LiVe'23

These are informal proceedings of the *7th Workshop on Learning in Verification*, held as a satellite event of ETAPS in Paris, on April 22, 2023. The details of the event can be found at `https://www7.in.tum.de/~kretinsk/LiVe2023.html`

Towards a Machine Learning Approach for the Detection of Null Pointer Errors

Charlie Molony and Vasileios Koutavas [*]

Trinity College Dublin, Dublin, Ireland
molonych@tcd.ie   Vasileios.Koutavas@tcd.ie

Abstract. We present preliminary results of our exploration of Machine Learning techniques for detecting program errors, and in particular null pointer errors in C code. We have curated a data set of positive and negative samples of such errors from open source repositories and used it to train a number of Machine Learning algorithms. Our current results show that this is a promising approach for detecting null pointer errors.

1 Introduction

In a 2009 talk at QCon, Sir Tony Hoare has referred to his invention of the null reference as his "billion dollar mistake" due to the innumerable errors it has caused since its inception in 1965. Several techniques have been developed to detect null pointer errors, such as testing (e.g. [7]), formal verification (e.g. [2]), code analysis (e.g. [1]), and runtime verification (e.g. [5]), among others. Nevertheless, none of these approaches has provided a complete remedy for null pointer errors in programming languages such as C.

In this ongoing work we explore the use of Machine Learning (ML) techniques in detecting null pointer errors, which may provide another tool in the arsenal of software developers for combating this type of errors. Our hypothesis is that, in many cases, programs with null pointer errors contain enough textual information in their code that would allow a machine learning algorithm to learn how to recognise them. In this work we set out to investigate whether this hypothesis holds in practice. Although ML techniques have been used for aiding in program verification (e.g. [8]) and in *repairing* program errors, including null pointer errors [6], we are not aware of previous work that aims to use ML-only techniques for detecting null pointer errors.

2 Methodology

In our investigation we have used the GitHub API to inspect more that one million commit points from open source C repositories to create a training data set of commits which aim to correct specific null pointer exceptions. This data set currently contains 1400 unique snippets of code with positive examples (containing null pointer exceptions) and the same number of negative examples

[*] This publication has emanated from research supported in part by a grant from Science Foundation Ireland under Grant number 13/RC/2094_2.

(the corresponding fixed code). We also created a testing data set of 300 positive and 300 negative examples. The latter data set is drawn from open source repositories which are different than those used for learning to avoid bias in our experiments due to project-specific programming conventions. In both data sets, only the C functions that were altered at each commit point were kept, with surrounding code being dropped. Typically these commit points altered a single functions but a small number of commits involved changes to more than one functions.

We have used the training set to train four ML algorithms: Support Vector Machines (SVM) [4], XGBoost [3], a neural network with a binary cross-entropy loss function, and a hybrid system with a Multi Layer Perceptron (MLP) combined with XGBoost.

3 Current Results

Our preliminary results indicate that Machine Learning techniques can indeed be used to recognise null pointer errors with a good degree of success, however some knowledge about the semantics of a programming language improve the approach. Our best results to date are derived by the hybrid MLP+XGBoost system achieving 79.08% accuracy ($\frac{\text{true positives}+\text{true negatives}}{\text{all examples}}$), 77.30% precision ($\frac{\text{true positives}}{\text{true positives}+\text{false positives}}$), 82.35% recall ($\frac{\text{true positives}}{\text{true positives}+\text{false negatives}}$), 79.75% F1 score ($2\frac{\text{recall}\times\text{precision}}{\text{recall}+\text{precision}}$), and 79.08% AUC score. We currently continue our investigation with exploring other ML algorithms, comparing our results to traditional techniques such as formal verification, and improving our learning and testing data sets.

Our data set leverages the semantic notion of a C function to focus the attention of the algorithms around the code snippet where a null pointer exception occurs. When we use entire C files for training and testing data sets, ML algorithms are only marginally more performant than a random binary classifier. This is a strong indication that leveraging the semantics of the underlying programming language can focus the attention of the algorithms to relevant parts of the program and improve the effectiveness of the approach. Exploring further semantics properties to improve the training data set could lead to better results.

As is with most ML models, our approach makes hard to explain to the programmer *why* a function may have a null pointer error. We envisage that incorporating semantics techniques (e.g. a flow analysis) in the construction of the training data or in the validation of the output of the ML algorithm could mitigate this.

4 Future Work

In the future we hope to evaluate ML algorithms against static analysis and formal verification tools. We also believe that there is scope of improvement of the learning data set. Our data set may also be improved in accuracy but also explainability if we leverage further programming language semantics techniques. Finally the ML approach to bug finding will need to be evaluated in practice but also in detecting a wider range of errors.

References

1. Ayewah, N., Pugh, W., Hovemeyer, D., Morgenthaler, J.D., Penix, J.: Using static analysis to find bugs. IEEE Software **25**(5), 22–29 (2008)
2. Calcagno, C., Distefano, D.: Infer: An automatic program verifier for memory safety of c programs. In: Bobaru, M., Havelund, K., Holzmann, G.J., Joshi, R. (eds.) NASA Formal Methods. pp. 459–465. Springer Berlin Heidelberg, Berlin, Heidelberg (2011)
3. Chen, T., Guestrin, C.: Xgboost: A scalable tree boosting system. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. p. 785–794. KDD '16, Association for Computing Machinery, New York, NY, USA (2016)
4. Hearst, M.A., Dumais, S.T., Osuna, E., Platt, J., Scholkopf, B.: Support vector machines. IEEE Intelligent Systems and their applications **13**(4), 18–28 (1998)
5. Kendall, S.C.: Bcc: Runtime checking for c programs. In: Proceedings of the USENIX Summer Conference. pp. 5–16 (1983)
6. Lee, J., Hong, S., Oh, H.: Npex: Repairing java null pointer exceptions without tests. In: Proceedings of the 44th International Conference on Software Engineering. p. 1532–1544. ICSE '22, Association for Computing Machinery, New York, NY, USA (2022)
7. Padhye, R., Lemieux, C., Sen, K., Papadakis, M., Le Traon, Y.: Semantic fuzzing with zest. In: Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis. p. 329–340. ISSTA 2019, Association for Computing Machinery, New York, NY, USA (2019)
8. Si, X., Naik, A., Dai, H., Naik, M., Song, L.: Code2inv: A deep learning framework for program verification. In: Lahiri, S.K., Wang, C. (eds.) Computer Aided Verification. pp. 151–164. Springer International Publishing, Cham (2020)

Formal verification of loss-functions: quality assessment in image super-resolution

Andoni Rodríguez-Barcenilla[1,2], César Sánchez[2], and Felipe Gorostiaga[2]

[1] Universidad Politécnica de Madrid. Spain
[2] IMDEA Software Institute, Madrid. Spain

Abstract. In this work in progress, we address the problem of the verification of properties of loss functions used in training (and, analogously, metrics used in testing). These correctness proofs are particularly interesting for critical tasks, from a developer's perspective; and also from a performance perspective when using large-scale neural networks, since it allows to get rid of a high number of run-time checking, which presumably enhances execution times.
We introduce an SMT-based approach to certify properties of loss functions used for microscopical images' super-resolution. Our early results are promising and we report on a prototype in Dafny-Z3. To the best of our knowledge, this is the first idea to perform such tasks.

1 Problem statement

So far, verification of machine-learning (ML) models has been approached utilizing many different paradigms; however, checking loss functions has not been addressed, since it has been considered that we can see in training (controlled) environments whether results are correct or not. Thus, the usual way to (informally) verify whether an implemented function respects a set of properties is to check them run-time. However, this yields two problems: (1) it is more probable that the developer implements functions that do not conform the properties; and (2) this run-time checking is resource-consuming, especially in large-scale networks where this checking is performed millions of times. To illustrate the first problematic, let us consider the following simple example:

Example 11 (An incorrect implementation.) *Let a function $f(x) = x^2$, which holds the following property: $\forall x \in \mathbb{R}.\ f(x) \geq 0$.*

Now, consider (1) the developer makes an inadvertent mistake and implements $f(x) = x^3$, which does not satisfy the property (consider negative x); and also consider (2) that he has a monitor to check that $f(x) \geq 0$ holds. Let a biased or corrupted dataset $\mathcal{D} = \{5, 6, 3, 10, ...\}$ that does not contain negative $x \in \mathcal{D}$; thus, the monitor will never complain. However, the function is incorrectly implemented.

This error would have been identified by the static verifier. Thus, in order to get closer to (pseudo) correctness-by-construction, the more precise properties *we add (and prove), the more sure we will be that our implementation conforms the specification.*

This formal-verification-powered practise becomes more important the harder are functions to implement or if datasets are suspected biased. And, of course, this pseudo-proofs are particularly interesting for critical tasks, e.g., aircraft vision systems or radiomedical images.

As for the second problematic, it is easy to see that verifiying loss functions (instead of using run-time checking) yields a performance improvement: it is better to have a verified function that will not call a monitor a million times per training and testing. Thus, the problem that urgently needs to be solved is this unnecessary extra resource consumption. This and the first problematic are both solved formally verifying loss functions. We are currently developing the idea applying it to the *super-resolution* (SR) problem.

SR is a technique that allows to obtain a high-resolution image from a low-resolution one. In ML-based SR loss functions and metrics go from Mean Squared Error (MSE) to more task specific ones. In this work in progress, we address the problem of formal-verification of loss functions used in microscopical-image SR.

We introduce an approach by which we can rely on *Sastisfiability Modulo Theories* (SMT) solvers to provably verify properties of loss functions. In particular, we already offer results on the verification of models and algorithms that make use of the most popular image-quality-metrics. These verifications already suggest a discussion, since many calculi are made using trascendental functions, which calls into question usability of (classic) SMTs for these problems.

In summary, the contributions of this work in progress are as follows: (1) It raises the idea of a novel formal verification for loss functions and metric using SMT solvers, in Dafny-Z3. (2) Formalizes and implements two classic image quality metrics for the SR problem: PSNR/MSE and Structural Similarity Index (SSIM). (3) Limitations of this method are discussed: they are mainly focused on trascendental functions that limit reliability of SMT solvers.

Example 12 (Why should we verify properties? SSIM as a use case.) *When dealing with SSIM as a loss function, we need to prove that the function has a minimum, otherwise it cannot be expected to converge. Other properties like boundedness are not strictly required but help avoiding exploding loss scores as well as improving convergence time. Symmetry, for instance, yields better convergence on corrupted labels.*

If we are talking, instead, of SSIM as a metric (i.e., a similarity score), then some other properties become compulsory. For instance, unique maximum; i.e., we want the function to return the maximum score only when comparing an image to itself. Symmetry is also a necessary requirement from a safety perspective, since comparing x to y should lead to the same score as comparing y to x.

We report on prototype in Dafny, a language that supports formal specification and which uses the Z3 automated theorem prover below. To the best of our knowledge, this is the first approach combining loss functions verification and SMT solving. We also suggests a set of future research lines.

Guessing Winning Policies in LTL Synthesis by Semantic Learning

Jan Křetínský$^{[0000-0002-8122-2881]}$, Max Prokop, Sabine Rieder, and Tobias Meggendorfer$^{[0000-0002-1712-2165]}$

Technical University of Munich

This work has been submitted to CAV 2023.

LTL synthesis [10] is one of the most prominent frameworks for automatic construction of reactive systems, due to how established linear temporal logic (LTL) [9] is in the area of safety-critical and provably reliable dynamic systems. A classical solution technique is the *automata-theoretic approach* [1, 11] which first translates the specification into an automaton. Together with a partition of the atomic propositions, the automaton induces a *parity game*, whose solution corresponds to solving the original problem. However, there is an important computational caveat: the problem of LTL synthesis is 2-EXPTIME complete. Despite the infeasibility in the worst-case, many efforts have been made to cope with practical problems by better LTL-to-ω-automata translations [2, 3] or by a variety of heuristics whose impact is documented in the synthesis competition SYNTCOMP [4]. A recent advancement by [5] is the exploitation of a semantic labeling of states, provided by the automata translation, in order to solve the parity game more efficiently or guide the exploration of on-the-fly approaches [7] as implemented in STRIX [8]. In this work, we extend the work of [5] by using the semantic labeling to learn more complex heuristics (backed by support vector machines) in order to solve the parity games that arise from LTL synthesis problems.

Naively, our approach works roughly as follows: We first solve numerous existing games by classical means to figure out the winning choices. Using this data and features derived from the semantic labeling, we train a machine learning model to predict said winning choices. Ideally, these predictions immediately result in a winning strategy, also for unseen games. However, even if the prediction is not entirely correct, it can still serve as an initial strategy for strategy iteration: SI gradually improves the current strategy and, if the initial strategy is only a few updates away from a winning one, SI terminates much faster.

To realize our approach, we overcome several interesting challenges. Obtaining a canonical ground truth, for example, proves to be a much tougher task than one might expect. As an edge might be selected in some but not in other winning strategies, we cannot simply declare *some* solution provided by strategy iteration as ground truth. Rather, we have to incorporate the information of *all* winning strategies, which we achieve by using approaches of classical game theory mixed with simulation based approaches to obtain practical feasibility.

In addition to the ground truth, we put significant effort into the design of over 200 features that try to capture the essence of the semantic labeling in various different ways. While we also have simple, syntactic features like the height of some LTL formulas syntax tree, we employ several semantic features like *trueness* as presented by [5] or features based on obligation sets [6].

Our heuristic can be used in multiple ways: As mentioned above, it can help solving parity games by predicting strategies that are either immediately winning or only need a few adjustments by strategy iteration algorithms to be made winning. In our experiments we observed the former surprisingly often, even for complex practical formulae without even reading them. Thus, in some sense our heuristic can be viewed as a constant time and space solution to LTL synthesis, as it can be run on-the-fly with no input-dependent precomputation. Further, our heuristic can guide the exploration of parity games in on-the-fly synthesis tools such as STRIX, by suggesting which successors are most likely to win and thus should be explored first. This can cause STRIX to find a solution with only a small part of the automaton constructed, which, due to the double-exponential nature of the latter, can have drastic impact on the overall runtime.

