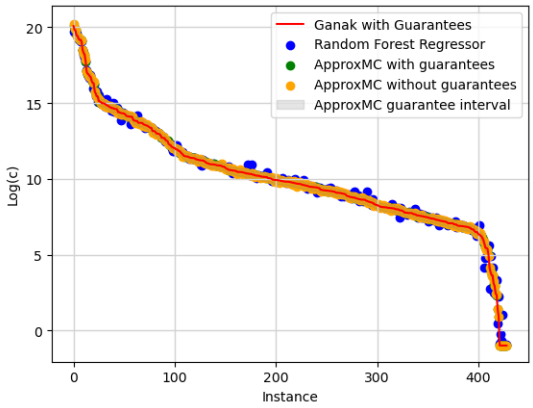

Fig. 1: Plot of GANAK solutions, ApproxMC approximate solutions (δ = 0.2, 1), ApproxMC guarantee interval, and Random Forest Regression predicted solutions for **Dataset 1** testing instances. The decimal logarithm of number of solutions (Y-axis) for each instance (X-axis) are plotted in descending order.

B. Computational Time

Table V presents the computational times in seconds required to produce the results of the previous section. *ApproxMC* and *GANAK* does not require a training phase and can be directly deployed on the test instances. For the machine learning approach, the steps needed to predict the number of solutions of a CNF are feature extraction of the training set, model fitting, feature extraction of target CNF, and model prediction. For the deep learning approach, the steps include the transformation of the CNF into a factor graph, the training of the model and model prediction. The strength of learning algorithms is their ability to provide almost instantaneous prediction after a training time. This characteristic makes them very interesting when the computational power and time are limited. In our case, most of the time is needed to extract the features from the instances; the tool used is

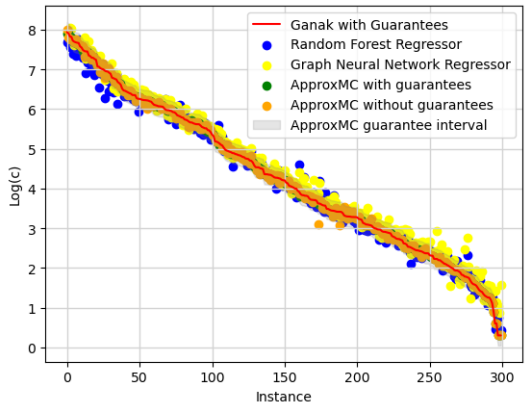

Fig. 2: Plot of GANAK solutions, ApproxMC approximate solutions (δ = 0.2, 1), ApproxMC guarantee interval, Graph Neural network and Random Forest Regression predicted solutions for **Dataset 2** testing instances. The decimal logarithm of number of solutions (Y-axis) for each instance (X-axis) are plotted in descending order.

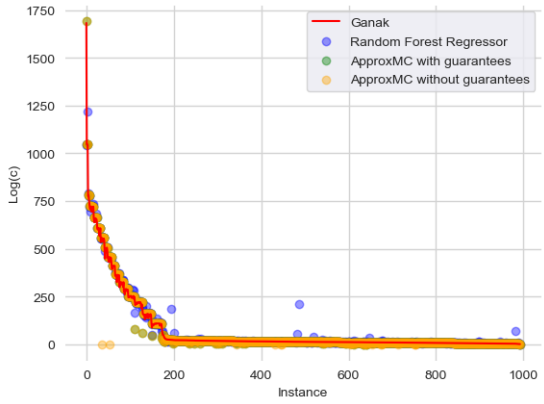

Fig. 3: Plot of GANAK solutions, ApproxMC approximate solutions (δ = 0.2, 1), and Random Forest Regression predicted solutions for **Dataset 3** testing instances. The decimal logarithm of number of solutions (Y-axis) for each instance (X-axis) are plotted in descending order.

a prototype implemented in *Python*. We observe that a part from **Dataset 2**, where the size of the instances is very small and the algorithmic approach is extremely quick, the feature extraction and training time is always much more efficient. Moreover, while model counters do not scale well and require a considerable amount of computing time and power for larger instances, the feature extraction is extremely consistent and scales much more efficiently with the problem size.

