Goose: A Meta-Solver for Deep Neural Network Verification*

Joseph Scott[1,*], Guanting Pan[1], Elias B. Khalil[2] and Vijay Ganesh[1]

[1]University of Waterloo, Ontario, Canada
[2]University of Toronto, Ontario, Canada

Abstract

The verification of deep neural networks is a recent algorithmic challenge that has attracted significant interest, resulting in a wide array of complete and incomplete solvers that draw on diverse techniques. As is typical in hard search problems, no single solver is expected to be the fastest on all inputs. While this insight has been leveraged to boost Boolean Satisfiability (SAT), for instance, by combining or tuning solvers, it is yet to lead to a leap in the neural network verification domain.

Towards this goal, we present Goose, a meta-solver for deep neural network verification. Goose's architecture supports a wide variety of complete and incomplete solvers and leverages three key meta-solving techniques to improve efficiency: algorithm selection, probabilistic satisfiability inference, and time iterative deepening. Using Goose we observe an 47.3% improvement in PAR-2 score across over 800 benchmarks and 13 solvers from VNN-COMP '21.

Keywords

Neural Network Verification, Meta-Solving, Algorithm Selection, AI Safety

1. Introduction

Software engineers increasingly leverage machine learning (ML) due to its empirical ability to implement some software functions from data [1]. This paradigm-shifting approach is increasingly incorporated into safety-critical systems such as driverless cars and drones [2]. In response, there have been several research initiatives to build systems that can give formal guarantees on the behavior of such learned models.

For example, consider the adversarial robustness problem. A desirable property of a ML system is to fit the training data while being robust to small adversarial perturbations "in the neighbourhood" of the training data. Surprisingly, researchers have shown that in the domain of driverless cars, a deep neural network can be tricked into thinking a stop sign is a speed-limit or green light if an adversary can put pieces of tape onto it [3]. Recently, several logical solvers

SMT 2022: Satisfiability Modulo Theories, August 11–12, 2022, Haifa, Israel

* You shall use this document as the template for preparing your publication. We recommend using the latest version of the ceurart style.

joseph.scott@uwaterloo.ca (J. Scott); g6pan@uwaterloo.ca (G. Pan); khalil@mie.utoronto.ca (E. B. Khalil); vijay.ganesh@uwaterloo.ca (V. Ganesh)
https://www.joe-scott.net/ (J. Scott); https://ca.linkedin.com/in/guanting-tony-pan-162452176 (G. Pan); https://ekhalil.com/ (E. B. Khalil); https://ece.uwaterloo.ca/~vganesh/ (V. Ganesh)
0000-0002-4145-1612 (J. Scott); 0000-0003-4108-9032 (G. Pan); 0000-0001-5844-9642 (E. B. Khalil); 0000-0002-6029-2047 (V. Ganesh)

have been constructed to find these adversarial inputs or give a guarantee that they do not exist [4, 5, 6, 7, 8, 9, 10, 11].

As the deep neural network verification community grows, an emergent body of decision procedures and semi-decision procedures has emerged. For a practitioner, it can be overwhelming to determine the most suitable verifier for the desired application and instance at hand. For example, complete algorithms we consider herein include: Reluplex – a lazy unwinding of ReLu activations within the simplex algorithm [4]; nnenum – a bound over-approximation algorithm on starsets [12]; and Eager Translation – Direct translation of the problem into a Satisfiability Modulo Theories (SMT) or Mixed Integer Linear Program (MILP) [13, 7]. Additionally, incomplete techniques that we consider include one based on abstract-refinement [6], auto_LiRPA [8, 10], projected gradient descent (PGD)[14, 15], and fuzzing.

This problem of *algorithm selection* [16] is not unique to deep neural network verification. For example, consider the competition-winning variant of the cvc5 SMT solver[1]. On just a single logic (e.g., UF), 23 configurations of the cvc5 tool alone in a statically ordered sequential portfolio are used to determine the satisfiability of the underlying benchmark. The authors acknowledge this in the latest system description, which outlines speculative research initiatives to implement better portfolios [17].

In this paper, we present Goose[2], a *meta-solver* – a solver designed around calling several subsolvers for deep neural network verification. Goose implements three meta-solving techniques. First, ML-driven *algorithm selection* based on empirical hardness models for runtime prediction. Algorithm selection solutions have had tremendous empirical success in logic solver communities [18, 19, 20]. Second, *probabilistic satisfiability inference* – a ML approach to determine which subproblems are most likely satisfiable. Finally, *time iterative deepening* an exponentially increasing wallclock timeout of the ML constructed sequential portfolio.

Contributions. Specifically, this paper makes the following contributions:

1. **The Goose Tool.** We present Goose, a tool for deep neural network verification (description Section 3 and architecture diagram Figure 1). The value-addition of Goose is its meta-solver architecture, which implements a comprehensive approach to algorithm selection, probabilistic satisfiability inference, and time interval deepening.

2. **Evaluation on VNN-COMP '21:** We demonstrate the efficiency of Goose with a competition-like evaluation over more than 800 instances from VNN-COMP '21 against 13 competition solvers [21] such as α, β-CROWN [8, 10, 11], Verinet [13], ERAN [6], and Marabou [22] (Section 4). We observe that Goose improves over the competition winner, α, β-CROWN , by 47.3% in PAR-2 score (Figure 2, Table 1).