References

1. Büchi, J.: On a decision method in restricted second-order arithmetic. In: Nagel, E., Suppes, P., Tarski, A. (eds.) Proceedings of the First International Congress on Logic, Methodology, and Philosophy of Science 1960 (1962)
2. Esparza, J., Kretínský, J.: From LTL to deterministic automata: A safraless compositional approach. In: Biere, A., Bloem, R. (eds.) Computer Aided Verification - 26th International Conference, CAV 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 18-22, 2014. Proceedings. Lecture Notes in Computer Science, vol. 8559, pp. 192–208. Springer (2014). https://doi.org/10.1007/978-3-319-08867-9“˙13
3. Esparza, J., Kretínský, J., Sickert, S.: A unified translation of linear temporal logic to ω-automata. J. ACM **67**(6), 33:1–33:61 (2020). https://doi.org/10.1145/3417995
4. Jacobs, S., Pérez, G.A., Abraham, R., Bruyère, V., Cadilhac, M., Colange, M., Delfosse, C., van Dijk, T., Duret-Lutz, A., Faymonville, P., Finkbeiner, B., Khalimov, A., Klein, F., Luttenberger, M., Meyer, K.J., Michaud, T., Pommellet, A., Renkin, F., Schlehuber-Caissier, P., Sakr, M., Sickert, S., Staquet, G., Tamines, C., Tentrup, L., Walker, A.: The reactive synthesis competition (SYNTCOMP): 2018-2021. CoRR **abs/2206.00251** (2022). https://doi.org/10.48550/arXiv.2206.00251
5. Kretínský, J., Manta, A., Meggendorfer, T.: Semantic labelling and learning for parity game solving in LTL synthesis. In: Chen, Y., Cheng, C., Esparza, J. (eds.) Automated Technology for Verification and Analysis - 17th International Symposium, ATVA 2019, Taipei, Taiwan, October 28-31, 2019, Proceedings. Lecture Notes in Computer Science, vol. 11781, pp. 404–422. Springer (2019). https://doi.org/10.1007/978-3-030-31784-3“˙24
6. Li, J., Zhang, L., Pu, G., Vardi, M.Y., He, J.: LTL satisfiability checking revisited. In: Sánchez, C., Venable, K.B., Zimányi, E. (eds.) 2013 20th International Symposium on Temporal Representation and Reasoning, Pensacola, FL, USA, September 26-28, 2013. pp. 91–98. IEEE Computer Society (2013). https://doi.org/10.1109/TIME.2013.19
7. Luttenberger, M., Meyer, P.J., Sickert, S.: Practical synthesis of reactive systems from LTL specifications via parity games. Acta Informatica **57**(1-2), 3–36 (2020). https://doi.org/10.1007/s00236-019-00349-3

8. Meyer, P.J., Sickert, S., Luttenberger, M.: Strix: Explicit reactive synthesis strikes back! In: Chockler, H., Weissenbacher, G. (eds.) Computer Aided Verification - 30th International Conference, CAV 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings, Part I. Lecture Notes in Computer Science, vol. 10981, pp. 578–586. Springer (2018). https://doi.org/10.1007/978-3-319-96145-3“˙31
9. Pnueli, A.: The temporal logic of programs. In: 18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977. pp. 46–57. IEEE Computer Society (1977). https://doi.org/10.1109/SFCS.1977.32
10. Pnueli, A., Rosner, R.: On the synthesis of an asynchronous reactive module. In: Ausiello, G., Dezani-Ciancaglini, M., Rocca, S.R.D. (eds.) Automata, Languages and Programming, 16th International Colloquium, ICALP89, Stresa, Italy, July 11-15, 1989, Proceedings. Lecture Notes in Computer Science, vol. 372, pp. 652–671. Springer (1989). https://doi.org/10.1007/BFb0035790
11. Vardi, M.Y., Wolper, P.: An automata-theoretic approach to automatic program verification (preliminary report). In: Proceedings of the Symposium on Logic in Computer Science (LICS '86), Cambridge, Massachusetts, USA, June 16-18, 1986. pp. 332–344. IEEE Computer Society (1986)

Translating Neural Networks Into IO-Equivalent Finite Automata

Eric Alsmann[1], Marco Sälzer[1], Florian Bruse[1], Rüdiger Ehlers[2], and Martin Lange[1]

[1] School of Electr. Eng. and Computer Science, University of Kassel, Germany
{eric.alsmann,marco.saelzer,florian.bruse,martin.lange}@uni-kassel.de
[2] Clausthal University of Technology, Germany
ruediger.ehlers@tu-clausthal.de

Introduction

Deep Neural Networks (DNN), trained using task-oriented and precisely crafted techniques, are the driving force of all modern deep learning applications that have produced astonishing results. Their striking performance comes with a downside: they are a blackbox. While it is easy to describe structure and parameters of a DNN, it is hard to obtain reliable predictions for or explanation of their behaviour. Certifying that some DNN satisfies specific safety properties, formally called *verifying* these properties, is difficult. A common, corresponding decision problem, formally called *output reachability*, is NP-complete [2], even for completely shallow DNN and simple specifications of relevant inputs and outputs [4]. A typical interpretation task for some DNN N and an input-output pair $(\overline{x}, N(\overline{x}))$ is to answer the question "which features of $\overline{x}$ are the relevant ones leading to the output $N(\overline{x})$?" A corresponding decision problem, called the MinimumSufficientReason problem, is known to be Σ_2^P-complete [1].

We propose a new, finite-state automata based approach for tackling challenges arising from the blackbox nature of DNN. A DNN N computes a function of type $\mathbb{R}^m \to \mathbb{R}^n$ for some $m, n \in \mathbb{N}$, which induces a relation $R_N \subseteq \mathbb{R}^m \times \mathbb{R}^n$. Using an appropriate encoding, R_N can be represented by a set of infinite words over an alphabet of $(m+n)$-*track symbols* of the form $(a_1, \dots, a_m, b_1, \dots, b_n)$ where a_i, b_i are taken from an alphabet like $\{0, 1, ., +, -\}$. A finite-state automaton $\mathcal{A}$ over such $(m+n)$-*track words* works similar to one over plain, one-dimensional words, but can be seen as a (synchronised) transducer between input symbols $(a_1, \dots, a_m)$ and output symbols $(b_1, \dots, b_n)$.

Translating DNN into WNBA

We present a complete construction of a weak Büchi automaton of exponential size that recognises the input-output behaviour of a given DNN.

Theorem 1. *[3] Let N be a DNN with input dimension m and output dimension n. There is a WNBA $\mathcal{A}_N$ of size $2^{\mathcal{O}(\|N\|)}$ s.t. $R(\mathcal{A}_N) = R_N$.*

A sketch of an auxiliary automaton called, recognizing the ternary addition relation, is given in Fig. 1. For details see [3]. There, similar auxiliary automata are

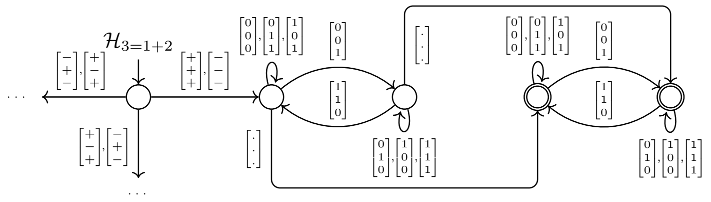


Fig. 1. WNBA that recognises a subset of the ternary addition relation.

constructed which recognize the multiplication or addition with a fixed rational constant or the application of the so called ReLU function, a popular choice for an activation function in DNN. Then, the overall idea of our construction uses combinations of such auxiliary automata to inductively build up an automaton representing the computation of a given DNN.

Using WNBA for Verification and Interpretation of DNN

We consider two possible use cases for our DNN to WNBA translation. First, the verification of adversarial robustness (AR) and output reachability (OR) properties and, second, the problem of finding a minimum sufficient reason (MinSufR) for some output described above. Our main result is that given some AR, OR or MinSufR property, we can use our translation of DNN into WNBA to reduce these problems to the emptiness problem of NBA respectively WNBA.

Theorem 2. *[3] Let N be a DNN with m inputs and n outputs, and P be an AR, OR or MinSufR property. There is a WNBA $\mathcal{A}^{N,P}$ of size $2^{\mathcal{O}(\|N\|+\|P\|)}$ s.t. $R(\mathcal{A}^{N,P}) = \emptyset$ iff $N \models P$.*

Translating DNN With Bounded Precision Leads to Equivalent NFA

Our translation from DNN to finite state automata presented above is carried out using Büchi automata. The reason for this is that infinite words are the most natural way to represent real numbers. But one can also use nondeterministic finite automata (NFA) over *finite* words. Clearly, for cardinality reasons alone their inputs cannot represent all real numbers. We can still use NFA at the expense of a less precise analysis. This can be seen as an abstraction mapping real numbers to their nearest integer for instance. On the other side using NFA enables us to use more efficient algorithms for minimisation.

We present a proof-of-concept implementation[3], translating so called Binarized Neural Networks (BNN), into input-output equivalent NFA. First experiments (see referenced repository) clearly show the exponential blow up in the translation but also indicate that automata-theoretic tools like minimisation can decrease the size of the resulting automaton tremendously.

[3] https://github.com/marcosaelzer/NN2NFA

References

1. Barceló, P., Monet, M., Pérez, J., Subercaseaux, B.: Model interpretability through the lens of computational complexity. In: Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., Lin, H. (eds.) Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual (2020), https://proceedings.neurips.cc/paper/2020/hash/b1adda14824f50ef24ff1c05bb66faf3-Abstract.html
2. Katz, G., Barrett, C.W., Dill, D.L., Julian, K., Kochenderfer, M.J.: Reluplex: An efficient SMT solver for verifying deep neural networks. In: Majumdar, R., Kuncak, V. (eds.) Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I. Lecture Notes in Computer Science, vol. 10426, pp. 97–117. Springer (2017). https://doi.org/10.1007/978-3-319-63387-9_5
3. Sälzer, M., Alsmann, E., Bruse, F., Lange, M.: Verifying and interpreting neural networks using finite automata (2022). https://doi.org/10.48550/ARXIV.2211.01022, https://arxiv.org/abs/2211.01022
4. Sälzer, M., Lange, M.: Reachability is NP-complete even for the simplest neural networks. In: Bell, P.C., Totzke, P., Potapov, I. (eds.) Reachability Problems - 15th International Conference, RP 2021, Liverpool, UK, October 25-27, 2021, Proceedings. Lecture Notes in Computer Science, vol. 13035, pp. 149–164. Springer (2021). https://doi.org/10.1007/978-3-030-89716-1_10

Verifying Learning-Based Robotic Navigation Systems

Guy Amir[1,*], Davide Corsi[2,*], Raz Yerushalmi[1,3], Luca Marzari[2], David Harel[3], Alessandro Farinelli[2], and Guy Katz[1]

[1] The Hebrew University of Jerusalem, Jerusalem, Israel
{guyam, guykatz}@cs.huji.ac.il
[2] University of Verona, Verona, Italy
{davide.corsi, luca.marzari, alessandro.farinelli}@univr.it
[3] The Weizmann Institute of Science, Rehovot, Israel
{raz.yerushalmi, david.harel}@weizmann.ac.il

Abstract. Deep reinforcement learning (DRL) has become a dominant deep-learning paradigm for tasks where complex policies are learned within reactive systems. Unfortunately, these policies are known to be susceptible to bugs. Despite significant progress in DNN verification, there has been little work demonstrating the use of modern verification tools on real-world, DRL-controlled systems. In this case study, we attempt to begin bridging this gap, and focus on the important task of mapless robotic navigation — a classic robotics problem, in which a robot, usually controlled by a DRL agent, needs to efficiently and safely navigate through an unknown arena towards a target. We demonstrate how modern verification engines can be used for effective *model selection*, i.e., selecting the best available policy for the robot in question from a pool of candidate policies. Specifically, we use verification to detect and rule out policies that may demonstrate suboptimal behavior, such as collisions and infinite loops. We also apply verification to identify models with overly conservative behavior, thus allowing users to choose superior policies, which might be better at finding shorter paths to a target. To validate our work, we conducted extensive experiments on an actual robot, and confirmed that the suboptimal policies detected by our method were indeed flawed. We also demonstrate the superiority of our verification-driven approach over state-of-the-art, gradient attacks. Our work is the first to establish the usefulness of DNN verification in identifying and filtering out suboptimal DRL policies in real-world robots, and we believe that the methods presented here are applicable to a wide range of systems that incorporate deep-learning-based agents.

1 Introduction

In recent years, *deep neural networks* (DNN) have become extremely popular, due to achieving state-of-the-art results in a variety of fields — such as natural

[*] Both authors contributed equally.

language processing [20], image recognition [69], autonomous driving [13], and more. The immense success of these DNN models is owed in part to their ability to train on a fixed set of training samples drawn from some distribution, and then *generalize*, i.e., correctly handle inputs that they had not encountered previously. Notably, *deep reinforcement learning* (DRL) [51] has recently become a dominant paradigm for training DNNs that implement control policies for complex systems that operate within rich environments. One domain in which DRL controllers have been especially successful is robotics, and specifically — robotic navigation, i.e., the complex task of efficiently navigating a robot through an arena, in order to safely reach a target [82,90].

Unfortunately, despite the immense success of DNNs, they have been shown to suffer from various safety issues [43,76]. For example, small perturbations to their inputs, which are either intentional or the result of noise, may cause DNNs to react in unexpected ways [60]. These inherent weaknesses, and others, are observed in almost every kind of neural network, and indicate a need for techniques that can supply formal guarantees regarding the safety of the DNN in question. These weaknesses have also been observed in DRL systems [6,26,46], showing that even state-of-the-art DRL models may err miserably.

To mitigate such safety issues, the verification community has recently developed a plethora of techniques and tools [10,12,23,30,37,39,43,47,53,54,83,88] for formally verifying that a DNN model is safe to deploy. Given a DNN, these methods usually check whether the DNN: (i) behaves according to a prescribed requirement for *all* possible inputs of interest; or (ii) violates the requirement, in which case the verification tool also provides a counterexample.

To date, despite the abundance of both DRL systems and DNN verification techniques, little work has been published on demonstrating the applicability and usefulness of verification techniques to real-world DRL systems. In this case study, we showcase the capabilities of DNN verification tools for analyzing DRL-based systems in the robotics domain — specifically, robotic navigation systems. To the best of our knowledge, this is the first attempt to demonstrate how off-the-shelf verification engines can be used to identify both *unsafe* and *suboptimal* DRL robotic controllers, that cannot be detected otherwise using existing, incomplete methods. Our approach leverages existing DNN verifiers that can reason about single and multiple invocations of DRL controllers, and this allows us to conduct a verification-based model selection process — through which we filter out models that could render the system unsafe.

In addition to model selection, we demonstrate how verification methods allow gaining better insights into the DRL training process, by comparing the outcomes of different training methods and assessing how the models improve over additional training iterations. We also compare our approach to gradient-based methods, and demonstrate the advantages of verification-based tools in this setting. We regard this as another step towards increasing the reliability and safety of DRL systems, which is one of the key challenges in modern machine learning [36]; and also as a step toward a more wholesome integration of verification techniques into the DRL development cycle.

In order to validate our experiments, we conducted an extensive evaluation on a real-world, physical robot. Our results demonstrate that policies classified as suboptimal by our approach indeed exhibited unwanted behavior. This evaluation highlights the practical nature of our work; and is summarized in a short video clip [4], which we strongly encourage the reader to watch. In addition, our code and benchmarks are available online [3].

The rest of the paper is organized as follows. Section 2 contains background on DNNs, DRLs, and robotic controlling systems. In Section 3 we present our DRL robotic controller case study, and then elaborate on the various properties that we considered in Section 4. In Section 5 we present our experimental results, and use them to compare our approach with competing methods. Related work appears in Section 6, and we conclude in Section 7.

2 Background

Deep Neural Networks. Deep neural networks (DNNs) [33] are computational, directed, graphs consisting of multiple layers. By assigning values to the first layer of the graph and propagating them through the subsequent layers, the network computes either a label prediction (for a classification DNN) or a value (for a regression DNN), which is returned to the user. The values computed in each layer depend on values computed in previous layers, and also on the current layer's *type*. Common layer types include the *weighted sum* layer, in which each neuron is an affine transformation of the neurons from the preceding layer; as well as the popular *rectified linear unit* (*ReLU*) layer, where each node y computes the value $y = \text{ReLU}(x) = \max(0, x)$, based on a single node x from the preceding layer to which it is connected. The DRL systems that are the subject matter of this case study consist solely of weighted sum and ReLU layers, although the techniques mentioned are suitable for DNNs with additional layer types, as we discuss later.

Fig. 1 depicts a small example of a DNN. For input $V_1 = [2, 3]^T$, the second (weighted sum) layer computes the values $V_2 = [20, -7]^T$. In the third layer, the ReLU functions are applied, and the result is $V_3 = [20, 0]^T$. Finally, the network's single output is computed as a weighted sum: $V_4 = [40]$.

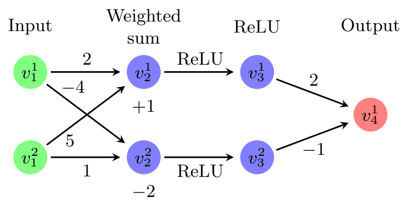


Fig. 1: A toy DNN.

Deep Reinforcement Learning. Deep reinforcement learning (DRL) [51] is a particular paradigm and setting for training DNNs. In DRL, an *agent* is trained to learn a *policy* π, which maps each possible *environment state* s (i.e., the current observation of the agent) to an *action* a. The policy can have different interpretations among various learning algorithms. For example, in some cases, π represents a probability distribution over the action space, while in others it encodes a function that estimates a *desirability score* over all the future actions from a state s.

During training, at each discrete time-step $t \in \{0, 1, 2, \ldots\}$, a *reward* r_t is presented to the agent, based on the action a_t it performed at time-step t. Different DRL training algorithms leverage the reward in different ways, in order to optimize the DNN-agent's parameters during training. The general DNN architecture described above also characterizes DRL-trained DNNs; the uniqueness of the DRL paradigm lies in the training process, which is aimed at generating a DNN that computes a mapping π that maximizes the *expected cumulative discounted reward* $R_t = \mathbb{E}\big[\sum_t \gamma^t \cdot r_t\big]$. The *discount factor*, $\gamma \in \big[0, 1\big]$, is a hyperparameter that controls the influence that past decisions have on the total expected reward.