Finally, for both **Dataset 1** and **Dataset 3**, both model counters reached the 5000 seconds timeout on multiple instances, thus failing to provide an answer, while our approach can make a prediction in less than one millisecond. The training and inference phase is also significantly faster for the machine learning algorithms with respect to the BPGAT, as training a deep learning algorithm with a complex architecture requires

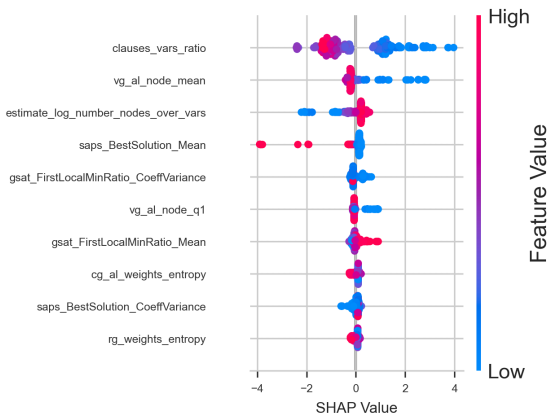

Fig. 4: SHAP analysis performed for the Random Forest Regressor on Dataset 1 testing set

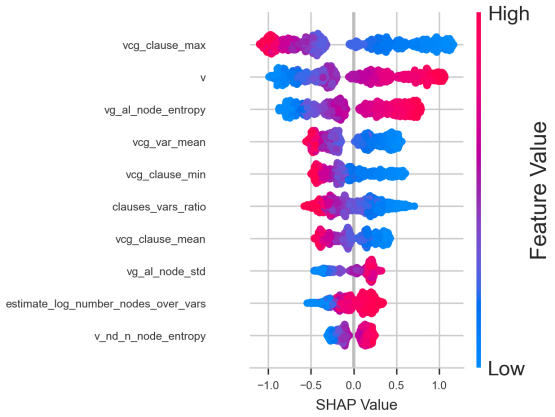

Fig. 5: SHAP analysis performed for the Random Forest Regressor on Dataset 2 testing set

multiple days on dedicated hardware.

C. Feature Importance

In this section, we investigate the statistical features that are mostly affecting the regressors' decisions. Understanding their relevance in the estimation process can lead to interesting insights into which factors mostly affect the number of solutions of a SAT instance. This information can be used to engineer additional features specifically relevant for predicting the model count, and to provide solvers developers with information that can be used in the search. We use *SHAP (SHapley Additive exPlanations)* to analyse the results and increase their explainability. SHAP is a popular and powerful method for interpreting machine learning models. It provides a way to explain the predictions of a model by assigning an importance value to each feature in the input data. These values represent the contribution of each feature to the final prediction made by the model. SHAP analysis calculates these importance values using a game-theoretic approach called Shapley values, which ensures that the values are fair and additive across all features. By using SHAP analysis, we can gain a better understanding

		ApproxMC $\delta = 0.2$		ApproxMC $\delta = 1$		Ganak $\delta = 0.2$		Ganak $\delta = 1$		Feature Extraction		RFR	BPGAT
		Mean	Max	Mean	Max	Mean	Max	Mean	Max	Mean	Max		
Dataset 1	Train	-	-	-	-	-	-	-	-	2.37	2.98	355.98	-
	Test	38	3324	12.80	152.59	677	4823	674	4958	2.27	2.86	10^{-4}	-
Dataset 2	Train	-	-	-	-	-	-	-	-	2.31	2.69	478.12	$7{\cdot}10^{5}$
	Test	0.05	0.31	0.013	0.04	0.071	0.43	0.069	0.26	2.17	2.61	10^{-4}	1.84
Dataset 3	Train	-	-	-	-	-	-	-	-	2.37	2.98	521.3	-
	Test	5.72	1172	3.82	689	45.30	3860	45.08	3881	2.23	2.89	10^{-4}	-

TABLE V: Mean and maximum computational time (in seconds) compared between the approximate and probabilistic exact model counters, with different confidence levels, and the machine and deep learning approaches. Test values are per individual instance.