2. Preliminaries

The Open Neural Network Exchange (ONNX) and VNN-LIB. ONNX is a research and industrial initiative to standardize machine learning models [23]. In the latest version (V13),

[1] https://github.com/cvc5/cvc5/blob/smtcomp2021/contrib/competitions/smt-comp/run-script-smtcomp-current
[2] Etymology: after the Canadian Goose (bird) in honor of Canada

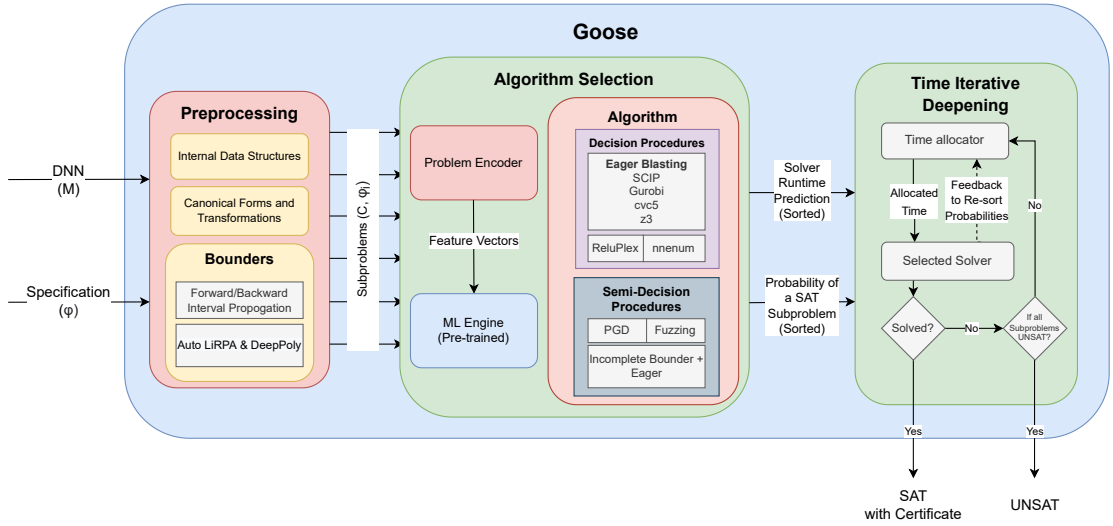

Figure 1: Architecture Diagram of of Goose (See description in Section 3.1).

syntax and semantics of computation graphs are outlined for 156 operations (e.g., ReLU, Gemm). ONNX is widely supported across major deep learning platforms such as TensorFlow [24], PyTorch [25], and Keras [26].

The VNN-LIB is an international initiative with the aim of supporting neural network verification research [27]. In the context of deep neural network verification, a verification query requires two parts, a computation graph C and a specification ψ. For the former, VNN-LIB defines syntax and semantics leveraging a subset of 17 ONNX operations, with the latest competition benchmarks leveraging 15 operators.

Problem Representations. Not all of the recently developed verifiers are compliant with arbitrary linear properties. For example, several solvers do not support disjunctions or conjunctions or linear real/integer constraints over the input/output. To overcome this, we leverage a recent equivalence result:

Theorem 1 ([28]). *Let C be a computation graph and let ψ be a linear specification over the input/output behaviour of C (disjunctions, conjunctions, negations, and linear constraints). Then there exists subproblems ψ_i such that $\Psi = (\psi_1, ...\psi_n)$ and each ψ_i is of the form*

$$x \in \mathcal{X}' \wedge y \in \mathcal{Y}' \wedge Ay \leq b$$

for some interval $\mathcal{X}', \mathcal{Y}'$, matrix A, and vector b, and input/output x/y.

This theorem allows for us to construct a transformer $\mathcal{T}$ of the original problem to create an equivalent disjunction. We will leverage this in multiple ways within Goose. A consequence, however, is a worst case exponential blowup in the number of disjunctions.

3. Goose

In this section, we describe Goose, a meta-solver for neural network verification; an architecture diagram is shown in Figure 1. We first provide a high-level description of the tool before developing each of its component in more detail.

Goose leverages an empirical model E to make predictions on runtimes and probabilities of satisfiability. This model is trained offline and requires data collection. We elaborate on this process in Section 4. In what follows, we assume the empirical model has already been trained.

The input to Goose is a computation graph C– a symbolic representation of the deep neural network of interest –and a linear specification ψ over the input/output behaviour of C over all $x \in \mathcal{X}$. Goose outputs UNSAT if and only if $\forall x \in \mathcal{X}, C(x) \models \psi$. Goose supports the input formats `.onnx` for computation graphs and `.vnnlib` for specifications.

At the beginning, Goose loads the input into its respective internal data structures and converts them into a disjunction over a canonical form (Theorem 1). Next, Goose featurizes the input and leverages (trained) empirical hardness models to determine which solver should be used; we refer to the aforementioned modules of Goose as the "Meta Empirical Engine". Solving the algorithm selection task in this way allows for a novel scheme in which the meta-engine can infer empirical probabilities of satisfiability for each of the individual disjuncts; then, the problem corresponding to the disjunct that is most likely to be SAT is attacked first. Furthermore, Goose employs a *time iterative deepening* which is inspired from the AI algorithm *incremental deepening* [29] on two-player games. The intuition is that in practice, runtimes are either usually either very short or long. Hence, when running a portfolio of algorithms, start with a shorter wallclock portfolio timeout, and exponentially increase it if the portfolio failed. While this may seem wasteful, it can be empirically effective.

3.1. Solver Architecture

Input/Output. The input to the Goose system is a computation graph C (in `.onnx` format) and a specification over its input-output behaviour ψ (in `.vnnlib` format). We assume ψ restricts the domain of C to be bounded in $\mathcal{X}$ (e.g., $0 \leq x_i, y_i \leq 1$). On a successful run, Goose outputs UNSAT if and only if $\forall x \in \mathcal{X}, C(x) \models \psi$, otherwise SAT.