DRL training algorithms are typically divided into three categories [74]:

1. **Value-Based Algorithms.** These algorithms attempt to learn a value function (called the *Q-function*) that assigns a value to each $\langle$state,action$\rangle$ pair. This iterative process relies on the *Bellman equation* [59] to update the function: $\mathbb{Q}^\pi(s_t, a_t) = r + \gamma \max_{a_{t+1}} \mathbb{Q}^\pi(s_{t+1}, a_{t+1})$. *Double Deep Q-Network* (DDQN) is an optimized implementation of this algorithm [79].
2. **Policy-Gradient Algorithms.** This class contains algorithms that attempt to directly learn the optimal policy, instead of assessing the value function. The algorithms in this class are typically based on the *policy gradient theorem* [75]. A common implementation is the *Reinforce* algorithm [89], which aims to directly optimize the following objective function, over the parameters θ of the DNN, through a gradient ascent process: $\nabla_\theta \mathbb{J}(\pi_\theta) = \mathbb{E}[\sum_t^T \nabla_\theta \log \pi_\theta(a_t|s_t) \cdot r_t]$. For additional details, see [89].
3. **Actor-Critic Algorithms.** This family of hybrid algorithms combines the two previous approaches. The key idea is to use two different neural networks: a *critic*, which learns the value function from the data, and an *actor*, which iteratively improves the policy by maximizing the value function learned by the critic. A state-of-the-art implementation of this approach is the *Proximal Policy Optimization* (PPO) algorithm [68].

All of these approaches are commonly used in modern DRL; and each has its advantages and disadvantages. For example, the value-based methods typically require only small sets of examples to learn from, but are unable to learn policies for continuous spaces of $\langle$state,action$\rangle$ pairs. In contrast, the policy-gradient methods can learn continuous policies, but suffer from a low sample efficiency and large memory requirements. Actor-Critic algorithms attempt to combine the benefits of value-based and policy-gradient methods, but suffer from high instability, particularly in the early stages of training, when the value function learned by the critic is unreliable.

DNN Verification and DRL Verification. A DNN verification algorithm receives as input [43]: (i) a trained DNN N; (ii) a precondition P on the DNN's inputs, which limits their possible assignments to inputs of interest; and (iii) a postcondition Q on N's output, which usually encodes the *negation* of the behavior we would like N to exhibit on inputs that satisfy P. The verification algorithm then searches for a concrete input x_0 that satisfies $P(x_0) \wedge Q(N(x_0))$, and returns one of the following outputs: (i) `SAT`, along with a concrete input x_0 that satisfies the given constraints; or (ii) `UNSAT`, indicating that no such x_0 exists. When Q encodes the negation of the required property, a `SAT` result indicates that the property is violated (and the returned input x_0 triggers a bug), while an `UNSAT` result indicates that the property holds.

For example, suppose we wish to verify that the DNN in Fig. 1 always outputs a value strictly smaller than 7; i.e., that for any input $x = \langle v_1^1, v_1^2 \rangle$, it holds that $N(x) = v_4^1 < 7$. This is encoded as a verification query by choosing a precondition that does not restrict the input, i.e., $P = (true)$, and by setting $Q = (v_4^1 \geq 7)$, which is the *negation* of our desired property. For this verification query, a sound verifier will return `SAT`, alongside a feasible counterexample such as $x = \langle 0, 2 \rangle$, which produces $v_4^1 = 22 \geq 7$. Hence, the property does not hold for this DNN.

To date, the DNN verification community has focused primarily on DNNs used for a single, non-reactive, invocation [30,37,43,54,83]. Some work has been carried out on verifying DRL networks, which pose greater challenges: beyond the general scalability challenges of DNN verification, in DRL verification we must also take into account that agents typically interact with a reactive environment [6, 11, 18, 26, 41]. In particular, these agents are implemented with neural networks that are invoked multiple times, and the inputs of each invocation are usually affected by the outputs of the previous invocations. This fact aggregates the scalability limitations (because multiple invocations must be encoded in each query), and also makes the task of defining P and Q significantly more complex [6].

3 Case Study: Robotic Mapless Navigation

Robotis Turtlebot 3. In our case study, we focus on the *Robotis Turtlebot 3* robot (*Turtlebot*, for short), depicted in Fig. 2. Given its relatively low cost and efficient sensor configuration, this robot is widely used in robotics research [9,61]. In particular, this robotic platform has the actuators required for moving and turning, as well as multiple lidar sensors for detecting obstacles. These sensors use laser beams to approximate the distance to the nearest object in their direction [86]. In our experiments, we used a configuration with seven lidar sensors, each with a maximal range of one meter. Each pair of sensors are 30° apart, thus allowing coverage of 180°. The images in Fig. 3 depict a simulation of the Turtlebot navigating through an arena, and highlight the lidar beams. See Section B of the Appendix for additional details.

The Mapless Navigation Problem. *Robotic navigation* is the task of navigating a robot (in our case, the Turtlebot) through an arena. The robot's goal is to reach a target destination while adhering to predefined restrictions; e.g., selecting as short a path as possible, avoiding obstacles, or optimizing energy consumption. In recent years, robotic navigation tasks have received a great deal of attention [82,90], primarily due to their applicability to autonomous vehicles.

We study here the popular *mapless* variant of the robotic navigation problem, where the robot can rely only on local observations (i.e., its sensors), without

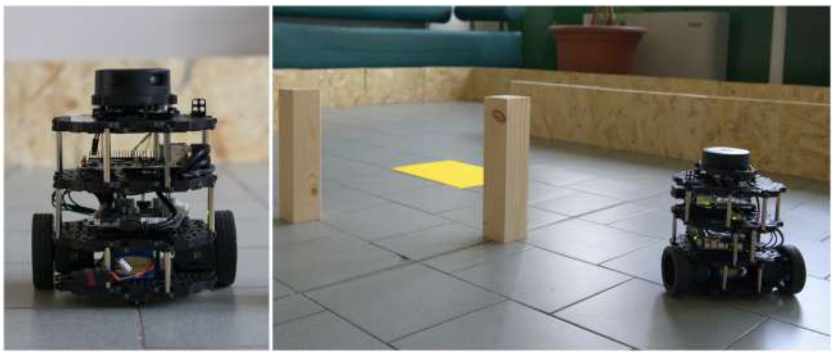


Fig. 2: The *Robotis Turtlebot 3* platform, navigating in an arena. The image on the left depicts a static robot, and the image on the right depicts the robot moving towards the destination (the yellow square), while avoiding two wooden obstacles in its route.

any information about the arena's structure or additional data from external sources. In this setting, which has been studied extensively [77], the robot has access to the *relative location* of the target, but does not have a *complete map* of the arena. This makes mapless navigation a partially observable problem, and among the most challenging tasks to solve in the robotics domain [16,58,77,92].

DRL-Controlled Mapless Navigation. State-of-the-art solutions to mapless navigation suggest training a DRL policy to control the robot. Such DRL-based solutions have obtained outstanding results from a performance point of view [63]. For example, recent work by Marchesini et al. [57] has demonstrated how DRL-based agents can be applied to control the Turtlebot in a mapless navigation setting, by training a DNN with a simple architecture, including two hidden layers. Following this recent work, in our case study we used the following topology for DRL policies:

– An input layer with nine neurons. These include seven neurons representing the Turtlebot's lidar readings. The additional, non-lidar inputs include one neuron representing the relative angle between the robot and the target, and one neuron representing the robot's distance from the target. A scheme of the inputs appears in Fig. 4a.
– Two subsequent fully-connected layers, each consisting of 16 neurons, and followed by a ReLU activation layer.
– An output layer with three neurons, each corresponding to a different (discrete) action that the agent can choose to execute in the following step: move `FORWARD`, turn `LEFT`, or turn `RIGHT`.[1]

[1] It has been shown that discrete controllers achieve excellent performance in robotic navigation, often outperforming continuous controllers in a large variety of tasks [57].

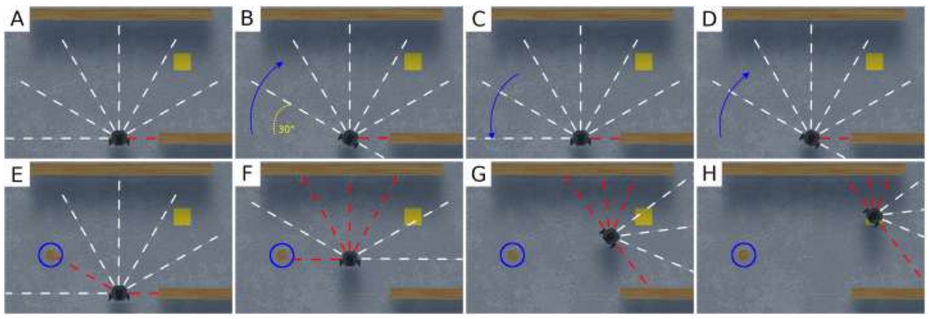


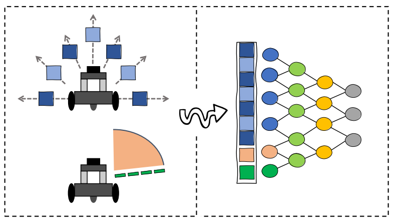
Fig. 3: An example of a simulated Turtlebot entering a 2-step loop. The white and red dashed lines represent the lidar beams (white indicates "clear", and red indicates that an obstacle is detected). The yellow square represents the target position; and the blue arrows indicate rotation. In the first row, from left to right, the Turtlebot is stuck in an infinite loop, alternating between right and left turns. Given the deterministic nature of the system, the agent will continue to select these same actions, ad infinitum. In the second row, from left to right, we present an almost identical configuration, but with an obstacle located 30° to the robot's left (circled in blue). The presence of the obstacle changes the input to the DNN, and allows the Turtlebot to avoid entering the infinite loop; instead, it successfully navigates to the target.

While the aforementioned DRL topology has been shown to be efficient for robotic navigation tasks, finding the optimal training algorithm and reward function is still an open problem. As part of our work, we trained multiple *deterministic* policies using the DRL algorithms presented in Section 2: DDQN [79], Reinforce [89], and PPO [68]. For the reward function, we used the following formulation:

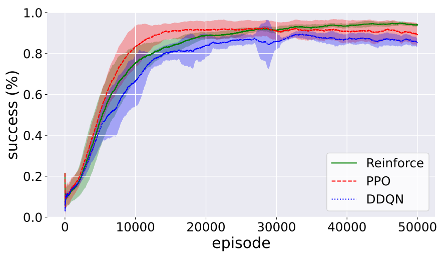
$$\mathbb{R}_t = (d_{t-1} - d_t) \cdot \alpha - \beta,$$

where d_t is the distance from the target at time-step t; α is a normalization factor used to guarantee the stability of the gradient; and β is a fixed value, decreased at each time-step, and resulting in a total penalty proportional to the length of the path (by minimizing this penalty, the agent is encouraged to reach the target quickly). In our evaluation, we empirically selected $\alpha = 3$ and $\beta = 0.001$. Additionally, we added a final reward of $+1$ when the robot reached the target, or -1 in case it collided with an obstacle. For additional information regarding the parameters chosen for the training phase, see Section A of the Appendix.

DRL Training and Results. Using the training algorithms mentioned in Section 2, we trained a collection of DRL agents to solve the Turtlebot mapless navigation problem. We ran a stochastic training process, and thus obtained varied agents; of these, we only kept those that achieved a success rate of at least 96% during training. A total of 780 models were selected, consisting of 260 models per each of the three training algorithms. More specifically, for each



(a) The DRL controller



(b) Average success rates

Fig. 4: (a) The DRL controller used for the robot in our case study. The DRL has nine input neurons: seven lidar sensor readings (blue), one input indicating the relative angle (orange) between the robot and the target, and one input indicating the distance (green) between the robot and the target. (b) The average success rates of models trained by each of the three DRL training algorithms, per training episode.

algorithm, all 260 models were generated from 52 random seeds. Each seed gave rise to a family of 5 models, where the individual family members differ in the number of training episodes used for training them. Fig. 4b shows the trained models' average success rate, for each algorithm used. We note that PPO was generally the fastest to achieve high accuracy. However, all three training algorithms successfully produced highly accurate agents.

4 Using Verification for Model Selection

All of our trained models achieved very high success rates, and so, at face value, there was no reason to favor one over the other. However, as we show next, a verification-based approach can expose multiple subtle differences between them. As our evaluation criteria, we define two properties of interest that are derived from the main goals of the robotic controller: (i) reaching the target; and (ii) avoiding collision with obstacles. Employing verification, we use these criteria to identify models that may fail to fulfill their goals, e.g., because they collide with various obstacles, are overly conservative, or may enter infinite loops without reaching the target. We now define the properties that we used, and the results of their verification are discussed in Section 5. Additional details regarding the precise encoding of our queries appear in Section D of the Appendix.

Collision Avoidance. Collision avoidance is a fundamental and ubiquitous safety property [17] for navigation agents. In the context of Turtlebot, our goal is to check whether there exists a setting in which the robot is facing an obstacle, and chooses to move forward — even though it has at least one other viable option, in the form of a direction in which it is not blocked. In such situations, it is clearly preferable to choose to turn `LEFT` or `RIGHT` instead of choosing to move `FORWARD` and collide. See Fig. 5 for an illustration.

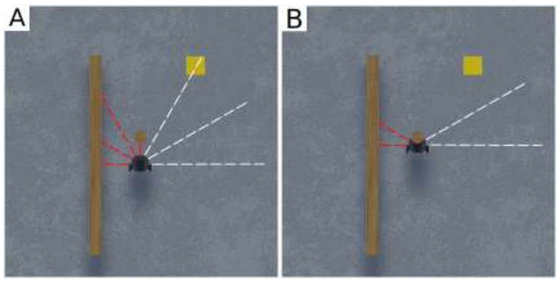


Fig. 5: Example of a single-step collision. The robot is not blocked on its right and can avoid the obstacle by turning (panel A), but it still chooses to move forward — and collides (panel B).

Given that turning `LEFT` or `RIGHT` produces an in-place rotation (i.e., the robot does not change its position), the only action that can cause a collision is `FORWARD`. In particular, a collision can happen when an obstacle is directly in front of the robot, or is slightly off to one side (just outside the front lidar's field of detection). More formally, we consider the safety property *"the robot does not collide at the next step"*, with three different types of collisions:

– `FORWARD COLLISION`: the robot detects an obstacle straight ahead, but nevertheless makes a step forward and collides with the obstacle.
– `LEFT COLLISION`: the robot detects an obstacle ahead and slightly shifted to the left (using the lidar beam that is 30° to the left of the one pointing straight ahead), but makes a single step forward and collides with the obstacle. The shape of the robot is such that in this setting, a collision is unavoidable.
– `RIGHT COLLISION`: the robot detects an obstacle ahead and slightly shifted to the right, but makes a single step forward and collides with the obstacle.

Recall that in mapless navigation, all observations are local — the robot has no sense of the global map, and can encounter any possible obstacle configuration (i.e., any possible sensor reading). Thus, in encoding these properties, we considered a single invocation of the DRL agent's DNN, with the following constraints:

1. All the sensors that are not in the direction of the obstacle receive a lidar input indicating that the robot can move either `LEFT` or `RIGHT` without risk of collision. This is encoded by lower-bounding these inputs.
2. The single input in the direction of the obstacle is upper-bounded by a value matching the representation of an obstacle, close enough to the robot so that it will collide if it makes a move `FORWARD`.
3. The input representing the distance to the target is lower-bounded, indicating that the target has not yet been reached (encouraging the agent to make a move).

The exact encoding of these properties is based on the physical characteristics of the robot and the lidar sensors, as explained in Section B of the Appendix.

Infinite Loops. Whereas collision avoidance is the natural safety property to verify in mapless navigation controllers, checking that progress is eventually made towards the target is the natural liveness property. Unfortunately, this property is difficult to formulate due to the absence of a complete map. Instead, we settle for a weaker property, and focus on verifying that the robot does not enter infinite loops (which would prevent it from ever reaching the target).

Unlike the case of collision avoidance, where a single step of the DRL agent could constitute a violation, here we need to reason about multiple consecutive invocations of the DRL controller, in order to identify infinite loops. This, again, is difficult to encode due to the absence of a global map, and so we focus on *in-place* loops: infinite sequences of steps in which the robot turns `LEFT` and `RIGHT`, but without ever moving `FORWARD`, thus maintaining its current location ad infinitum.