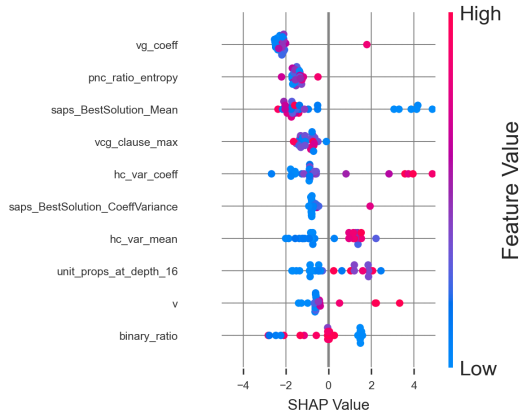

Fig. 6: SHAP analysis performed for the Random Forest Regressor on Dataset 3

of how a model works and in which direction the factors are driving its predictions. In Figure 4, 5 and 6, we can observe the importance of the features for the different datasets. In **Dataset 1**, the most relevant feature is the average number of clauses in which a variable appears; a higher value of this ratio leads to a smaller number of solutions. This is intuitive since being involved in fewer constraints allows the variables to have multiple assignments that do not make the formula unsatisfiable, and so increases the number of solutions. The second most important feature belongs to the set stemming from the variable graph of the instance. The third feature, *estimate_log_number_nodes_over_vars*, is of particular interest. This feature belongs to the DPLL probing feature set. It is an estimate of the number of nodes in the search tree, which can be regarded as an approximate measure of the search space. Other features that have a considerable impact on the total number of models are *gSat_BestAvgImprovement_Mean*, *gSatFirstLocalMinStep_Mean*, *saps_BestSolution_Mean* and *saps_BestSolution_CoeffVariance*. These are obtained by probing the search space with the stochastic local search algorithms GSAT and SAPS. The algorithms are run multiple times until the search trajectory reaches a plateau that cannot be escaped within a given number of steps. It seems that measurements of statistics and estimations of the search space have a significant contribution to the calculation of the total number of solutions.

For **Dataset 2**, the most relevant features belong to the ones extracted respectively from the Variable Clause Graph and Variable Graph representation of the instances. These encapsulate the structure of the instance, while *v* (the number of variables) is associated with its size. Unsurprisingly, instances with more variables have a higher number of solutions. It is worth noting as well that *estimate_log_number_nodes_over_vars* and *clauses_vars_ratio* provide important contributions to this dataset as well.

For **Dataset 3**, the most important features belong to both graph sets and probing sets. Especially for this dataset, which incorporate a large number of instances with different structures and characteristic, it is shown that both a graph representation of the CNF, as well as an accurate estimation of the search size space are extremely important to provide a correct prediction of the number of solutions.

In the analysis, we can see that the features selected are dataset dependent. However, information relative to the number of variables and clauses, and estimantions of the search space size are fundamental regardless of the instance type. This insight suggests that improving features estimating the search space would lead to a more accurate approximate model counting.

D. Model Count Guarantees

ApproxMC employs a hashing-based partitioning technique to probabilistically estimate the number of satisfying assignments of a Boolean formula. Utilizing universal hash functions, it partitions the solution space into manageable subsets, counts the solutions within these partitions, and aggregates the results to provide an overall estimate. The methodology offers probabilistic guarantees on the accuracy of the estimate, ensuring that the derived count is within a specified confidence interval of the true solution count with high probability.

In contrast, machine learning algorithms, due to their inherent reliance on empirical data and non-deterministic optimization processes, cannot provide analogous exact probabilistic guarantees on their predictions or classifications. However, we are currently exploring the potential of providing probability distributions for machine learning regressors in the context of model counting. The aim is to bridge the gap between

deterministic model counting tools like ApproxMC and the probabilistic nature of machine learning predictions, offering a more nuanced understanding of the solution landscape.

V. Conclusions

In this paper, we presented a new learning approach to the problem of model counting based on feature extraction. In this approach, machine learning algorithms predict the number of solutions of a SAT instance after being trained on statistical and structural features of a crafted dataset. We used widely studied regression techniques and compared the results with the state-of-the-art approximate model counter and a graph deep learning regressor.