Preprocessing. Goose has two key preprocessing steps. First, the inputs C and ψ are parsed and loaded into flexible graph and specification classes. These classes were designed to have significant utility to be later leveraged by a decision procedure. The second step is the conversion into a canonical form. Specifically, we implement a transformer $\mathcal{T}$ that implements Theorem 1 so that

$$\psi = \bigvee^{n} \psi_i,$$

where each *subproblem* ψ_i is composed of interval constraints on the input ($\mathcal{X}$) and output ($\mathcal{Y}$), and a halfspace polytope constraint $Ay \leq b$, for some A, b computed by $\mathcal{T}$. By leveraging $\mathcal{T}$, we achieve a canonical form as each $\psi_i := x \in \mathcal{X}' \wedge y \in \mathcal{Y}' \wedge Ay \leq b$, for a computed $\mathcal{X}', \mathcal{Y}', A, b$.

Meta Empirical Engine. Goose implements a ML-driven engine to make predictions from a dataset of empirical performance measurements that is constructed once in advance. The engine is composed of three key components. First, a data management scheme which includes select training data and online data collection schemes. Online data collection and retraining can be useful when solving problems on unseen problem classes. We however do not consider this in the paper. This can be done easily via the command-line interface and programmatic API with user adjustable resource constraints. The second major component is the problem encoder, $\xi(C; \psi_i)$ that converts the input to a real valued feature vector. This feature vector is extendable by the user[3]. Out-of-the-box, Goose has 500 features on the problem format[4]. The last major component is the ML backend. Goose is platform-agnostic, abstracted around a base class. We have support for and evaluation with `pytorch` [25], but allow for `scikit-learn` [30], and XGBoost [31] solutions.

Algorithms. Goose implements several decision procedures and semi-decision procedures and configurations thereof. One widely spread technique is a direct or eager translation to a core solver, such as MILP. Goose further supports two other complete algorithms, namely, Reluplex and nnenum. This is done by leveraging Marabou's and nnenum's python API. Goose supports semi-decision procedures Projected Gradient Descent (PGD), random fuzzing, and over-approximation based bounding leveraging an eager translation.

Bounders. Several decision procedures require that every operation is tensor-component-wise bounded. For example, consider the $y = \mathrm{ReLU}(x) = \max(0, x)$ activation function. If y can be bounded by $\ell \leq y \leq u$, then this can be expressed in MILP [7] or SMT (QF_LIA) as:

$$(y \leq x - \ell(1 - a)) \wedge (y \geq x) \wedge (y \leq u \cdot a) \wedge (y \geq 0) \wedge a \in \{0, 1\}$$

Goose includes a forward/backward interval arithmetic [32, 33] driven bounding system, as well as auto_LiRPA and DeepPoly. A bounding algorithm is either exact or an overapproximation. If a complete decision procedure requires a bounding, and is given an over-approximation, then the decision procedure becomes incomplete. We compute two sets of boundings, via exact bounding algorithms and over-approximation bounding algorithms. The final bounding for each is computed via an intersection.

Eager Blasting[5]. One of the more sophisticated modules of Goose is its eager blasting engine. The engine was designed to be agnostic to the underlying core solver. Goose implements a base class that interacts with the engine. We provide support for the SCIP [34] and Gurobi [35] MILP solvers in addition to the cvc5 [17] and z3 solver [36] (QF_LIRA).

Execution Loop. Goose implements a main solving loop that leverages all the above components. It is outlined in Algorithm 1 and described in Section 3.2. The key value-added of Goose

[3]Note that this requires retraining

[4]The feature vector includes a one hot encoding of built in solvers and their configurations. The value-addition of this is to not require complete empirical labelling over all decision procedures

[5]We borrow this term from the SMT community

Algorithm 1 The main execution loop of Goose

Input: A computation graph C and a linear specification ψ over C
Output: SAT/UNSAT

procedure Goose-MainLoop
  $\mathcal{P} = \mathcal{T}(C, \psi)$ ▷ $\mathcal{T}$ is the transform from Theorem 1
  $\mathcal{F} = [\bot\ \forall C, \phi_i\ \in \mathcal{P}]$
  Sort $\mathcal{P}$ in decreasing order of SAT probabilities predicted by E
  $t = t_{init}$
  solved = $\bot$
  while not solved **do**
    for C, ψ_i in $\mathcal{P}$ **do**
      if $\mathcal{F}[i]$ **then**
        continue
      end if
      Sort $\mathcal{S}$ in ascending order by predicted runtimes by E.
      α_s = time allocation for each $s \in \mathcal{S}$ determined by E, t.
      for $s \in \mathcal{S}$ **do**
        $\rho = \texttt{run}(s, C, \psi_i, \alpha_s)$
        if ρ is SAT **then**
          return SAT
        else if ρ is UNSAT **then**
          $\mathcal{F}[D, \psi] = \top$
        end if
      end for
    end for
    t += an exponential increment
    Re-sort $\mathcal{P}$ leveraging ξ, E and empirical feedback ▷ Optional
    solved = $\bigwedge_{v \in \mathcal{F}} v$
  end while
  return UNSAT
end procedure

comes from three main features: First, its ability to leverage ML driven *algorithm selection*. Second, its *time iterative deepening* strategy, which calls solvers with exponentially increasing wallclock timeouts. We do not believe this has been considered in the logic solver context. Third, *probabilistic satisfiability inference* leveraging the same ML model for algorithm selection, except the label vector is extended to include SAT and UNSAT class labels. This allows for us to infer probabilities of satisfiability of each ψ_i of ψ. As the ML model uses a feature vector including the one-hot-encoding of the solver, an average over the set of solvers is taken. We are not aware of such a scheme in the logic solver context.