Our queries for identifying in-place loops encode that: (i) the robot does not reach the target in the first step; (ii) in the following k steps, the robot never moves `FORWARD`, i.e., it only performs turns; and (iii) the robot returns to an already-visited configuration, guaranteeing that the same behavior will be repeated by our deterministic agents. The various queries differ in the choice of k, as well as in the sequence of turns performed by the robot. Specifically, we encode queries for identifying the following kinds of loops:

– `ALTERNATING LOOP`: a loop where the robot performs an infinite sequence of $\langle$`LEFT`, `RIGHT`, `LEFT`, `RIGHT`, `LEFT`...$\rangle$ moves. A query for identifying this loop encodes $k = 2$ consecutive invocations of the DRL agent, after which the robot's sensors will again report the exact same reading, leading to an infinite loop. An example appears in Fig. 3. The encoding uses the "sliding window" principle, on which we elaborate later.
– `LEFT CYCLE`, `RIGHT CYCLE`: loops in which the robot performs an infinite sequence of $\langle$`LEFT`, `LEFT`, `LEFT`, $\ldots\rangle$ or $\langle$`RIGHT`, `RIGHT`, `RIGHT`, $\ldots\rangle$ operations accordingly. Because the Turtlebot turns at a $30°$ angle, this loop is encoded as a sequence of $k = 360°/30° = 12$ consecutive invocations of the DRL agent's DNN, all of which produce the same turning action (either `LEFT` or `RIGHT`). Using the sliding window principle guarantees that the robot returns to the same exact configuration after performing this loop, indicating that it will never perform any other action.

We also note that all the loop-identification queries include a condition for ensuring that the robot is not blocked from all directions. Consequently, any loops that are discovered demonstrate a clearly suboptimal behavior.

Specific Behavior Profiles. In our experiments, we noticed that the safe policies, i.e., the ones that do not cause the robot to collide, displayed a wide spectrum of different behaviors when navigating to the target. These differences occurred not only between policies that were trained by different algorithms, but also between policies trained by the same reward strategy — indicating that these differences are, at least partially, due to the stochastic realization of the DRL training process.

Specifically, we noticed high variability in the length of the routes selected by the DRL policy in order to reach the given target: while some policies demonstrated short, efficient, paths that passed very close to obstacles, other policies demonstrated a much more conservative behavior, by selecting longer paths, and avoiding getting close to obstacles (an example appears in Fig. 6).

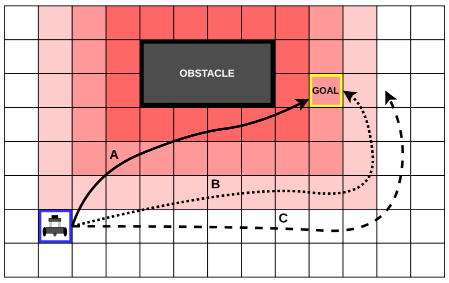


Fig. 6: Comparing paths selected by policies with different *bravery* levels. Path *A* takes the Turtlebot close to the obstacle (red area), and is the shortest. Path *B* maintains a greater distance from the obstacle (light red area), and is consequently longer. Finally, path *C* maintains such a significant distance from the obstacle (white area) that it is unable to reach the target.

Thus, we used our verification-driven approach to quantify how conservative the learned DRL agent is in the mapless navigation setting. Intuitively, a highly conservative policy will keep a significant safety margin from obstacles (possibly taking a longer route to reach its destination), whereas a "braver" and less conservative controller would risk venturing closer to obstacles. In the case of Turtlebot, the preferable DRL policies are the ones that guarantee the robot's safety (with respect to collision avoidance), and demonstrate a high level of bravery — as these policies tend to take shorter, optimized paths (see path A in Fig. 6), which lead to reduced energy consumption over the entire trail.

Bravery assessment is performed by encoding verification queries that identify situations in which the Turtlebot *can* move forward, but its control policy chooses not to. Specifically, we encode single invocations of the DRL model, in which we bound the lidar inputs to indicate that the Turtlebot is sufficiently distant from any obstacle and can safely move forward. We then use the verifier to determine whether, in this setting, a `FORWARD` output is possible. By altering and adjusting the bounds on the central lidar sensor, we can control how far away the robot perceives the obstacle to be. If we limit this distance to large values and the policy will still not move `FORWARD`, it is considered conservative; otherwise, it is considered brave. By conducting a binary search over these bounds [6], we can identify the shortest distance from an obstacle for which the policy *safely* orders the robot to move `FORWARD`. This value's inverse then serves as a bravery score for that policy.

Design-for-Verification: Sliding Windows. A significant challenge that we faced in encoding our verification properties, especially those that pertain to multiple consecutive invocations of the DRL policy, had to do with the local nature of the sensor readings that serve as input to the DNN. Specifically, if the robot is in some initial configuration that leads to a sensor input x, and then chooses to move forward and reaches a successor configuration in which the sensor input is x', some connection between x and x' must be expressed as part of the verification query (i.e., nearby obstacles that exist in x cannot suddenly vanish in x'). In the absence of a global map, this is difficult to enforce.

In order to circumvent this difficulty, we used the *sliding window* principle, which has proven quite useful in similar settings [6, 26]. Intuitively, the idea is to focus on scenarios where the connections between x and x' are particularly straightforward to encode — in fact, most of the sensor information that appeared in x also appears in x'. This approach allows us to encode multistep queries, and is also beneficial in terms of performance: typically, adding sliding-window constraints reduces the search space explored by the verifier, and expedites solving the query.

In the Turtlebot setting, this is achieved by selecting a robot configuration in which the angle between two neighboring lidar sensors is identical to the turning angle of the robot (in our case, $30°$). This guarantees, for example, that if the central lidar sensor observes an obstacle at distance d and the robot chooses to turn `RIGHT`, then at the next step, the lidar sensor just to the left of the central sensor must detect the same obstacle, at the same distance d. More generally, if at time-step t the 7 lidar readings (from left to right) are $\langle l_1, \ldots, l_7\rangle$ and the robot turns `RIGHT`, then at time-step $t + 1$ the 7 readings are $\langle l_2, l_3, \ldots, l_7, l_8\rangle$, where only l_8 is a new reading. The case for a `LEFT` turn is symmetrical. By placing these constraints on consecutive states encountered by the robot, we were able to encode complex properties that involve multiple time-steps, e.g., as in the aforementioned infinite loops. An illustration appears in Fig. 3.

5 Experimental Evaluation

Next, we ran verification queries with the aforementioned properties, in order to assess the quality of our trained DRL policies. The results are reported below. In many cases, we discovered configurations in which the policies would cause the robot to collide or enter infinite loops; and we later validated the correctness of these results using a physical robot. We strongly encourage the reader to watch a short video clip that demonstrates some of these results [4]. Our code and benchmarks are also available online [3]. In our experiments, We used the *Marabou* verification engine [45] as our backend, although other engines could be used as well. For additional details regarding the experiments, we refer the reader to Section D of the Appendix.

Model Selection. In this set of experiments, we used verification to assess our trained models. Specifically, we used each of the three training algorithms (DDQN, Reinforce, PPO) to train 260 models, creating a total of 780 models. For each of these, we verified six properties of interest: three collision properties (`FORWARD COLLISION`, `LEFT COLLISION`, `RIGHT COLLISION`), and three loop properties (`ALTERNATING LOOP`, `LEFT CYCLE`, `RIGHT CYCLE`), as described in Section 4. This gives a total of 4680 verification queries. We ran all queries with a

	`LEFT COLLISION`		`FORWARD COLLISION`		`RIGHT COLLISION`	
`Algorithm`	`SAT`	`UNSAT`	`SAT`	`UNSAT`	`SAT`	`UNSAT`
DDQN	259	1	248	12	258	2
Reinforce	255	5	254	6	252	8
PPO	196	64	197	63	207	53

	`ALTERNATING LOOP`		`LEFT CYCLE`		`RIGHT CYCLE`		`INSTABILITY`
`Algorithm`	`SAT`	`UNSAT`	`SAT`	`UNSAT`	`SAT`	`UNSAT`	`# alternations`
DDQN	260	0	56	77	56	61	21
Reinforce	145	115	5	185	120	97	10
PPO	214	45	26	198	30	198	1

Table 1: Results of the policy verification queries. We verified six properties over each of the 260 models trained per algorithm; `SAT` indicates that the property was violated, whereas `UNSAT` indicates that it held (to reduce clutter, we omit `TIMEOUT` and `FAIL` results). The rightmost column reports the stability values of the various training methods. For the full results see [3].

`TIMEOUT` value of 12 hours and a `MEMOUT` limit of $2G$; the results are summarized in Table 1. The single-step collision queries usually terminated within seconds, and the 2-step queries encoding an `ALTERNATING LOOP` usually terminated within minutes. The 12-step cycle queries, which are more complex, usually ran for a few hours. 9.6% of all queries hit the `TIMEOUT` limit (all from the 12-step cycle category), and none of the queries hit the `MEMOUT` limit.[2]

Our results exposed various differences between the trained models. Specifically, of the 780 models checked, 752 (over 96%) violated at least one of the single-step collision properties. These 752 collision-prone models include *all* 260 DDQN-trained models, 256 Reinforce models, and 236 PPO models. Furthermore, when we conducted a model filtering process based on all six properties (three collisions and three infinite loops), we discovered that 778 models out of the total of 780 (over 99.7%!) violated at least one property. The only two models that passed our filtering process were trained by the PPO algorithm.

Further analyzing the results, we observed that PPO models tended to be safer to use than those trained by other algorithms: they usually had the fewest violations per property. However, there are cases in which PPO proved less successful. For example, our results indicate that PPO-trained models are more prone to enter an `ALTERNATING LOOP` than those trained by Reinforce. Specifically, 214 (82.3%) of the PPO models have entered this undesired state, compared to 145 (55.8%) of the Reinforce models. We also point out that, similarly to the case with collision properties, *all* DDQN models violated this property.

Finally, when considering 12-step cycles (either `LEFT CYCLE` or `RIGHT CYCLE`), 44.8% of the DDQN models entered such cycles, compared to 30.7% of the Reinforce models, and just 12.4% of the PPO models. In computing these results, we

[2] We note that two queries failed due to internal errors in *Marabou*.

computed the fraction of violations (SAT queries) out of the number of queries that did not time out or fail, and aggregated SAT results for both cycle directions.

Interestingly, in some cases, we observed a bias toward violating a certain subcase of various properties. For example, in the case of entering full cycles — although 125 (out of 520) queries indicated that Reinforce-trained agents may enter a cycle in either direction, in 96% of these violations, the agent entered a RIGHT CYCLE. This bias is not present in models trained by the other algorithms, where the violations are roughly evenly divided between cycles in both directions.

We find that our results demonstrate that different "black-box" algorithms generalize very differently with respect to various properties. In our setting, PPO produces the safest models, while DDQN tends to produce models with a higher number of violations. We note that this does not necessarily indicate that PPO-trained models perform better, but rather that they are more robust to corner cases. Using our filtering mechanism, it is possible to select the safest models among the available, seemingly equivalent candidates.

Next, we used verification to compute the bravery score of the various models. Using a binary search, we computed for each model the minimal distance a dead-ahead obstacle needs to have for the robot to *safely* move forward. The search range was $[0.18, 1]$ meters, and the optimal values were computed up to a 0.01 precision (see Section D of the Appendix for additional details). Almost all binary searches terminated within minutes, and none hit the TIMEOUT threshold.

By first filtering the models based on their safe behavior, and then by their bravery scores, we are able to find the few models that are both safe (do not collide), and not overly conservative. These models tend to take efficient paths, and may come close to an obstacle, but without colliding with it. We also point out that over-conservativeness may significantly reduce the success rate in specific scenarios, such as cases in which the obstacle is close to the target. Specifically, of the only two models that survived the first filtering stage, one is considerably more conservative than the other — requiring the obstacle to be twice as distant as the other, braver, model requires it to be, before moving forward.

Algorithm Stability Analysis. As part of our experiments, we used our method to assess the three training algorithms — DDQN, PPO, and Reinforce. Recall that we used each algorithm to train 52 families of 5 models each, in which the models from the same family are generated from the same random seed, but with a different number of training iterations. While all models obtained a high success rate, we wanted to check how often it occurred that a model successfully learned to satisfy a desirable property after some training iterations, only to forget it after additional iterations. Specifically, we focused on the 12-step full-cycle properties (LEFT CYCLE and RIGHT CYCLE), and for each family of 5 models checked whether some models satisfied the property while others did not.

We define a family of models to be *unstable* in the case where a property holds in the family, but ceases to hold for another model from the same family with a higher number of training iterations. Intuitively, this means that the model "forgot" a desirable property as training progressed. The *instability value* of each algorithm type is defined to be the number of unstable 5-member families.

Although all three algorithms produced highly accurate models, they displayed significant differences in the stability of their produced policies, as can be seen in the rightmost column of Table 1. Recall that we trained 52 families of models using each algorithm, and then tested their stability with respect to two properties (corresponding to the two full cycle types). Of these, the DDQN models display 21 *unstable* alternations — more than twice the number of alterations demonstrated by Reinforce models (10), and significantly higher than the number of alternations observed among the PPO models (1).

These results shed light on the nature of these training algorithms — indicating that DDQN is a significantly less stable training algorithm, compared to PPO and Reinforce. This is in line with previous observations in non-verification-related research [68], and is not surprising, as the primary objective of PPO is to limit the changes the optimizer performs between consecutive training iterations.

Gradient-Based Methods. We also conducted a thorough comparison between our verification-based approach and competing gradient-based methods. Although gradient-based attacks are extremely scalable, our results (summarized in Table 2 of the Appendix) show that they may miss many of the violations found by our complete, verification-based procedure. For example, when searching for collisions, our approach discovered a total of 2126 `SAT` results, while the gradient-based method discovered only 1421 `SAT` results — a 33% decrease (!). In addition, given that gradient-based methods are unable to return `UNSAT`, they are also incapable of proving that a property always holds, and hence cannot formally guarantee the safety of a policy in question. Thus, performing model selection based on gradient-based methods could lead to skewed results. We refer the reader to Section E of the Appendix, in which we elaborate on gradient attacks and the experiments we ran, demonstrating the advantages of our approach for model selection, when compared to gradient-based methods.

6 Related Work

Due to the increasing popularity of DNNs, the formal methods community has put forward a plethora of tools and approaches for verifying DNN correctness [24, 30, 35, 37, 43–45, 48, 53, 70, 78]. Recently, the verification of systems involving multiple DNN invocations, as well as hybrid systems with DNN components, has been receiving significant attention [6, 11, 21, 22, 28, 46, 73, 80]. Our work here is another step toward applying DNN verification techniques to additional, real-world systems and properties of interest.

In the robotics domain, multiple approaches exist for increasing the reliability of learning-based systems [65,81,91]; however, these methods are mostly heuristic in nature [1, 29, 56]. To date, existing techniques rely mostly on Lagrangian multipliers [52,67,71], and do not provide formal safety guarantees; rather, they optimize the training in an attempt to learn the required policies [14]. Other, more formal approaches focus solely on the systems' input-output relations [18, 55], without considering multiple invocations of the agent and its interactions with the environment. Thus, existing methods are not able to provide rigorous guarantees regarding the correctness of multistep robotic systems, and do not take into account sequential decision making — which renders them insufficient for detecting various safety and liveness violations.

Our approach is orthogonal and complementary to many existing safe DRL techniques. Reward reshaping and shielding techniques (e.g., [2]) improve safety by altering the training loop, but typically afford no formal guarantees. Our approach can be used to complement them, by selecting the most suitable policy from a pool of candidates, post-training. Guard rules and runtime shields are beneficial for preventing undesirable behavior of a DNN agent, but are sometimes less suited for specifying the *desired* actions it should take instead. In contrast, our approach allows selecting the optimal policy from a pool of candidates, without altering its decision-making.