The empirical study shows that our approach can outperform both in terms of evaluation metrics for regression and computation time for the existing state-of-the-art deep learning technique. Moreover, while still not as accurate as the state-of-the-art approximate model counter, it can provide a number of solutions within the confidence bounds of *ApproxMC* and perform a prediction in a time that in some cases is orders of magnitude lower, and therefore could be used as a back-up solution for particularly difficult instances. Furthermore, statistical and structural features provide insight into which characteristics of the instance tend to influence the total number of solutions. Our approach offers performance that is useful in practical settings, despite there being no guarantee that the predicted number of models is within an interval near the correct number of solutions.

Our work opens a variety of research directions. First of all, a further investigation of the extracted features, particularly the probing ones, and the extraction of different estimations of the search space might lead to increasingly accurate predictions. It would also be of particular interest to see if these techniques can aid the current model counting solution to achieve better and faster results. Moreover, a further improvement would be to provide guarantees to the correctness of the solution in a similar way that approximate model counters produce. Finally, it would be beneficial to see if learning techniques can be used on different representations of the instances to evade the arbitrariness of the particular extracted features.

Acknowledgments

This publication has emanated from research conducted with financial support of Science Foundation Ireland under Grant 12/RC/2289-P2, and 18/CRT/6223, which are co-funded under the European Regional Development Fund.

References

[1] S. Toda, "On the computational power of PP and (+)P," *30th Annual Symposium on Foundations of Computer Science*, pp. 514–519, 1989.

[2] V. Dahllöf, P. Jonsson, and M. Wahlström, "Counting models for 2SAT and 3SAT formulae," *Theoretical Computer Science*, vol. 332, no. 1, pp. 265–291, 2005.

[3] J. Rintanen, "Planning as satisfiability: Heuristics," *Artificial Intelligence*, vol. 193, pp. 45–86, 2012.

[4] F. Bacchus, S. Dalmao, and T. Pitassi, "Algorithms and complexity results for #SAT and bayesian inference," in *Proceedings of FOCS*, pp. 340–351, 2003.

[5] C. P. Gomes, A. Sabharwal, and B. Selman, "Model counting: A new strategy for obtaining good bounds," in *Proceedings of AAAI*, p. 54–61, 2006.

[6] J. K. Fichte, M. Hecher, and F. Hamiti, "The model counting competition 2020," *Journal of Experimental Algorithmics (JEA)*, vol. 26, pp. 1–26, 2021.

[7] R. Kiesel and T. Eiter, "Knowledge compilation and more with sharpsat-td," in *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning*, vol. 19, pp. 406–416, 2023.

[8] F. Beskyd and P. Surynek, "Parameter setting in SAT solver using machine learning techniques," in *Proceedings of ICAART 2022* (A. P. Rocha, L. Steels, and H. J. van den Herik, eds.), pp. 586–597, 2022.

[9] J. H. Liang, V. Ganesh, P. Poupart, and K. Czarnecki, "Learning rate based branching heuristic for SAT solvers," pp. 123–140, 2016.

[10] J. H. Liang, C. Oh, M. Mathew, C. Thomas, C. Li, and V. Ganesh, "Machine learning-based restart policy for CDCL SAT solvers," pp. 94–110, 2018.

[11] D. Devlin and B. O'Sullivan, "Satisfiability as a classification problem," *Proc. of the 19th Irish Conf. on Artificial Intelligence and Cognitive Science*, 2008.

[12] D. Selsam and N. Bjørner, "Guiding high-performance SAT solvers with unsat-core predictions," vol. 11628 LNCS, pp. 336–353, 2019.

[13] G. Saveri and L. Bortolussi, "Graph neural networks for propositional model counting," *arXiv preprint*, 2022.

[14] S. Chakraborty, K. S. Meel, and M. Y. Vardi, "Algorithmic improvements in approximate counting for probabilistic inference: From linear to logarithmic SAT calls," in *Proceedings of the International Joint Conference on Artificial Intelligence*, pp. 3569–3576, 2016.