3.2. Algorithmic Description

We next describe Algorithm 1, the main execution loop of Goose. For input C, ψ, Goose implements the transformation $\mathcal{T}$ corresponding to Theorem 1, specifically, $\mathcal{T}(C, \psi)$ computes the subproblems over the disjunction as specification graph pairs (line 2). Additionally, a flag $\mathcal{F}$ is used to denote whether or not a subproblem is solved (line 3). Goose then invokes the meta-engine to use the problem encoder ξ over all subproblems to create feature vectors[6] Upon computing all feature vectors, we query the learnt model E to compute the probabilities of SAT and predicted runtimes over the set of considered decision procedures S. This is used to sort $\mathcal{P}$ by probability of satisfiability (line 4).

From here, we start the main loop (line 5). The termination condition is the conjunction over $\mathcal{F}$ (i.e., whether or not all problems in $\mathcal{P}$ are UNSAT) or if there exists a SAT subproblem. On the first iteration of the main loop, we first iterate over each $C, \psi_i \in \mathcal{P}$. We sort the set of decision procedures $\mathcal{S}$ such that they are in descending order of inferred probability of satisfiability (line 12).

On each iteration of the main loop, a global wallclock timeout for the iteration t initialized to t_{init} (line 5) and is exponentially increased after each iteration (line 23). From here, we leverage E and the computed features over the subproblems to determine a resource allocation scheme α_s for $s \in \mathcal{S}$. We compute this by taking a softmin [7] over the predicted runtimes from the empirical model E multiplied by the wallclock time limit t for this iteration divided by subproblems (line 13). We next run the solver and save the result ρ (line 5). On a successful run (i.e., $\rho \in \{\text{SAT}, \text{UNSAT}\}$), we either terminate on SAT or on UNSAT mark the problems flag $\mathcal{F}$ as solved (line 19). At the end of the iteration of the loop, if the problem remains unsolved, we exponentially increase the time increment.

Implementation Details. Goose is built on python 3.8 and can be run via command-line or python API. Preprocessing was built leveraging the onnx python packages. The meta-engine was built with the assistance of `pandas`, `pytorch`, and `scikit-learn`. We implement our forward and backward interval internally and leverage `auto_LiRPA` for incomplete bounding verification and handling of select activations.

4. Evaluation

In this section, we present an empirical evaluation of Goose over VNN-COMP '21 instances and its computation environment.

4.1. Experimental Setup

Empirical Model Architecture. We construct the empirical model using PyTorch. The empirical model is an 8-layer fully-connected neural network with 1024 neurons per layer for a total of 25M parameters. Each hidden layer is composed of a linear layer with bias, batch normalization, dropout (p=0.25), and ReLU activations.

[6] In the worst case, an exponential number of of sub-problems and feature vectors need to be computed, however, this is rarely observed in practice.

[7] $\text{softmin}(\vec{x}) = \exp(-x_j)/\sum_j \exp(-x_j)$

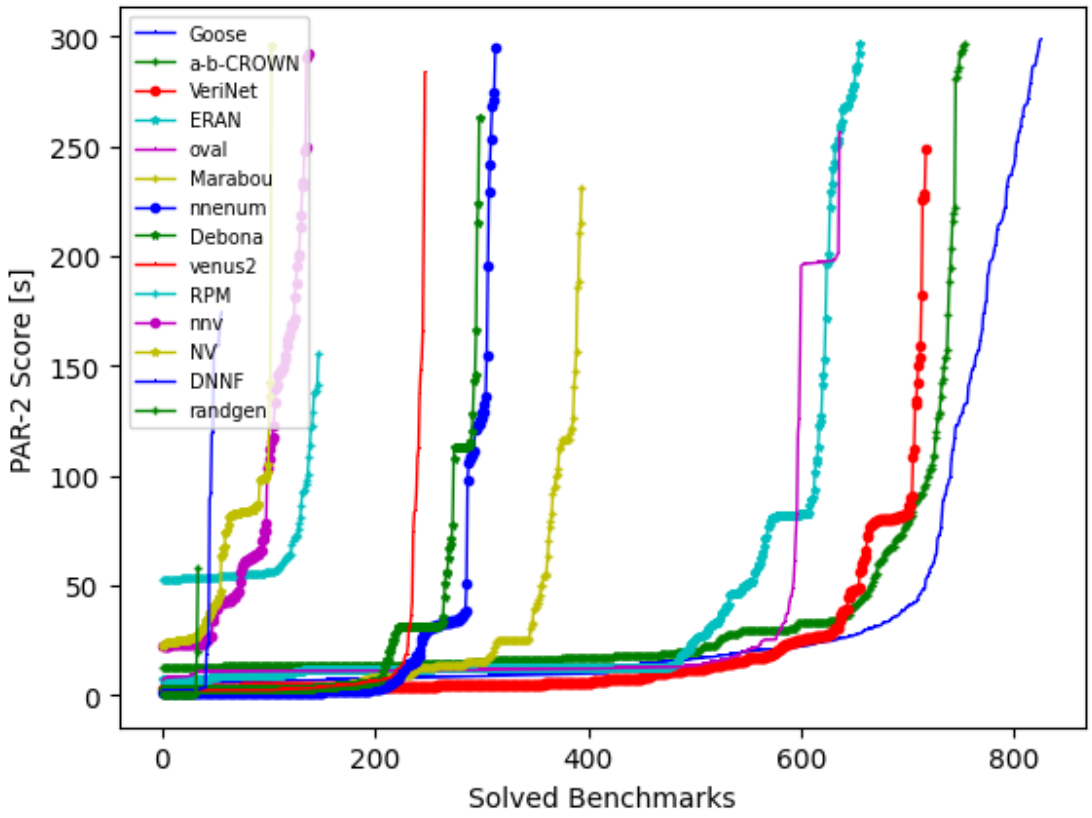

Figure 2: Main experimental cactus plot over VNN-COMP '21 benchmarks (Section 4). A cactus plot is a visualization of a solver's performance on a benchmark suite the horizontal axis represents the number of benchmarks solved (higher is better) and the vertical axis is the benchmark wise PAR-2 (lower is better).