7 Conclusion

Through the case study described in this paper, we demonstrate that current verification technology is applicable to real-world systems. We show this by applying verification techniques for improving the navigation of DRL-based robotic systems. We demonstrate how off-the-shelf verification engines can be used to conduct effective model selection, as well as gain insights into the stability of state-of-the-art training algorithms. As far as we are aware, ours is the first work to demonstrate the use of formal verification techniques on multistep properties of actual, real-world robotic navigation platforms. We also believe the techniques developed here will allow the use of verification to improve additional multistep systems (autonomous vehicles, surgery-aiding robots, etc.), in which we can impose a transition function between subsequent steps. However, our approach is limited by DNN-verification technology, which we use as a black-box backend. As that technology becomes more scalable, so will our approach. Moving forward, we plan to generalize our work to richer environments — such as cases where a memory-enhanced agent interacts with moving objects, or even with multiple agents in the same arena, as well as running additional experiments with deeper networks, and more complex DRL systems. In addition, we see probabilistic verification of stochastic policies as interesting future work.

Acknowledgements. The work of Amir, Yerushalmi and Katz was partially supported by the Israel Science Foundation (grant number 683/18). The work of Amir was supported by a scholarship from the Clore Israel Foundation. The work of Corsi, Marzari, and Farinelli was partially supported by the "Dipartimenti di Eccellenza 2018-2022" project, funded by the Italian Ministry of Education, Universities, and Research (MIUR). The work of Yerushalmi and Harel was partially supported by a research grant from the Estate of Harry Levine, the Estate of Avraham Rothstein, Brenda Gruss and Daniel Hirsch, the One8 Foundation, Rina Mayer, Maurice Levy, and the Estate of Bernice Bernath, grant 3698/21 from the ISF-NSFC (joint to the Israel Science Foundation and the National Science Foundation of China), and a grant from the Minerva foundation. We thank Idan Refaeli for his contribution to this project.

References

1. J. Achiam, D. Held, A. Tamar, and P. Abbeel. Constrained Policy Optimization. In *Proc. 34th Int. Conf. on Machine Learning (ICML)*, pages 22–31, 2017.
2. M. Alshiekh, R. Bloem, R. Ehlers, B. Könighofer, S. Niekum, and U. Topcu. Safe Reinforcement Learning via Shielding. In *Proc. 32th AAAI Conf. on Artificial Intelligence (AAAI)*, pages 2669–2678, 2018.
3. G. Amir, D. Corsi, R. Yerushalmi, L. Marzari, D. Harel, A. Farinelli, and G. Katz. Supplementary Artifact, 2022. https://zenodo.org/record/7479970.
4. G. Amir, D. Corsi, R. Yerushalmi, L. Marzari, D. Harel, A. Farinelli, and G. Katz. Supplementary Video, 2022. https://youtu.be/QIZqOgxLkAE.
5. G. Amir, Z. Freund, G. Katz, E. Mandelbaum, and I. Refaeli. veriFIRE: Verifying an Industrial, Learning-Based Wildfire Detection. In *Proc. 25th Int. Symposium on Formal Methods (FM)*, 2023.
6. G. Amir, M. Schapira, and G. Katz. Towards Scalable Verification of Deep Reinforcement Learning. In *Proc. 21st Int. Conf. on Formal Methods in Computer-Aided Design (FMCAD)*, pages 193–203, 2021.
7. G. Amir, H. Wu, C. Barrett, and G. Katz. An SMT-Based Approach for Verifying Binarized Neural Networks. In *Proc. 27th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pages 203–222, 2021.
8. G. Amir, T. Zelazny, G. Katz, and M. Schapira. Verification-Aided Deep Ensemble Selection. In *Proc. 22nd Int. Conf. on Formal Methods in Computer-Aided Design (FMCAD)*, pages 27–37, 2022.
9. R. Amsters and P. Slaets. Turtlebot 3 as a Robotics Education Platform. In *Proc. 10th Int. Conf. on Robotics in Education (RiE)*, pages 170–181, 2019.
10. G. Avni, R. Bloem, K. Chatterjee, T. Henzinger, B. Konighofer, and S. Pranger. Run-Time Optimization for Learned Controllers through Quantitative Games. In *Proc. 31st Int. Conf. on Computer Aided Verification (CAV)*, pages 630–649, 2019.
11. E. Bacci, M. Giacobbe, and D. Parker. Verifying Reinforcement Learning Up to Infinity. In *Proc. 30th Int. Joint Conf. on Artificial Intelligence (IJCAI)*, 2021.
12. T. Baluta, S. Shen, S. Shinde, K. Meel, and P. Saxena. Quantitative Verification of Neural Networks and its Security Applications. In *Proc. ACM SIGSAC Conf. on Computer and Communications Security (CCS)*, pages 1249–1264, 2019.
13. M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba. End to End Learning for Self-Driving Cars, 2016. Technical Report. http://arxiv.org/abs/1604.07316.
14. L. Brunke, M. Greeff, A. Hall, Z. Yuan, S. Zhou, J. Panerati, and A. Schoellig. Safe Learning in Robotics: From Learning-Based Control to Safe Reinforcement Learning. *Annual Review of Control, Robotics, and Autonomous Systems*, 5, 2021.
15. M. Casadio, E. Komendantskaya, M. Daggitt, W. Kokke, G. Katz, G. Amir, and I. Refaeli. Neural Network Robustness as a Verification Property: A Principled Case Study. In *Proc. 34th Int. Conf. on Computer Aided Verification (CAV)*, 2022.
16. H. Chiang, A. Faust, M. Fiser, and A. Francis. Learning Navigation Behaviors End-to-End with AutoRL. *IEEE Robotics and Automation Letters (RA-L/ICRA)*, 4(2):2007–2014, 2019.
17. E. Clarke, T. Henzinger, H. Veith, and R. Bloem. *Handbook of Model Checking*, volume 10. Springer, 2018.

18. D. Corsi, E. Marchesini, and A. Farinelli. Formal Verification of Neural Networks for Safety-Critical Tasks in Deep Reinforcement Learning. In *Proc. 37th Conf. on Uncertainty in Artificial Intelligence (UAI)*, pages 333–343, 2021.
19. D. Corsi, R. Yerushalmi, G. Amir, A. Farinelli, D. Harel, and G. Katz. Constrained Reinforcement Learning for Robotics via Scenario-Based Programming, 2022. Technical Report. `https://arxiv.org/abs/2206.09603`.
20. L. Deng and Y. Liu. *Deep Learning in Natural Language Processing.* Springer, 2018.
21. S. Dutta, X. Chen, and S. Sankaranarayanan. Reachability Analysis for Neural Feedback Systems using Regressive Polynomial Rule Inference. In *Proc. 22nd ACM Int. Conf. on Hybrid Systems: Computation and Control (HSCC)*, pages 157–168, 2019.
22. S. Dutta, S. Jha, S. Sankaranarayanan, and A. Tiwari. Learning and Verification of Feedback Control Systems using Feedforward Neural Networks. *IFAC-PapersOnLine*, 51(16):151–156, 2018.
23. S. Dutta, S. Jha, S. Sankaranarayanan, and A. Tiwari. Output Range Analysis for Deep Feedforward Neural Networks. In *Proc. 10th NASA Formal Methods Symposium (NFM)*, pages 121–138, 2018.
24. R. Ehlers. Formal Verification of Piece-Wise Linear Feed-Forward Neural Networks. In *Proc. 15th Int. Symp. on Automated Technology for Verification and Analysis (ATVA)*, pages 269–286, 2017.
25. Y. Elboher, J. Gottschlich, and G. Katz. An Abstraction-Based Framework for Neural Network Verification. In *Proc. 32nd Int. Conf. on Computer Aided Verification (CAV)*, pages 43–65, 2020.
26. T. Eliyahu, Y. Kazak, G. Katz, and M. Schapira. Verifying Learning-Augmented Systems. In *Proc. Conf. of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM)*, pages 305–318, 2021.
27. O. Ferhat and S. Yildirim-Yayilgan. Deep Neural Network Based Malicious Network Activity Detection Under Adversarial Machine Learning Attacks. In *Proc. 3rd Int. Conf. on Intelligent Technologies and Applications (INTAP)*, pages 280–291, 2020.
28. N. Fulton and A. Platzer. Safe Reinforcement Learning via Formal Methods: Toward Safe Control through Proof and Learning. In *Proc. 32nd AAAI Conf. on Artificial Intelligence (AAAI)*, 2018.
29. J. Garcıa and F. Fernández. A Comprehensive Survey on Safe Reinforcement Learning. *Journal of Machine Learning Research*, 16(1):1437–1480, 2015.
30. T. Gehr, M. Mirman, D. Drachsler-Cohen, E. Tsankov, S. Chaudhuri, and M. Vechev. AI2: Safety and Robustness Certification of Neural Networks with Abstract Interpretation. In *Proc. 39th IEEE Symposium on Security and Privacy (S&P)*, 2018.
31. S. Gokulanathan, A. Feldsher, A. Malca, C. Barrett, and G. Katz. Simplifying Neural Networks using Formal Verification. In *Proc. 12th NASA Formal Methods Symposium (NFM)*, pages 85–93, 2020.
32. B. Goldberger, Y. Adi, J. Keshet, and G. Katz. Minimal Modifications of Deep Neural Networks using Verification. In *Proc. 23rd Int. Conf. on Logic for Programming, Artificial Intelligence and Reasoning (LPAR)*, pages 260–278, 2020.
33. I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning.* MIT Press, 2016.
34. I. Goodfellow, J. Shlens, and C. Szegedy. Explaining and Harnessing Adversarial Examples, 2014. Technical Report. `http://arxiv.org/abs/1412.6572`.

35. D. Gopinath, G. Katz, C. Păsăreanu, and C. Barrett. DeepSafe: A Data-driven Approach for Assessing Robustness of Neural Networks. In *Proc. 16th. Int. Symposium on Automated Technology for Verification and Analysis (ATVA)*, pages 3–19, 2018.
36. D. Gunning. Explainable Artificial Intelligence (XAI), 2017. Defense Advanced Research Projects Agency (DARPA) Project.
37. X. Huang, M. Kwiatkowska, S. Wang, and M. Wu. Safety Verification of Deep Neural Networks. In *Proc. 29th Int. Conf. on Computer Aided Verification (CAV)*, pages 3–29, 2017.
38. O. Isac, C. Barrett, M. Zhang, and G. Katz. Neural Network Verification with Proof Production. In *Proc. 22nd Int. Conf. on Formal Methods in Computer-Aided Design (FMCAD)*, pages 38–48, 2022.
39. R. Ivanov, T. Carpenter, J. Weimer, R. Alur, G. Pappas, and I. Lee. Verifying the Safety of Autonomous Systems with Neural Network Controllers. *ACM Transactions on Embedded Computing Systems (TECS)*, 20(1):1–26, 2020.
40. Y. Jacoby, C. Barrett, and G. Katz. Verifying Recurrent Neural Networks using Invariant Inference. In *Proc. 18th Int. Symposium on Automated Technology for Verification and Analysis (ATVA)*, pages 57–74, 2020.
41. P. Jin, J. Tian, D. Zhi, X. Wen, and M. Zhang. Trainify: A CEGAR-Driven Training and Verification Framework for Safe Deep Reinforcement Learning. In *Proc. 34th Int. Conf. on Computer Aided Verification (CAV)*, pages 193–218, 2022.
42. A. Juliani, V. Berges, E. Teng, A. Cohen, J. Harper, C. Elion, C. Goy, Y. Gao, H. Henry, M. Mattar, et al. Unity: A General Platform for Intelligent Agents, 2018. Technical Report. `https://arxiv.org/abs/1809.02627`.
43. G. Katz, C. Barrett, D. Dill, K. Julian, and M. Kochenderfer. Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks. In *Proc. 29th Int. Conf. on Computer Aided Verification (CAV)*, pages 97–117, 2017.
44. G. Katz, C. Barrett, D. Dill, K. Julian, and M. Kochenderfer. Reluplex: a Calculus for Reasoning about Deep Neural Networks. *Formal Methods in System Design (FMSD)*, 2021.
45. G. Katz, D. Huang, D. Ibeling, K. Julian, C. Lazarus, R. Lim, P. Shah, S. Thakoor, H. Wu, A. Zeljić, D. Dill, M. Kochenderfer, and C. Barrett. The Marabou Framework for Verification and Analysis of Deep Neural Networks. In *Proc. 31st Int. Conf. on Computer Aided Verification (CAV)*, pages 443–452, 2019.
46. Y. Kazak, C. Barrett, G. Katz, and M. Schapira. Verifying Deep-RL-Driven Systems. In *Proc. 1st ACM SIGCOMM Workshop on Network Meets AI & ML (NetAI)*, pages 83–89, 2019.
47. B. Könighofer, F. Lorber, N. Jansen, and R. Bloem. Shield Synthesis for Reinforcement Learning. In *Proc. Int. Symposium on Leveraging Applications of Formal Methods, Verification and Validation (ISoLA)*, pages 290–306, 2020.
48. L. Kuper, G. Katz, J. Gottschlich, K. Julian, C. Barrett, and M. Kochenderfer. Toward Scalable Verification for Safety-Critical Deep Networks, 2018. Technical Report. `https://arxiv.org/abs/1801.05950`.
49. A. Kurakin, I. Goodfellow, and S. Bengio. Adversarial Examples in the Physical World. *Artificialc Intelligence Safety and Security*, pages 99–112, 2018.
50. O. Lahav and G. Katz. Pruning and Slicing Neural Networks using Formal Verification. In *Proc. 21st Int. Conf. on Formal Methods in Computer-Aided Design (FMCAD)*, pages 183–192, 2021.
51. Y. Li. Deep Reinforcement Learning: An Overview, 2017. Technical Report. `http://arxiv.org/abs/1701.07274`.

52. Y. Liu, J. Ding, and X. Liu. Ipo: Interior-Point Policy Optimization under Constraints. In *Proc. 34th AAAI Conf. on Artificial Intelligence (AAAI)*, pages 4940–4947, 2020.
53. A. Lomuscio and L. Maganti. An Approach to Reachability Analysis for Feed-Forward ReLU Neural Networks, 2017. Technical Report. `http://arxiv.org/abs/1706.07351`.
54. Z. Lyu, C. Y. Ko, Z. Kong, N. Wong, D. Lin, and L. Daniel. Fastened Crown: Tightened Neural Network Robustness Certificates. In *Proc. 34th AAAI Conf. on Artificial Intelligence (AAAI)*, pages 5037–5044, 2020.
55. E. Marchesini, D. Corsi, and A. Farinelli. Benchmarking Safe Deep Reinforcement Learning in Aquatic Navigation. In *Proc. IEEE/RSJ Int. Conf on Intelligent Robots and Systems (IROS)*, 2021.
56. E. Marchesini, D. Corsi, and A. Farinelli. Exploring Safer Behaviors for Deep Reinforcement Learning. In *Proc. 35th AAAI Conf. on Artificial Intelligence (AAAI)*, 2021.
57. E. Marchesini and A. Farinelli. Discrete Deep Reinforcement Learning for Mapless Navigation. In *Proc. IEEE Int. Conf. on Robotics and Automation (ICRA)*, pages 10688–10694, 2020.
58. L. Marzari, D. Corsi, E. Marchesini, and A. Farinelli. Curriculum Learning for Safe Mapless Navigation. In *Proc. 37th ACM/SIGAPP Symposium on Applied Computing (SAC)*, pages 766–769, 2022.
59. V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing Atari with Deep Reinforcement Learning, 2013. Technical Report. `https://arxiv.org/abs/1312.5602`.
60. S. M. Moosavi-Dezfooli, A. Fawzi, O. Fawzi, and P. Frossard. Universal Adversarial Perturbations. In *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 1765–1773, 2017.
61. C. Nandkumar, P. Shukla, and V. Varma. Simulation of Indoor Localization and Navigation of Turtlebot 3 using Real Time Object Detection. In *Proc. Int. Conf. on Disruptive Technologies for Multi-Disciplinary Research and Applications (CENTCON)*, 2021.
62. M. Ostrovsky, C. Barrett, and G. Katz. An Abstraction-Refinement Approach to Verifying Convolutional Neural Networks. In *Proc. 20th. Int. Symposium on Automated Technology for Verification and Analysis (ATVA)*, pages 391–396, 2022.
63. M. Pfeiffer, S. Shukla, M. Turchetta, C. Cadena, A. Krause, R. Siegwart, and J. Nieto. Reinforced Imitation: Sample Efficient Deep Reinforcement Learning for Mapless Navigation by Leveraging Prior Demonstrations. *IEEE Robotics and Automation Letters*, 3(4):4423–4430, 2018.
64. A. Pore, D. Corsi, E. Marchesini, D. Dall'Alba, A. Casals, A. Farinelli, and P. Fiorini. Safe Reinforcement Learning using Formal Verification for Tissue Retraction in Autonomous Robotic-Assisted Surgery. In *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pages 4025–4031, 2021.
65. A. Ray, J. Achiam, and D. Amodei. Benchmarking Safe Exploration in Deep Reinforcement Learning, 2019. Technical Report. `https://cdn.openai.com/safexp-short.pdf`.
66. I. Refaeli and G. Katz. Minimal Multi-Layer Modifications of Deep Neural Networks. In *Proc. 5th Workshop on Formal Methods for ML-Enabled Autonomous Systems (FoMLAS)*, pages 46–66, 2022.
67. J. Roy, R. Girgis, J. Romoff, P. Bacon, and C. Pal. Direct Behavior Specification via Constrained Reinforcement Learning, 2021. Technical Report. `https://arxiv.org/abs/2112.12228`.