[15] S. Sharma, S. Roy, M. Soos, and K. S. Meel, "Ganak: A scalable probabilistic exact model counter," in *Proceedings of the International Joint Conference on Artificial Intelligence*, pp. 1169–1176, 7 2019.

[16] F. Brglez, D. Bryan, and K. Koiminski, "Combinational profiles of sequential benchmark circuits," *IEEE International Symposium on Circuits and Systems,*, pp. 1929–1934 vol.3, 1989.

[17] T. Sang, P. Beame, and H. A. Kautz, "Performing bayesian inference by weighted model counting," in *The 20th AAAI Conference on Artificial Intelligence* (M. M. Veloso and S. Kambhampati, eds.), pp. 475–482, 2005.

[18] J. Lagniez, E. Lonca, and P. Marquis, "Improving model counting by leveraging definability," in *Proceedings of the International Joint Conference on Artificial Intelligence* (S. Kambhampati, ed.), 2016.

[19] S. M. Majercik and M. L. Littman, "Using caching to solve larger probabilistic planning problems," in *Proceedings of AAAI*, p. 954–959, 1998.

[20] R. J. B. Jr. and J. D. Pehoushek, "Counting models using connected components," in *Proceedings of AAAI*, pp. 157–162, 2000.

[21] T. Sang, F. Bacchus, P. Beame, H. A. Kautz, and T. Pitassi, "Combining component caching and clause learning for effective model counting," in *SAT*, 2004.

[22] A. Darwiche, "Decomposable negation normal form," *Journal of the ACM (JACM)*, vol. 48, no. 4, pp. 608–647, 2001.

[23] C. J. Muise, S. A. McIlraith, J. C. Beck, and E. I. Hsu, "Dsharp: Fast d-DNNF compilation with sharpsat," in *Proceedings of the Canadian AI Conference* (L. Kosseim and D. Inkpen, eds.), vol. 7310, pp. 356–361, Springer, 2012.

[24] R. M. Karp, M. Luby, and N. Madras, "Monte-carlo approximation algorithms for enumeration problems," *Journal of Algorithms*, 1989.

[25] L. Kroc, A. Sabharwal, and B. Selman, "Leveraging Belief Propagation, backtrack search, and statistics for model counting," *Proceedings of ISAIM*, 2008.

[26] S. Ermon, C. Gomes, and B. Selman, "Uniform solution sampling using a constraint solver as an oracle," in *Proceedings of UAI*, p. 255–264, 2012.

[27] W. Wei and B. Selman, "A new approach to model counting," in *Theory and Applications of Satisfiability Testing* (F. Bacchus and T. Walsh, eds.), (Berlin, Heidelberg), pp. 324–339, 2005.

[28] L. Xu, F. Hutter, H. H. Hoos, and K. Leyton-Brown, "SATzilla: portfolio-based algorithm selection for sat," *Journal of artificial intelligence research*, vol. 32, pp. 565–606, 2008.

[29] M. Dalla, A. Visentin, and B. O'Sullivan, "Automated SAT problem feature extraction using convolutional autoencoders," pp. 232–239, 12 2021.

[30] C. Grozea and M. Popescu, "Can machine learning learn a decision oracle for np problems? a test on SAT," *Fundamenta Informaticae*, vol. 131, pp. 441–450, 2014.
[31] D. Selsam, M. Lamm, B. Bünz, P. Liang, L. de Moura, and D. L. Dill, "Learning a SAT solver from single-bit supervision," in *Proceedings of ICLR 2019*, 2019.
[32] G. Escamocher and B. O'Sullivan, "Generation and prediction of difficult model counting instances," *CoRR*, vol. abs/2212.02893, 2022.
[33] M. Thurley, "sharpSAT – counting models with advanced component caching and implicit bcp," vol. 4121, pp. 424–429, 07 2006.
[34] M. Dalla, B. Provan-Bessell, A. Visentin, and B. O'Sullivan, "Sat feature analysis for machine learning classification tasks," in *Proceedings of the International Symposium on Combinatorial Search*, vol. 16, pp. 138–142, 2023.
[35] M. B. Kursa and W. R. Rudnicki, "Feature selection with the boruta package," *Journal of Statistical Software*, vol. 36, no. 11, p. 1–13, 2010.