Empirical Model Training Setup. We generate a training dataset of 10,000 instances produced by using a random fuzzer over VNN-LIB [37]. An instance is generated and solved across all considered solvers, with a 30-second timeout[8]. This data collection is performed on Compute Canada [38], particularly on a CentOS V7 cluster of Intel Xeon Processor E5-2683 running at 2.10 GHz with 8 GB of memory. Wallclock runtimes are rounded to the nearest second. All solvers were ran sequentially. For labels we use log PAR-2 scores. A PAR-2 is the wallclock runtime if successful, else twice the wallclock timeout. The empirical model was trained with Adam [39] and a learning rate of $4 \cdot 10^{-4}$, mean squared error loss, and $1 \cdot 10^{-4}$ weight decay on a NVIDIA1080 GPU for 2 hours.

Computation Environment. The evaluation was ran on the Amazon Web Service (AWS). Per VNN-COMP rules, since some tools are either CPU or GPU based, there are two different types of AWS instances depending on the solver [21]. We evaluate `Goose` on the GPU instance.

[8]This is not used in the main experiment, and is only lowered to increase data collection times

Solver	PAR-2 Score	Solver	PAR-2 Score
Goose	42917.306	Debona	342215.706
α-β CROWN	81455.553	venus2	364639.422
VeriNet	96584.104	RPM	435868.435
ERAN	143568.969	nnv	440426.117
oval	148884.067	NV	458472.933
Marabou	285863.380	DNNF	483079.800
nnenum	333131.499		

Table 1
Table of sums of PAR-2 scores across the solvers from the empirical evaluation (Section 4). The PAR-2 score of a solver on a benchmark is the wallclock runtime if successful, otherwise twice the wallclock runtime (lower is better).

Solvers. For complete decision procedures we use the following: Reluplex (via Marabou), nnenum, Eager Blasting + SCIP, Eager Blasting + Gurobi, Eager Blasting + cvc5, Eager Blasting + z3. For incomplete procedures, we consider two configurations of PGD and two configurations of random fuzzing. Additionally, we consider two variants of incomplete bounding within `auto_LiRPA` for both Gurobi and SCIP. This yields a set of $|\mathcal{P}| = 14$ decision procedures.

4.2. Results

The cactus plot of the evaluation is presented in Figure 2. A cactus plot is a visualization of a solver's performance on a instance suite. The horizontal axis represents the number of instances solved (higher is better) and the vertical axis is the instance-wise PAR-2 (lower is better). In Figure 2, we observe that Goose outperforms the competition winning solver α-β CROWN.

Table 1 presents the PAR-2 across all instances for all solvers. The PAR-2 score of a solver on a instance is the wallclock runtime if successful, otherwise twice the wallclock runtime (lower is better). We observe Goose to improve on the competition winning solver α-β CROWN by 47.3% in PAR-2 score.

VNN-COMP '21 determined competition winners via a custom scoring scheme. We do not consider this. However, the standings of the competition are highly correlated.

4.3. Analysis

One possible limitation of any ML-driven approach is that it is difficult to generalize to novel problem occurrences and not overfit to ones it has been trained on. While this evaluation is limited to 800 instances from VNN-COMP, the training dataset was produced from a completely independent class of problems [37]. As such, our approach does not seem to suffer from the aforementioned issues.

The incomplete solvers were notably particularly effective, in particular on the first round of the incremental deepening. PGD was extremely effective on SAT instances. Complete verifiers have a strong preference for Gurobi, while Gurobi did outperform SCIP in general, SCIP was observed faster on several instances, with up to a 7x improvement over Gurobi.

5. Related Work

Perhaps the biggest inspirations and insights of this work were from DNNV [28]. In particular, Theorem 1 for the canonical form. This further enabled the probabilistic satisfiability inference of Goose.

Algorithm selection tools have a rich history and have been around since at least 1976 when Rice et al. first proposed it [16]. Algorithm selectors have been extensively used in many contexts, e.g., classifiers for machine learning [40], combinatorics [41], and other NP-hard optimization problems [42, 43].

Within the context of logic solvers, algorithm selectors have been proposed for SAT [44, 45, 46] and Quantified Boolean Formulas [47, 48]. Scott et al. proposed the MachSMT algorithm selection tool for SMT Solvers [19]. A system for dynamic algorithm selection over SMT solvers was also recently proposed [18].

6. Conclusions

In this paper, we presented Goose, a solver for deep neural network verification. Goose has a meta-solver architecture (Figure 1) and supports a wide variety of decision procedures and semi-decision procedure. Goose leverages three key meta-solving techniques to further improve efficiency, namely, algorithm selection, probabilistic satisfiability inference, and time interval deepening. Using Goose we observe a 47.3% improvement across benchmarks and solvers from VNN-COMP '21.

Future Work. Logic solvers for SAT, SMT, and MILP, among others, often implement several configurations of a base decision procedure, as there is no known universally optimal procedure for such hard search and optimization problems. This phenomenon has been observed across numerous communities [19, 49, 20], and this paper suggests something analogous for VNN-LIB solvers. Despite this success, several modern logic solvers architectures do not take this into account. This poses challenges when incorporating meta-solving with existing solvers. With the empirical success of Goose, we believe meta-solving should be seen as a "first-class citizen" engineering (standard, not meta) solvers in the future.