68. J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal Policy Optimization Algorithms, 2017. Technical Report. http://arxiv.org/abs/1707.06347.
69. K. Simonyan and A. Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition, 2014. Technical Report. http://arxiv.org/abs/1409.1556.
70. G. Singh, T. Gehr, M. Puschel, and M. Vechev. An Abstract Domain for Certifying Neural Networks. In *Proc. 46th ACM SIGPLAN Symposium on Principles of Programming Languages (POPL)*, 2019.
71. A. Stooke, J. Achiam, and P. Abbeel. Responsive Safety in Reinforcement Learning by Pid Lagrangian Methods. In *Proc. 37th Int. Conf. on Machine Learning (ICML)*, pages 9133–9143, 2020.
72. C. Strong, H. Wu, A. Zeljić, K. Julian, G. Katz, C. Barrett, and M. Kochenderfer. Global Optimization of Objective Functions Represented by ReLU Networks. *Journal of Machine Learning*, pages 1–28, 2021.
73. X. Sun, H. Khedr, and Y. Shoukry. Formal Verification of Neural Network Controlled Autonomous Systems. In *Proc. 22nd ACM Int. Conf. on Hybrid Systems: Computation and Control (HSCC)*, 2019.
74. R. Sutton and A. Barto. *Reinforcement Learning: An Introduction*. MIT press, 2018.
75. R. Sutton, D. McAllester, S. Singh, and Y. Mansour. Policy Gradient Methods for Reinforcement Learning with Function Approximation. In *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, 1999.
76. C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing Properties of Neural Networks, 2013. Technical Report. http://arxiv.org/abs/1312.6199.
77. L. Tai, G. Paolo, and M. Liu. Virtual-to-Real Deep Reinforcement Learning: Continuous Control of Mobile Robots for Mapless Navigation. In *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pages 31–36, 2017.
78. V. Tjeng, K. Xiao, and R. Tedrake. Evaluating Robustness of Neural Networks with Mixed Integer Programming, 2017. Technical Report. http://arxiv.org/abs/1711.07356.
79. H. Van Hasselt, A. Guez, and D. Silver. Deep Reinforcement Learning with Double Q-Learning. In *Proc. 30th AAAI Conf. on Artificial Intelligence (AAAI)*, 2016.
80. M. Vasić, A. Petrović, K. Wang, M. Nikolić, R. Singh, and S. Khurshid. MoËT: Mixture of Expert Trees and its Application to Verifiable Reinforcement Learning. *Neural Networks*, 151:34–47, 2022.
81. A. Wachi and Y. Sui. Safe Reinforcement Learning in Constrained Markov Decision Processes. In *Proc. 37th Int. Conf. on Machine Learning (ICML)*, pages 9797–9806, 2020.
82. A. Wahid, A. Toshev, M. Fiser, and T. Lee. Long Range Neural Navigation Policies for the Real World. In *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pages 82–89, 2019.
83. S. Wang, K. Pei, J. Whitehouse, J. Yang, and S. Jana. Formal Security Analysis of Neural Networks using Symbolic Intervals. In *Proc. 27th USENIX Security Symposium*, pages 1599–1614, 2018.
84. H. Wu, A. Ozdemir, A. Zeljić, A. Irfan, K. Julian, D. Gopinath, S. Fouladi, G. Katz, C. Păsăreanu, and C. Barrett. Parallelization Techniques for Verifying Neural Networks. In *Proc. 20th Int. Conf. on Formal Methods in Computer-Aided Design (FMCAD)*, pages 128–137, 2020.

85. H. Wu, A. Zeljić, K. Katz, and C. Barrett. Efficient Neural Network Analysis with Sum-of-Infeasibilities. In *Proc. 28th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pages 143–163, 2022.
86. K. Yoneda, H. Tehrani, T. Ogawa, N. Hukuyama, and S. Mita. Lidar Scan Feature for Localization with Highly Precise 3-D Map. In *Proc. IEEE Intelligent Vehicles Symposium (IV)*, pages 1345–1350, 2014.
87. T. Zelazny, H. Wu, C. Barrett, and G. Katz. On Reducing Over-Approximation Errors for Neural Network Verification. In *Proc. 22nd Int. Conf. on Formal Methods in Computer-Aided Design (FMCAD)*, pages 17–26, 2022.
88. H. Zhang, M. Shinn, A. Gupta, A. Gurfinkel, N. Le, and N. Narodytska. Verification of Recurrent Neural Networks for Cognitive Tasks via Reachability Analysis. In *Proc. 24th European Conf. on Artificial Intelligence (ECAI)*, pages 1690–1697, 2020.
89. J. Zhang, J. Kim, B. O'Donoghue, and S. Boyd. Sample Efficient Reinforcement Learning with REINFORCE, 2020. Technical Report. https://arxiv.org/abs/2010.11364.
90. J. Zhang, J. Springenberg, J. Boedecker, and W. Burgard. Deep Reinforcement Learning with Successor Features for Navigation across Similar Environments. In *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2017.
91. L. Zhang, R. Zhang, T. Wu, R. Weng, M. Han, and Y. Zhao. Safe Reinforcement Learning with Stability Guarantee for Motion Planning of Autonomous Vehicles. *IEEE Transactions on Neural Networks and Learning Systems*, 32(12):5435–5444, 2021.
92. O. Zhelo, J. Zhang, L. Tai, M. Liu, and B. W. Curiosity-Driven Exploration for Mapless Navigation with Deep Reinforcement Learning, 2018. Technical Report. https://arxiv.org/abs/2212.03287.

Appendices

A Training the DRL Models

In this Appendix, we elaborate on the hyperparameters used for training, alongside various implementation details. The code is based on the *BasicRL* baselines[3]. Our full code for training, as well as our original models, can be found in our publicly-available artifact accompanying this paper [3].

General Parameters

- *episode limit*: 100,000
- *number of hidden layers*: 2
- *size of hidden layers*: 16
- *gamma (γ)*: 0.99

To facilitate our experiments, we followed [57] — and used the same network topology, i.e., two fully-connected ReLU layers, but focused on the smallest layer size that still achieved state-of-the-art results: 16 neurons, instead of 64 in [57], as these achieved similar accuracy and allowed us to expedite verification.

An additional difference, between our setting and the one appearing in [57], is that our agents have three outputs, instead of five. However, this is beneficial, as our agents achieve similar rewards to those reported in [57], and are more straightforward to verify ("design-for-verification").

DDQN Parameters

For the DDQN experiments, we rely on the Double DQN implementation (DDQN) with a "soft" update performed at each step. The network is a standard feed-forward DNN, without dueling architecture.

- *memory limit*: 5,000
- *epochs*: 40
- *batch size*: 128
- *ϵ-decay*: 0.99995
- *tau (τ)*: 0.005

Reinforce Parameters

Our implementation is based on a version of Reinforce which directly implements the policy gradient theorem [75]. The strategy for the update rule is a pure Monte Carlo approach, without temporal difference rollouts.

- *memory limit*: None
- *trajectory update frequency*: 20
- *trajectory reduction strategy*: sum

[3] `https://github.com/d-corsi/BasicRL`

PPO Parameters

For the value function estimation and the critic update, we adopted a 1-step temporal difference strategy (TD-1). The update rule follows the guidelines of the *OpenAI's Spinning Up* documentation.[4]

- *memory limit*: None
- *trajectory update frequency*: 20
- *trajectory reduction strategy*: sum
- *critic batch size*: 128
- *critic epochs*: 60
- *critic network size*: same as actor
- *PPO clip*: 0.2

Random Seeds for Reproducibility

We now present a complete list of our seeds, sorted by value:

[35, 47, 67, 68, 73, 76, 77, 81, 90, 91, 114, 128, 158, 165, 174, 176, 180, 196, 201, 215, 234, 239, 240, 267, 286, 296, 298, 299, 303, 308, 318, 319, 321, 343, 352, 379, 381, 393, 399, 418, 425, 444, 457, 491, 502, 512, 518, 528, 530, 535, 540, 549]

At each execution, we fed the same seeds to procedures from the *Random*, *NumPy* and *TensorFlow* Python modules.

[4] `https://spinningup.openai.com/en/latest/`

B Technical Specifications of the Robot

In this Appendix, we describe the details regarding the technical specifications of the robot, its sensors, and our design choices. We performed our experiments on the *Robotis Turtlebot 3*, in the *burger* version.[5] Turtlebot is a small research and development platform (138*mm* x 178*mm* x 192*mm*), that includes a set of various sensors for mapping and navigation:

– 360° lidar sensor (with a maximal distance of 3.5 m)
– Raspberry Pi 3 to control the platform
– Gyroscope, Accelerometer, and Magnetometer sensors
– (optional) Raspberry Pi Camera for perception

The manufacturer provides all the libraries and scripts for full compatibility with the standard Robotic Operating System (ROS). The robot is designed to receive continuous input (*linear* and *angular* velocity). In our settings, we discretized the action space: therefore a single step `FORWARD` corresponds to a linear translation of 5 cm, while a `LEFT/RIGHT` turn produces a rotation of 30° in the desired direction.

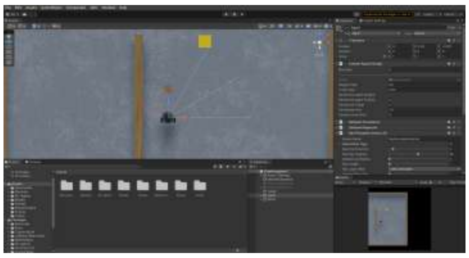


Fig. 7: The *Unity3D* engine with our simulation environment.

To perform the training of our robot, we rely on *Unity3D*, a popular engine originally designed for game development, that has recently been adopted for robotics simulation [55,64]. In particular, the built-in physics engine, the powerful 3*D* rendering algorithm and the time control system (which allows speeding up the simulation by more than 10 times), have made *Unity3D* a very powerful tool in these contexts [42]. Fig. 7 depicts an example of our Turtlebot3 environment in the *Unity3D* simulator. A key advantage of mapless navigation is that the robot can access only local observations. Thus, the robot can "see" only the distance to the nearest collision point; hence the agent's decisions are agnostic to the actual shape and size of the obstacles, and thus our system is encouraged to generalize to unseen environments during training.

[5] `https://www.robotis.us/turtlebot-3/`

C Adversarial Trajectories

In this Appendix, we depict a complete trajectory of a `RIGHT CYCLE`, as seen in Fig. 8. We note that in this, and other, infinite-loop trajectories, it is crucial to encode that the first and last states of the trajectory are identical (e.g., Subfig. *A* and Subfig. *L* in Fig. 8). Thus, given the deterministic nature of the controller, the robot will repeat the same sequence of actions, ending in an infinite loop, in which it constantly turns `RIGHT`.

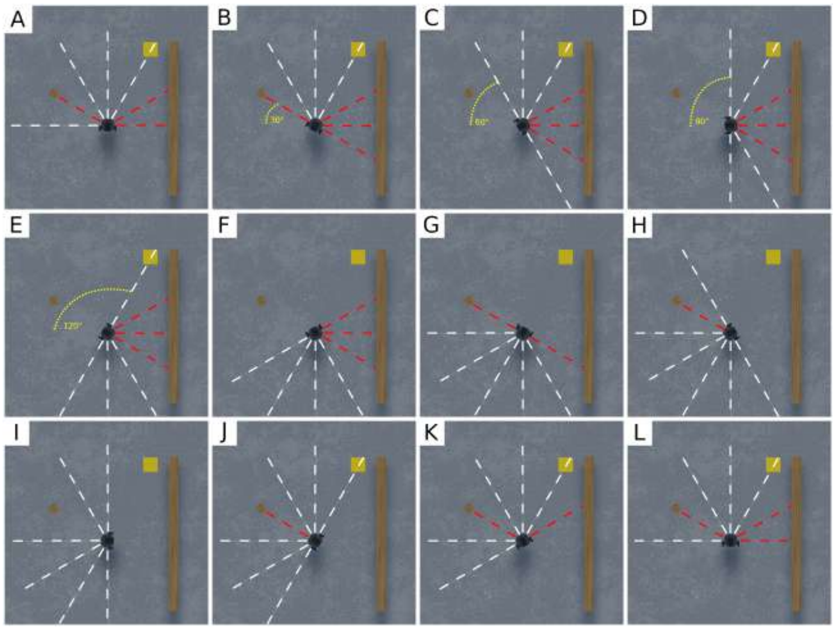


Fig. 8: A 12-step trajectory of an infinite `RIGHT CYCLE`. The white and red dashed lines represent the lidar scan (white indicates a clear path; red indicates an obstacle is present), while the yellow square represents the target position towards which the robot navigates. The yellow dotted line represents the angular step the robot performs in the first six steps. Given that the spacing between each lidar scan is the same as the angular step size (30°), at each time-step it is possible to encode the sliding window for the state. The trajectory of a `LEFT CYCLE` is symmetric.

D Encoding Verification Queries

In this Appendix, we formally define the complete list of properties we analyzed in our experiments. All values are calibrated based on the technical specifications of the robot, and the size of the discrete actions presented in this paper.

Notice that, the symbol x_n^t corresponds to the input node n of the network at time t, while y_n^t is the output node n at time t. Time (t) is omitted for the one-step properties (i.e., k=1).

Determinsitic Policies and Stochastic Policies. In order to compare DDQN to the stochastic policies (Reinforce and PPO), we first trained the stochastic policies as usual, and then treated them as deterministic. We note that in the case of stochastic policies, our approach can only detect whether a property *may* be violated, and not how likely the violation is.

Marabou. All our queries were dispatched using the sound and complete *Marabou* verification engine [45,84]. *Marabou* is a modern DNN verifier, whose core consists of a native Simplex solver, combined with abstraction and abstract-interpretation techniques [25,62,70,83], splitting heuristics [85], optimization capabilities [72], and enhanced to support varied activation functions [7] and recurrent networks [40]. *Marabou* has been previously applied to various verification-based tasks, such as network repair [32,66], network simplification [31,50], ensemble selection [8], and more [5,15,19,38,87]. The *Marabou* engine supports DNNs with ReLU layers, max-pooling, convolution, absolute value, and sign layers; and also supports sigmoids and softmax constraints.

D.1 Property Constraints

Trivial Bounds: all the inputs of the network are normalized in the range $[0, 1]$. However, in the following cases, the actual bounds can be tightened: (i) the true lower-bound for a lidar scan is 0.135, given that the lidar scan is positioned in the center of the robot, therefore the size of the robot itself constitutes a minimal distance from an obstacle; (ii) this lower bound can be further tightened to a value of 0.2 to guarantee enough space for all the possible actions (i.e., `FORWARD`, `LEFT`, `RIGHT`); (iii) for the collision properties, we found that 0.185 is the minimal distance with which the robot can make an action `FORWARD`, while avoiding a collision due to a rotation; and (iv) to avoid configurations in which reaching the target position is a trivial problem (e.g., a 1-step trajectory), we set a minimum distance from the target of 0.2.

A complete list of constraints, per property

Notice that all the properties are encoded as a negation of the expected behavior (see Sec. 4 for details). Therefore a `SAT` assignment corresponds to a violation of the required property. We note that the outputs $< y_0, y_1, y_2 >$ correspond to the actions $<$ `FORWARD`, `LEFT`, `RIGHT` $>$.