Acknowledgments

Thanks to Piyush Jha for feedback and improvements on Figure 1.

References

[1] L. Kaiser, A. N. Gomez, N. Shazeer, A. Vaswani, N. Parmar, L. Jones, J. Uszkoreit, One model to learn them all, CoRR abs/1706.05137 (2017). URL: http://arxiv.org/abs/1706.05137. arXiv:1706.05137.

[2] G. Manfredi, Y. Jestin, An introduction to acas xu and the challenges ahead, in: 2016 IEEE/AIAA 35th Digital Avionics Systems Conference (DASC), IEEE, 2016, pp. 1–9.

[3] D. Song, K. Eykholt, I. Evtimov, E. Fernandes, B. Li, A. Rahmati, F. Tramèr, A. Prakash, T. Kohno, Physical adversarial examples for object detectors, in: C. Rossow, Y. Younan (Eds.), 12th USENIX Workshop on Offensive Technologies, WOOT 2018, Baltimore, MD, USA, August 13-14, 2018, USENIX Association, 2018. URL: https://www.usenix.org/conference/woot18/presentation/eykholt.

[4] G. Katz, C. W. Barrett, D. L. Dill, K. Julian, M. J. Kochenderfer, Reluplex: An efficient SMT solver for verifying deep neural networks, in: R. Majumdar, V. Kuncak (Eds.), Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I, volume 10426 of *Lecture Notes in Computer Science*, Springer, 2017, pp. 97–117. URL: https://doi.org/10.1007/978-3-319-63387-9_5. doi:10.1007/978-3-319-63387-9_5.

[5] H. Salman, G. Yang, H. Zhang, C.-J. Hsieh, P. Zhang, A convex relaxation barrier to tight robustness verification of neural networks, Advances in Neural Information Processing Systems 32 (2019) 9835–9846.

[6] G. Singh, J. Maurer, C. Müller, M. Mirman, T. Gehr, A. Hoffmann, P. Tsankov, D. D. Cohen, M. Püschel, M. Vechev, Eth robustness analyzer for neural networks (eran), 2020, URL https://github.com/eth-sri/eran (????).

[7] V. Tjeng, K. Y. Xiao, R. Tedrake, Evaluating robustness of neural networks with mixed integer programming, in: 7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019, OpenReview.net, 2019. URL: https://openreview.net/forum?id=HyGIdiRqtm.

[8] S. Wang, H. Zhang, K. Xu, X. Lin, S. Jana, C.-J. Hsieh, J. Z. Kolter, Beta-CROWN: Efficient bound propagation with per-neuron split constraints for complete and incomplete neural network verification, Advances in Neural Information Processing Systems 34 (2021).

[9] K. Xu, Z. Shi, H. Zhang, Y. Wang, K.-W. Chang, M. Huang, B. Kailkhura, X. Lin, C.-J. Hsieh, Automatic perturbation analysis for scalable certified robustness and beyond, Advances in Neural Information Processing Systems 33 (2020).

[10] K. Xu, H. Zhang, S. Wang, Y. Wang, S. Jana, X. Lin, C.-J. Hsieh, Fast and Complete: Enabling complete neural network verification with rapid and massively parallel incomplete verifiers, in: International Conference on Learning Representations, 2021. URL: https://openreview.net/forum?id=nVZtXBI6LNn.

[11] H. Zhang, T.-W. Weng, P.-Y. Chen, C.-J. Hsieh, L. Daniel, Efficient neural network robustness certification with general activation functions, Advances in Neural Information Processing Systems 31 (2018) 4939–4948. URL: https://arxiv.org/pdf/1811.00866.pdf.

[12] S. Bak, nnenum: Verification of relu neural networks with optimized abstraction refinement, in: A. Dutle, M. M. Moscato, L. Titolo, C. A. Muñoz, I. Perez (Eds.), NASA Formal Methods - 13th International Symposium, NFM 2021, Virtual Event, May 24-28, 2021, Proceedings, volume 12673 of *Lecture Notes in Computer Science*, Springer, 2021, pp. 19–36. URL: https://doi.org/10.1007/978-3-030-76384-8_2. doi:10.1007/978-3-030-76384-8_2.

[13] P. Henriksen, A. R. Lomuscio, Efficient neural network verification via adaptive refinement and adversarial search, in: G. D. Giacomo, A. Catalá, B. Dilkina, M. Milano, S. Barro, A. Bugarín, J. Lang (Eds.), ECAI 2020 - 24th European Conference on Artificial Intelligence, 29 August-8 September 2020, Santiago de Compostela, Spain, August 29 - September 8, 2020 - Including 10th Conference on Prestigious Applications of Artificial Intelligence (PAIS

2020), volume 325 of *Frontiers in Artificial Intelligence and Applications*, IOS Press, 2020, pp. 2513–2520. URL: https://doi.org/10.3233/FAIA200385. doi:10.3233/FAIA200385.

[14] Y. Chen, M. J. Wainwright, Fast low-rank estimation by projected gradient descent: General statistical and algorithmic guarantees, CoRR abs/1509.03025 (2015). URL: http://arxiv.org/abs/1509.03025. arXiv:1509.03025.

[15] H. Gupta, K. H. Jin, H. Q. Nguyen, M. T. McCann, M. Unser, Cnn-based projected gradient descent for consistent CT image reconstruction, IEEE Trans. Medical Imaging 37 (2018) 1440–1453. URL: https://doi.org/10.1109/TMI.2018.2832656. doi:10.1109/TMI.2018.2832656.

[16] J. R. Rice, The algorithm selection problem, Adv. Comput. 15 (1976) 65–118. URL: https://doi.org/10.1016/S0065-2458(08)60520-3. doi:10.1016/S0065-2458(08)60520-3.

[17] H. Barbosa, C. Barrett, M. Brain, G. Kremer, H. Lachnitt, M. Mann, A. Mohamed, M. Mohamed, A. Niemetz, A. Nötzli, A. Ozdemir, M. Preiner, A. Reynolds, Y. Sheng, C. Tinelli, Y. Zohar, cvc5: A versatile and industrial-strength SMT solver, in: D. Fisman, G. Rosu (Eds.), Tools and Algorithms for the Construction and Analysis of Systems - 28th International Conference, TACAS 2022, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Munich, Germany, April 2-7, 2022, Proceedings, to appear, Lecture Notes in Computer Science, 2022.

[18] N. Pimpalkhare, F. Mora, E. Polgreen, S. A. Seshia, Medleysolver: Online SMT algorithm selection, in: C. Li, F. Manyà (Eds.), Theory and Applications of Satisfiability Testing - SAT 2021 - 24th International Conference, Barcelona, Spain, July 5-9, 2021, Proceedings, volume 12831 of *Lecture Notes in Computer Science*, Springer, 2021, pp. 453–470. URL: https://doi.org/10.1007/978-3-030-80223-3_31. doi:10.1007/978-3-030-80223-3_31.

[19] J. Scott, A. Niemetz, M. Preiner, S. Nejati, V. Ganesh, Machsmt: A machine learning-based algorithm selector for SMT solvers, in: J. F. Groote, K. G. Larsen (Eds.), Tools and Algorithms for the Construction and Analysis of Systems - 27th International Conference, TACAS 2021, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2021, Luxembourg City, Luxembourg, March 27 - April 1, 2021, Proceedings, Part II, volume 12652 of *Lecture Notes in Computer Science*, Springer, 2021, pp. 303–325. URL: https://doi.org/10.1007/978-3-030-72013-1_16. doi:10.1007/978-3-030-72013-1_16.

[20] L. Xu, F. Hutter, H. H. Hoos, K. Leyton-Brown, Satzilla: Portfolio-based algorithm selection for SAT, CoRR abs/1111.2249 (2011). URL: http://arxiv.org/abs/1111.2249. arXiv:1111.2249.

[21] S. Bak, C. Liu, T. T. Johnson, The second international verification of neural networks competition (VNN-COMP 2021): Summary and results, CoRR abs/2109.00498 (2021). URL: https://arxiv.org/abs/2109.00498. arXiv:2109.00498.

[22] G. Katz, D. A. Huang, D. Ibeling, K. Julian, C. Lazarus, R. Lim, P. Shah, S. Thakoor, H. Wu, A. Zeljic, D. L. Dill, M. J. Kochenderfer, C. W. Barrett, The marabou framework for verification and analysis of deep neural networks, in: I. Dillig, S. Tasiran (Eds.), Computer Aided Verification - 31st International Conference, CAV 2019, New York City, NY, USA, July 15-18, 2019, Proceedings, Part I, volume 11561 of *Lecture Notes in Computer Science*, Springer, 2019, pp. 443–452. URL: https://doi.org/10.1007/978-3-030-25540-4_26. doi:10.1007/978-3-030-25540-4_26.

[23] O. R. developers, Onnx runtime, https://onnxruntime.ai/, 2021. Version: x.y.z.

[24] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. J. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Józefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. G. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. A. Tucker, V. Vanhoucke, V. Vasudevan, F. B. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, X. Zheng, Tensorflow: Large-scale machine learning on heterogeneous distributed systems, CoRR abs/1603.04467 (2016). URL: http://arxiv.org/abs/1603.04467. arXiv:1603.04467.

[25] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Z. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, S. Chintala, Pytorch: An imperative style, high-performance deep learning library, in: H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. B. Fox, R. Garnett (Eds.), Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada, 2019, pp. 8024–8035. URL: https://proceedings.neurips.cc/paper/2019/hash/bdbca288fee7f92f2bfa9f7012727740-Abstract.html.

[26] F. Chollet, et al., Keras, https://keras.io, 2015.

[27] The Verification of Neural Networks Library (VNN-LIB), www.vnnlib.org, 2019.

[28] D. Shriver, S. G. Elbaum, M. B. Dwyer, DNNV: A framework for deep neural network verification, in: A. Silva, K. R. M. Leino (Eds.), Computer Aided Verification - 33rd International Conference, CAV 2021, Virtual Event, July 20-23, 2021, Proceedings, Part I, volume 12759 of *Lecture Notes in Computer Science*, Springer, 2021, pp. 137–150. URL: https://doi.org/10.1007/978-3-030-81685-8_6. doi:10.1007/978-3-030-81685-8_6.

[29] A. Reinefeld, T. A. Marsland, Enhanced iterative-deepening search, IEEE Trans. Pattern Anal. Mach. Intell. 16 (1994) 701–710. URL: https://doi.org/10.1109/34.297950. doi:10.1109/34.297950.

[30] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. VanderPlas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay, Scikit-learn: Machine learning in python, J. Mach. Learn. Res. 12 (2011) 2825–2830. URL: http://dl.acm.org/citation.cfm?id=2078195.

[31] T. Chen, C. Guestrin, Xgboost: A scalable tree boosting system, in: B. Krishnapuram, M. Shah, A. J. Smola, C. C. Aggarwal, D. Shen, R. Rastogi (Eds.), Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016, ACM, 2016, pp. 785–794. URL: https://doi.org/10.1145/2939672.2939785. doi:10.1145/2939672.2939785.