– `FORWARD COLLISION` (k=1): if the robot is closer than 185 mm to an obstacle in front, never select the action `FORWARD`. If violated, the robot collides in, at most, two steps.
  - *Precondition (P):* $x_i \in [0.2, 1]$ for $i = [0, 1, 2, 4, 5, 6] \wedge x_3 \in [0.135, 0.185] \wedge x_7 \in [0, 1] \wedge x_8 \in [0.2, 1]$
  - *Postcondition (Q):* $y_0 > y_1 \wedge y_0 > y_2$

– `LEFT COLLISION` (k=1): if the robot is closer than 185 mm to an obstacle on the front-left, never select the action `FORWARD`. If violated, the robot collides in, at most, two steps.
  - *Precondition (P):* $x_i \in [0.2, 1]$ for $i = [0, 1, 3, 4, 5, 6] \wedge x_2 \in [0.135, 0.185] \wedge x_7 \in [0, 1] \wedge x_8 \in [0.2, 1]$
  - *Postcondition (Q):* $y_0 > y_1 \wedge y_0 > y_2$

– `RIGHT COLLISION` (k=1): if the robot is closer than 185 mm to an obstacle on the front-right, never select the action `FORWARD`. If violated, the robot will collide in, at most, two steps.
  - *Precondition (P):* $x_i \in [0.2, 1]$ for $i = [0, 1, 2, 3, 5, 6] \wedge x_4 \in [0.135, 0.185] \wedge x_7 \in [0, 1] \wedge x_8 \in [0.2, 1]$
  - *Postcondition (Q):* $y_0 > y_1 \wedge y_0 > y_2$

– `ALTERNATING LOOP` (k=2): for all possible positions of the target and the obstacles, never sequentially select the actions `RIGHT` and then immediately `LEFT`. If violated, the robot gets stuck in an infinite 2-step loop.
  - *Precondition (P):* $x_i^t \in [0.2, 1]$ for $i = [0, 1, 2, 3, 4, 5, 6, 8] \wedge x_7^t \in [0, 1] \wedge x_i^{t+1} \in [0.2, 1]$ for $i = [0, 1, 2, 3, 4, 5, 6, 8] \wedge x_7^{t+1} \in [0, 1] \wedge x_8^t = x_8^{t+1} \wedge x_i^t = x_{i+1}^t$ for $i = [0, 1, 2, 3, 4, 5] \wedge x_7^{t+1} = x_7^t \pm \frac{1}{12}$
  - *Postcondition (Q):* $(y_2^t > y_0^t \wedge y_2^t > y_1^t) \wedge (y_1^{t+1} > y_0^{t+1} \wedge y_1^{t+1} > y_2^{t+1})$

  Given the symmetric nature of this property, the same encoding allowed us to check both the `LEFT`/`RIGHT` and `RIGHT`/`LEFT` alternating loops. However, we note that our encodings (arbitrarily) search for trajectories starting with a turn to the `RIGHT`.

– `LEFT CYCLE` and `RIGHT CYCLE` (k=12): for all possible positions of the target and the obstacles, if the robot has at least one escape direction, never select in 12 consecutive steps the action `LEFT` (or `RIGHT`). If this property is violated, the robot will get stuck in an infinite loop of 360° rotating counter-clockwise (or clockwise).
  - *Precondition - bounds (P):* $x_i^t \in [0.2, 1]$ for $i = [0, 1, 2, 3, 4, 5, 6, 8]$ and $t = [0, ..., 11] \wedge x_7^t \in [0, 1]$ for $t = [0, ..., 11] \wedge x_8^t = x_8^{t+1}$ for $t = [0, ..., 10]$
  - *Precondition - sliding window (P):* to better explain the property, we refer to Fig. 9 (for `RIGHT CYCLE` the figure should be read from top to bottom; the opposite for `LEFT CYCLE`), which includes a scheme of the sliding window encoding, as explained in Sec. 4. We note that sensors (in different time-steps) are colored the same to represent that they encode the same equality constraint (e.g., $\langle L_1, T_0\rangle = \langle L_0, T_1\rangle$). The input relative to the distance from the target (i.e., x_8^t) must be the same for all the 12 steps, while the angle (i.e., x_7^t) must respect the property $x_7^{t+1} = x_7^t \pm \frac{1}{12}$ (depending on whether `RIGHT CYCLE` or `LEFT CYCLE` is encoded).

- *Postcondition (Q):* choose the same action (`LEFT` or `RIGHT`) for 12 consecutive steps.

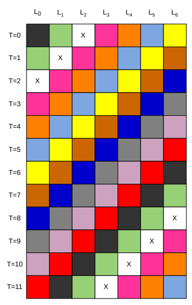


Fig. 9: A visual scheme of the "sliding window" constraints for encoding a 12-step full cycle. In the case of encoding a `RIGHT CYCLE`, the constraints should be read from top to bottom. In the case of encoding a `LEFT CYCLE`, the constraints should be read from bottom to top.

Note. The fact that the agent may enter infinite loops is not due to the turning angle matching the angle between consecutive lidar sensors; the same can happen when different angles are used, as in [57]. Configuring the lidar angles to match the turning angle is part of our methodology for facilitating verification ("sliding window", for reducing the state space), but it does not cause the infinite loop, or any other property violation.

D.2 The Slack Margin

When searching for adversarial inputs with formal verification techniques, it is common practice to search for *strong* violations, i.e., requiring a minimal difference between the original output and the output generated by the adversarial input. Formally, if the original output is y, we consider a violation only if $y' > y + \gamma$, for some predefined margin ("slack") $\gamma > 0$, and an output y'.

In our experiments, to conduct a fair comparison between models trained by different algorithms (all outputting values in different ranges) we analyzed the empirical distribution of a random variable that consists of the difference between the "winner" (classified) output and the "runner-up" (second highest) output.

As can be seen in Fig. 10, the empirical distribution for this random variable is different (both in the shape, and values) per each of the training algorithms.

Given these results, to guarantee a fair comparison between the algorithms, we used a slack variable with a value corresponding to the *75-th percentile*, i.e., a value that is larger than 75% of the samples. The values matching this percentile (per each training algorithm) are:

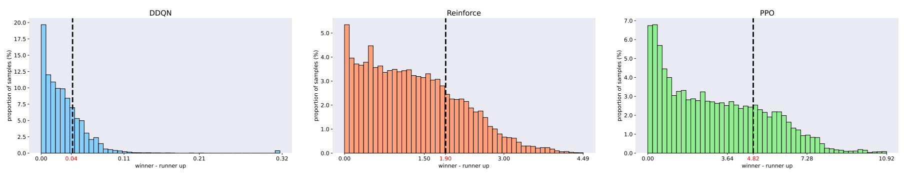


Fig. 10: The empirical distribution of the difference between the “winning” (highest) output and the second-highest output. Each distribution is based on approximately 4000 samples, taken from $N = 100$ episodes for each algorithm.

– **DDQN**: 0.042
– **Reinforce**: 1.904
– **PPO**: 4.821

These values were chosen to be the γ slack values encoded for the single-step ($k = 1$) verification queries, i.e., queries encoding: `FORWARD COLLISION`, `LEFT COLLISION` or `RIGHT COLLISION`.

When encoding multistep properties (the three loop types) this slack was divided by the number of k steps encoded in the relevant query. This is because each additional step adds constraints to the query and so to balance the additional constraints — the γ slack is reduced.

For the original verification query encodings, we refer the reader to our publicly-available artifact accompanying this paper [3].

D.3 Bravery Property Results

Our verification results indicate that:

1. the braver model may move forward when the distance is slightly over 0.42 meters; while
2. the over-conservative model never moves forward in cases where a similar obstacle is closer than 0.88 meters.

E Comparison to Gradient-Based Methods

We also compared our verification-based method to state-of-the-art gradient attacks, which can also be used to search for property violations — and hence, in model selection. Gradient attacks are optimization methods, designed to produce inputs that are misclassified by the DNN [76]. Intuitively, starting from some arbitrary input point, these methods seek perturbations that cause significant changes to the model's outputs — increasing the value it assigns to outputs that correspond to some (targeted) incorrect label. These perturbations are discovered using a local search that performs gradient descent on a tailored loss function [34]. It is common practice to consider a gradient-based search as successful if it finds a perturbed input on which the target label receives a score that is higher than the true label (by some predefined margin). Notice that, in contrast to complete verification methods, a gradient-based attack can only answer `SAT` or `TIMEOUT`.

In our evaluation, we ran the Basic Iterative Method (BIM) [49] — a popular gradient attack based on the Fast-Gradient Sign Method (FGSM) [27,34]. We ran the attack on all 780 models, searching for violations of all three single-step safety properties. Each attack ran for 40 iterations, with a step size of 0.01. In order to conduct a fair comparison between this gradient-based attack and our verification-based approach, we used a heuristic of choosing the initial point for the attack to be the *center* of the bounds for each input. In addition, we used the same slack margin by which the incorrect label should pass the correct one in all experiments, for both approaches. For additional details see Section D of the Appendix.

Although gradient-based methods are extremely scalable, they suffer from many setbacks. Firstly, they tend to miss many violations otherwise found by verification-driven approaches (up to a third of our violations were missed, as seen in Table 2). In addition, gradient-based methods are incomplete, and thus, are unable to guarantee that some properties always *hold*, while sound and complete verification tools may guarantee a property holds in various settings. Another pitfall of gradient-based methods is their inadequacy to check for adversarial inputs across multiple steps (loops, in our case), bringing us to focus solely on collision properties in our comparison. Hence, although gradient approaches

`Algorithm`	`Gradient`	`Verification`	`Ratio (%)`
DDQN	405	765	53
Reinforce	671	761	88
PPO	345	600	58
`Total`	1421	2126	67

Table 2: The number of counterexamples (`SAT` results) discovered using the gradient-based BIM method [49], versus our verification-based approach. The rightmost column indicates the ratio of values in the two previous columns.

are ill-suited for the task described, we compared them to our approach, despite the inherent limitations of such a comparison. Still, gradient attacks are the best tool previously available, and so we believe this highlights the usefulness of our approach.

Semantic Abstraction of Neural Networks

Calvin Chau, Jan Křetínský, and Stefanie Mohr

Technical University of Munich, Germany
calvin.chau@tum.de

Keywords: Neural networks · Abstraction · Formal verification · Explainability

1 Extended Abstract

Neural networks have gained a lot of popularity in recent years, due to their ability to accurately solve a wide range of prediction tasks. In safety-critical settings, such as autonomous driving [Che+17; Gri+19], it is important to ensure the correctness of neural networks. To this end, different techniques can be used, e.g. abstract interpretation [Sin+19] or SMT-solving [Kat+19], which however often do not scale to networks of practical sizes.

In this talk, we present a generic abstraction approach using the previously introduced *semantics* for neural networks [Ash+20]. Based on the semantics, we apply linear programming and orthogonal projection to carefully replace neurons by linear combinations of neurons. We then outline how our abstraction could help scale existing verification procedures. Additionally, we discuss how our approach can be used to reveal redundancies in the networks, possibly leading to more informed network architecture choices. Finally, we describe the refinement of our abstraction, allowing for the application of a counter-example guided abstraction refinement (CEGAR) scheme [Cla+00].

References

[Ash+20] Pranav Ashok et al. "DeepAbstract: Neural Network Abstraction for Accelerating Verification". In: Hanoi, Vietnam: Springer-Verlag, 2020, 92–107. ISBN: 978-3-030-59151-9. DOI: 10.1007/978-3-030-59152-6_5.

[Che+17] Xiaozhi Chen et al. "Multi-view 3D Object Detection Network for Autonomous Driving". In: July 2017, pp. 6526–6534. DOI: 10.1109/CVPR.2017.691.

[Cla+00] Edmund Clarke et al. "Counterexample-Guided Abstraction Refinement". In: *Computer Aided Verification.* Ed. by E. Allen Emerson and Aravinda Prasad Sistla. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 154–169. ISBN: 978-3-540-45047-4.

[Gri+19] Sorin Grigorescu et al. "A survey of deep learning techniques for autonomous driving". In: *Journal of Field Robotics* 37 (Nov. 2019). DOI: 10.1002/rob.21918.

[Kat+19] Guy Katz et al. “The Marabou Framework for Verification and Analysis of Deep Neural Networks”. In: *Computer Aided Verification*. Ed. by Isil Dillig and Serdar Tasiran. Cham: Springer International Publishing, 2019, pp. 443–452. ISBN: 978-3-030-25540-4.

[Sin+19] Gagandeep Singh et al. “An Abstract Domain for Certifying Neural Networks”. In: *Proc. ACM Program. Lang.* 3.POPL (Jan. 2019). DOI: `10.1145/3290354`. URL: `https://doi.org/10.1145/3290354`.

Towards Formal XAI: Formally Approximate Minimal Explanations of Neural Networks

Shahaf Bassan and Guy Katz

The Hebrew University of Jerusalem, Jerusalem, Israel
{shahaf.bassan, g.katz}@mail.huji.ac.il

Abstract. With the rapid growth of machine learning, deep neural networks (DNNs) are now being used in numerous domains. Unfortunately, DNNs are "black-boxes", and cannot be interpreted by humans, which is a substantial concern in safety-critical systems. To mitigate this issue, researchers have begun working on explainable AI (XAI) methods, which can identify a subset of input features that are the cause of a DNN's decision for a given input. Most existing techniques are heuristic, and cannot guarantee the correctness of the explanation provided. In contrast, recent and exciting attempts have shown that formal methods can be used to generate provably correct explanations. Although these methods are sound, the computational complexity of the underlying verification problem limits their scalability; and the explanations they produce might sometimes be overly complex. Here, we propose a novel approach to tackle these limitations. We (i) suggest an efficient, verification-based method for finding *minimal explanations*, which constitute a *provable approximation* of the global, minimum explanation; (ii) show how DNN verification can assist in calculating lower and upper bounds on the optimal explanation; (iii) propose heuristics that significantly improve the scalability of the verification process; and (iv) suggest the use of *bundles*, which allows us to arrive at more succinct and interpretable explanations. Our evaluation shows that our approach significantly outperforms state-of-the-art techniques, and produces explanations that are more useful to humans. We thus regard this work as a step toward leveraging verification technology in producing DNNs that are more reliable and comprehensible.

1 Introduction

Machine learning (ML) is a rapidly growing field with a wide range of applications, including safety-critical, high-risk systems in the fields of health care [19], aviation [39] and autonomous driving [12]. Despite their success, ML models, and especially deep neural networks (DNNs), remain "black-boxes" — they are incomprehensible to humans and are prone to unexpected behaviour and errors. This issue can result in major catastrophes [13,74], and also in poor decision-making due to brittleness or bias [8,25].

In order to render DNNs more comprehensible to humans, researchers have been working on *explainable AI* (*XAI*), where we seek to construct models for explaining and interpreting the decisions of DNNs [51,56–58]. Work to date has focused on heuristic approaches, which provide explanations, but do not provide guarantees about the correctness or succinctness of these explanations [14,33,45]. Although these approaches are an important step, their limitations might result in skewed results, possibly failing to meet the regulatory guidelines of institutions and organizations such as the European Union, the US government, and the OECD [52]. Thus, producing DNN explanations that are provably accurate remains of utmost importance.

More recently, the formal verification community has proposed approaches for providing formal and rigorous explanations for DNN decision making [28,32,52,60]. Many of these approaches rely on the recent and rapid developments in DNN verification [1,9,10,40]. These approaches typically produce an *abductive explanation* (also known as a *prime implicant*, or *PI-explanation*) [32,59,60]: a minimum subset of input features, which by themselves already determine the classification produced by the DNN, regardless of any other input features. These explanations afford formal guarantees, and can be computed via DNN verification [32].

Abductive explanations are highly useful, but there are two major difficulties in computing them. First, there is the issue of scalability: computing locally minimal explanations might require a polynomial number of costly invocations of the underlying DNN verifier, and computing a globally minimal explanation is even more challenging [**?**,32,49]. The second difficulty is that users may sometimes prefer "high-level" explanations, not based solely on input features, as these may be easier to grasp and interpret compared to "low-level", complex, feature-based explanations.

To tackle the first difficulty, we propose here new approaches for more efficiently producing verification-based abductive explanations. More concretely, we propose a method for *provably approximating* minimum explanations, allowing stakeholders to use slightly larger explanations that can be discovered much more quickly. To accomplish this, we leverage the recently discovered dual relationship between explanations and contrastive examples [31]; and also take advantage of the sensitivity of DNNs to small adversarial perturbations [65], to compute both lower and upper bounds for the minimum explanation. In addition, we propose novel heuristics for significantly expediting the underlying verification process.