[32] S. Gowal, K. Dvijotham, R. Stanforth, R. Bunel, C. Qin, J. Uesato, R. Arandjelovic, T. Mann, P. Kohli, On the effectiveness of interval bound propagation for training verifiably robust models, arXiv preprint arXiv:1810.12715 (2018).

[33] T. Hickey, Q. Ju, M. H. Van Emden, Interval arithmetic: From principles to implementation, Journal of the ACM (JACM) 48 (2001) 1038–1068.

[34] K. Bestuzheva, M. Besançon, W.-K. Chen, A. Chmiela, T. Donkiewicz, J. van Doornmalen, L. Eifler, O. Gaul, G. Gamrath, A. Gleixner, L. Gottwald, C. Graczyk, K. Halbig, A. Hoen, C. Hojny, R. van der Hulst, T. Koch, M. Lübbecke, S. J. Maher, F. Matter, E. Mühmer, B. Müller, M. E. Pfetsch, D. Rehfeldt, S. Schlein, F. Schlösser, F. Serrano, Y. Shinano,

B. Sofranac, M. Turner, S. Vigerske, F. Wegscheider, P. Wellner, D. Weninger, J. Witzig, The SCIP Optimization Suite 8.0, ZIB-Report 21-41, Zuse Institute Berlin, 2021. URL: http://nbn-resolving.de/urn:nbn:de:0297-zib-85309.

[35] Gurobi Optimization, LLC, Gurobi Optimizer Reference Manual, 2021. URL: https://www.gurobi.com.

[36] L. M. de Moura, N. Bjørner, Z3: an efficient SMT solver, in: C. R. Ramakrishnan, J. Rehof (Eds.), Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings, volume 4963 of *Lecture Notes in Computer Science*, Springer, 2008, pp. 337–340. URL: https://doi.org/10.1007/978-3-540-78800-3_24. doi:10.1007/978-3-540-78800-3_24.

[37] J. Scott, G. Pan, E. B. Khalil, V. Ganesh, Pierce: A testing infrastructure for machine learning verification tools, URL https://guantingpan.github.io/pierce/ (????).

[38] S. Baldwin, Compute Canada: Advancing Computational Research, in: Journal of Physics: Conference Series, volume 341, IOP Publishing, 2012, p. 012001.

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

[40] S. Ali, K. A. Smith, On learning algorithm selection for classification, Appl. Soft Comput. 6 (2006) 119–138. URL: https://doi.org/10.1016/j.asoc.2004.12.002. doi:10.1016/j.asoc.2004.12.002.

[41] L. Kotthoff, Algorithm selection for combinatorial search problems: A survey, in: C. Bessiere, L. D. Raedt, L. Kotthoff, S. Nijssen, B. O'Sullivan, D. Pedreschi (Eds.), Data Mining and Constraint Programming - Foundations of a Cross-Disciplinary Approach, volume 10101 of *Lecture Notes in Computer Science*, Springer, 2016, pp. 149–190. URL: https://doi.org/10.1007/978-3-319-50137-6_7. doi:10.1007/978-3-319-50137-6_7.

[42] K. Tierney, Y. Malitsky, An algorithm selection benchmark of the container pre-marshalling problem, in: C. Dhaenens, L. Jourdan, M. Marmion (Eds.), Learning and Intelligent Optimization - 9th International Conference, LION 9, Lille, France, January 12-15, 2015. Revised Selected Papers, volume 8994 of *Lecture Notes in Computer Science*, Springer, 2015, pp. 17–22. doi:10.1007/978-3-319-19084-6_2.

[43] M. Vallati, L. Chrpa, D. E. Kitchin, Portfolio-based planning: State of the art, common practice and open challenges, AI Commun. 28 (2015) 717–733. URL: https://doi.org/10.3233/AIC-150671. doi:10.3233/AIC-150671.

[44] L. Xu, F. Hutter, H. H. Hoos, K. Leyton-Brown, : The design and analysis of an algorithm portfolio for SAT, in: C. Bessiere (Ed.), Principles and Practice of Constraint Programming - CP 2007, 13th International Conference, CP 2007, Providence, RI, USA, September 23-27, 2007, Proceedings, volume 4741 of *Lecture Notes in Computer Science*, Springer, 2007, pp. 712–727. URL: https://doi.org/10.1007/978-3-540-74970-7_50. doi:10.1007/978-3-540-74970-7_50.

[45] L. Xu, F. Hutter, H. H. Hoos, K. Leyton-Brown, Satzilla: Portfolio-based algorithm selection for SAT, J. Artif. Intell. Res. 32 (2008) 565–606. URL: https://doi.org/10.1613/jair.2490. doi:10.1613/jair.2490.

[46] L. Xu, F. Hutter, H. H. Hoos, K. Leyton-Brown, Satzilla2009: an automatic algorithm

portfolio for sat, SAT 4 (2009) 53–55.

[47] Y. Malitsky, Evolving instance-specific algorithm configuration, in: Instance-Specific Algorithm Configuration, Springer, 2014, pp. 93–105.

[48] L. Pulina, A. Tacchella, A multi-engine solver for quantified boolean formulas, in: C. Bessiere (Ed.), Principles and Practice of Constraint Programming - CP 2007, 13th International Conference, CP 2007, Providence, RI, USA, September 23-27, 2007, Proceedings, volume 4741 of *Lecture Notes in Computer Science*, Springer, 2007, pp. 574–589. doi:10.1007/978-3-540-74970-7_41.

[49] A. Shukla, A. Biere, L. Pulina, M. Seidl, A survey on applications of quantified boolean formulas, in: 31st IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2019, Portland, OR, USA, November 4-6, 2019, IEEE, 2019, pp. 78–84. URL: https://doi.org/10.1109/ICTAI.2019.00020. doi:10.1109/ICTAI.2019.00020.