In addressing the second difficulty, i.e. the interpretability limitations of "low-level" explanations, we propose to construct explanations in terms of *bundles*, which are sets of related features. We empirically show that using our method to produce bundle explanations can significantly improve the interpretability of the results, and even the scalability of the approach, while still maintaining the soundness of the resulting explanations.

To summarize, our contributions include the following: (i) We are the first to suggest a method that formally produces sound and minimal abductive explanations that *provably approximate* the global-minimum explanation. (ii) Our three suggested novel heuristics expedite the search for minimal abductive explanations, significantly outperforming the state of the art. (iii) We suggest a novel approach for using bundles to efficiently produce sound and provable explanations that are more interpretable and succinct.

For evaluation purposes, we implemented our approach as a proof-of-concept tool. Although our method can be applied to any ML model, we focused here on DNNs, where the verification process is known to be NP-complete [40], and the scalable generation of explanations is known to be challenging [32,59]. We used our tool to test the approach on DNNs trained for digit and clothing classification, and also compared it to state-of-the-art approaches [32,33]. Our results indicate that our approach was successful in quickly producing meaningful explanations, often running 40% faster than existing tools. We believe that these promising results showcase the potential of this line of work.

The rest of the paper is organized as follows. Sec. 2 contains background on DNNs and their verification, as well as on formal, minimal explanations. Sec. 3 covers the main method for calculating approximations of minimum explanations, and Sec. 4 covers methods for improving the efficiency of calculating these approximations. Sec. 5 covers the use of *bundles* in constructing "high-level", provable explanations. Next, we present our evaluation in Sec. 6. Related work is covered in Sec. 7, and we conclude in Sec. 8.

2 Background

DNNs. A deep neural network (DNN) [47] is a directed graph composed of layers of nodes, commonly called *neurons*. In feed-forward NNs the data flows from the first (*input*) layer, through intermediate (*hidden*) layers, and onto an *output* layer. A DNN's output is calculated by assigning values to its input neurons, and then iteratively calculating the values of neurons in subsequent layers. In the case of *classification*, which is the focus of this paper, each output neuron corresponds to a specific *class*, and the output neuron with the highest value corresponds to the class the input is classified to.

Fig. 1 depicts a simple, feed-forward DNN. The input layer includes three neurons, followed by a weighted sum layer, which calculates an affine transformation of values from the input layer. Given the input $V_1 = [1,1,1]^T$, the second layers computes the values $V_2 = [6,9,11]^T$. Next comes a ReLU layer, which computes the function $\text{ReLU}(x) = \max(0,x)$ for each neuron in the preceding layer, resulting in $V_3 = [6,9,11]^T$. The final (output) layer then computes an affine transformation, resulting in $V_4 = [15,-4]^T$. This indicates that input $V_1 = [1,1,1]^T$ is classified as the category corresponding to the first output neuron, which is assigned the greater value.

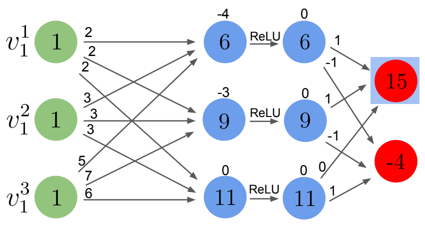


Fig. 1: A simple DNN.

DNN Verification. A DNN verification query is a tuple $\langle P, N, Q\rangle$, where N is a DNN that maps an input vector x to an output vector $y = N(x)$, P is a predicate on x, and Q is a predicate on y. A DNN verifier needs to decide whether there exists an input x_0 that satisfies $P(x_0) \wedge Q(N(x_0))$ (the SAT case) or not (the UNSAT case). Typically, P and Q are expressed in the logic of real arithmetic [50]. The DNN verification problem is known to be NP-Complete [40].

Formal Explanations. We focus here on explanations for classification problems, where a model is trained to predict a label for each given input. A classification problem is a tuple $\langle F, D, K, N\rangle$ where (i) $F = \{1, ..., m\}$ denotes the features; (ii) $D = \{D_1, D_2 ..., D_m\}$ denotes the domains of each of the features, i.e. the possible values that each feature can take. The entire feature (input) space is hence $\mathbb{F} = D_1 \times D_2 \times ... \times D_m$; (iii) $K = \{c_1, c_2, ..., c_n\}$ is a set of classes, i.e. the possible labels; and (iv) $N : F \to K$ is a (non-constant) classification function (in our case, a neural network). A classification instance is the pair (v, c), where $v \in \mathbb{F}$, $c \in K$, and $c = N(v)$. In other words, v is mapped by the neural network N to class c.

Looking at (v, c), we often wish to know why v was classified as c. Informally, an *explanation* is a subset of features $E \subseteq F$, such that assigning these features to the values assigned to them in v already determines that the input will be classified as c, regardless of the remaining features $F \setminus E$. In other words, even if the values that are *not* in the explanation are changed arbitrarily, the classification remains the same. More formally, given input $v = (v_1, ...v_m) \in \mathbb{F}$ with the classification $N(v) = c$, an explanation (sometimes referred to as an *abductive explanation*, or an *AXP*) is a subset of the features $E \subseteq F$, such that:

$$\forall(x \in \mathbb{F}). \quad [\bigwedge_{i \in E}(x_i = v_i) \to (N(x) = c)] \tag{1}$$

We continue with the running example from Fig. 1. For simplicity, we assume that each input neuron can only be assigned the values 0 or 1. It can be observed that for input $V_1 = [1, 1, 1]^T$, the set $\{v_1^1, v_1^2\}$ is an explanation; indeed, once the first two entries in V_1 are set to 1, the classification remains the same for any value of the third entry (see Fig. 2). We can prove this by encoding a verification query $\langle P, N, Q\rangle = \langle E = v, N, Q_{\neg c}\rangle$, where E is the candidate explanation, and $E = v$ means that we restrict the features in E to their values in v; and $Q_{\neg c}$ implies that the classification is not c. An UNSAT result for this query indicates that E is an explanation for instance (v, c).

Clearly, the set of all features constitutes a trivial explanation. However, we are interested in *smaller* explanation subsets, which can provide useful information regarding the decision of the classifier. More precisely, we search for *minimal explanations* and *minimum explanations*. A subset $E \subseteq F$ is a *minimal explanation* (also referred to as a *local-minimal explanation*, or a *subset-minimal explanation*) of instance (v, c) if it is an explanation that ceases to be an explanation if even a single feature is removed from it:

$$\begin{aligned}&(\forall(x \in \mathbb{F}).[\wedge_{i \in E}(x_i = v_i) \to (N(x) = c)]) \wedge \\ &(\forall(j \in E).[\exists(y \in \mathbb{F}).[\wedge_{i \in E \setminus j}(y_i = v_i) \wedge (N(y) \neq c)]])\end{aligned} \tag{2}$$

Fig. 3 demonstrates that $\{v_1^1, v_1^2\}$ is a minimal explanation in our running example: removing any of its features allows mis-classification.

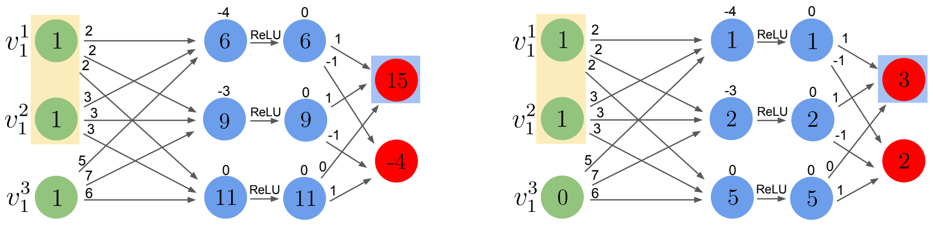


Fig. 2: $\{v_1^1, v_1^2\}$ is an explanation for input $V_1 = [1,1,1]^T$

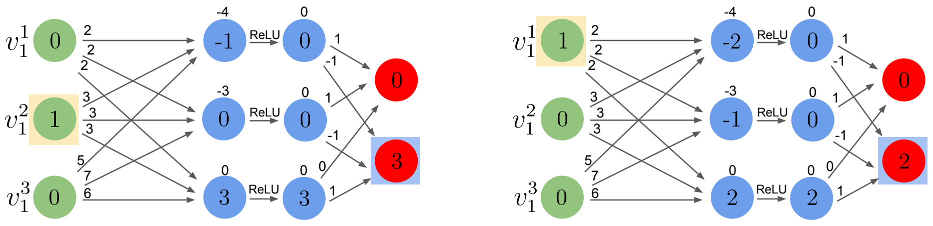


Fig. 3: $\{v_1^1, v_1^2\}$ is a minimal explanation for input $V_1 = [1,1,1]^T$.

A *minimum explanation* (sometimes referred to as a *cardinal minimal explanation* or a *PI-explanation*) is defined as a minimal explanation of minimum size; i.e., if E is a minimum explanation, then there does not exist a minimal explanation $E' \neq E$ such that $|E'| < |E|$. Fig. 4 demonstrates that $\{v_1^3\}$ is a minimum explanation for input $V_1 = [1,1,1]^T$.

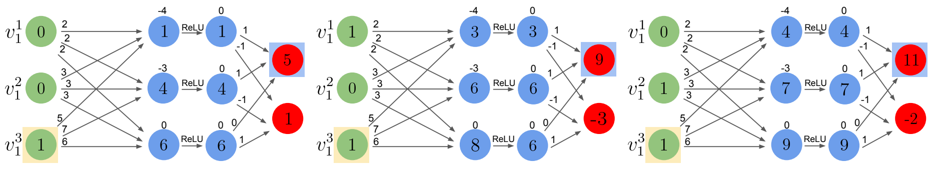


Fig. 4: $\{v_1^3\}$ is a minimum explanation for input $V_1 = [1,1,1]^T$.

Contrastive Example. A subset of features $C \subseteq F$ is called a *contrastive example* or a *contrastive explanation (CXP)* if altering the features in C is sufficient to cause the misclassification of a given classification instance (v, c):

$$\exists(x \in \mathbb{F}).[\wedge_{i \in F \setminus C}(x_i = v_i) \wedge (N(x) \neq c)] \tag{3}$$

A contrastive example for our running example is shown in Fig. 5. Notice that the question of whether a set is a contrastive example can be encoded into a verification query $\langle P, N, Q \rangle = \langle (F \setminus C) = v, N, Q_{\neg c} \rangle$, where a `SAT` result indicates that C is a contrastive example. As with explanations, smaller contrastive examples are more valuable than large ones. One useful notion is that of a *contrastive singleton*: a contrastive example of size one. A contrastive singleton could represent a specific pixel in an image, the alteration of which could result in misclassification. Such singletons are leveraged in "one-pixel attacks" [65] (see Fig. 16 in the appendix). Contrastive singletons have the following important property:

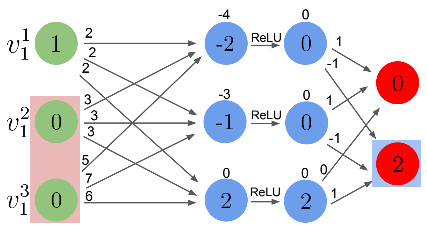


Fig. 5: $\{v_1^2, v_1^3\}$ is a contrastive example for $V_1 = [1, 1, 1]^T$.

Lemma 1. *Every contrastive singleton is contained in all explanations.*

The proof appears in Sec. A of the appendix. Lemma 1 implies that each contrastive singleton is contained in all minimal/minimum explanations.

We consider also the notion of a *contrastive pair*, which is a contrastive example of size 2. Clearly, for any pair of features (u, v) where u or v are contrastive singletons, (u, v) is a contrastive pair; however, when we next refer to contrastive pairs, we consider only pairs that *do not* contain any contrastive singletons. Likewise, for every $k > 2$, we can consider contrastive examples of size k, and we exclude from these any contrastive examples of sizes $1, \ldots, k-1$ as subsets.

We state the following theorem, whose proof also appears in Sec. A of the appendix:

Lemma 2. *All explanations contain at least one element of every contrastive pair.*

The theorem can be generalized to any $k > 2$; and can be used in showing that the *minimum hitting set (MHS)* of all contrastive examples is exactly the minimum explanation [30,55] (see Sec. B of the appendix). Further, the theorem implies a duality between contrastive examples and explanations [31,35]: a minimal hitting set of all contrastive examples constitutes a minimal explanation, and a minimal hitting set of all explanations constitutes a minimal contrastive example.

3 Provable Approximations for Minimal Explanations

State-of-the-art approaches for finding minimum explanations exploit the MHS duality between explanations and contrastive examples [32]. The idea is to iteratively compute contrastive examples, and then use their MHS as an under-approximation for the minimum explanation. Finding this MHS is an NP-complete problem, and is difficult in practice as the number of contrastive examples increases [21]; and although the MHS can be approximated using maximum satisfiability (MaxSAT) or mixed integer linear programming (MILP) solvers [27,48], existing approaches tackle simpler ML models, such as decision trees [34,37], but face scalability limitations when applied to DNNs [32,59]. Further, enumerating all contrastive examples may in itself take exponential time. Finally, recall that DNN verification is an NP-Complete problem [40]; and so dispatching a verification query to identify each explanation or contrastive example is also very slow, when the feature space is large. Finding *minimal* explanations may be easier [32], but may converge to larger and less meaningful explanations, while still requiring a linear number of calls to the underlying verifier. Our approach, described next, seeks to mitigate these difficulties.

Our overall approach is described in Algorithm 1. It is comprised of two separate threads, intended to be run in parallel. The *upper bounding thread* (T_{UB}) is responsible for computing a minimal explanation. It starts with the entire feature space, and then gradually reduces it, until converging to a minimal explanation. The size of the presently smallest explanation is regarded as an upper bound (UB) for the size of the minimum explanation. Symmetrically, the *lower bounding thread* (T_{LB}) attempts to construct small contrastive sets, used for computing a lower bound (LB) on the size of the minimum explanation. Together, these two bounds allow us to compute the approximation ratio between the minimal explanation that we have discovered and the minimum explanation. For instance, given a minimal explanation of size 7 and a lower bound of size 5, we can deduce that our explanation is at most $\frac{\text{UB}}{\text{LB}} = \frac{7}{5}$ times larger than the minimum. The two threads share global variables that indicate the set of contrastive singletons (Singletons), the set of contrastive pairs (Pairs), the upper and lower bounds (UB, LB), and the set of features that were determined not to participate in the explanation and are "free" to be set to any value (Free). The output of our algorithm is a minimal explanation (F∖Free), and the approximation ratio ($\frac{\text{UB}}{\text{LB}}$). We next discuss each of the two threads in detail.

Algorithm 1 Minimal Explanation Search

```
Input N (Neural network), F (features), v (input values), c (class prediction)
1: Singletons, Pairs, Free ← ∅, UB ← |F|, LB ← 0          ▷ Global variables
2: Launch thread $T_{\text{UB}}$
3: Launch thread $T_{\text{LB}}$
4: return F∖Free, $\frac{\text{UB}}{\text{LB}}$
```

The Upper Bounding Thread (T_{UB}). This thread, whose pseudocode appears in Algorithm 2, follows the framework proposed by Ignatiev et al. [32]: it seeks a minimal explanation by starting with the entire feature space, and then iteratively attempting to remove individual features. If removing a feature allows misclassification, we keep it as part of the explanation; otherwise, we remove it and continue. This process issues a single verification query for each feature, until converging to a minimal explanation (lines 2–8). Although this naïve search is guaranteed to converge to a minimal explanation, it needs not to converge to a *minimum* explanation; and so we apply a more sophisticated ordering scheme, similar to the one proposed by [33], where we use some heuristic model as a way for assigning weights of importance to each input feature. We then check the "least important" input features first, since freeing them has a lower chance of causing a misclassification, and they are consequently more likely to be successfully removed. We then continue iterating over features in ascending order of importance, hopefully producing small explanations.

Algorithm 2 T_{UB}: Upper Bounding Thread

```
1: Use a heuristic model to sort $F$'s features by ascending relevance
2: for each $f \in F$ do
3:     Explanation ← F∖Free
4:     if Verify((Explanation∖{f})=v,N,$Q_{\neg c}$) is UNSAT then
5:         Free ← Free ∪ {f}
6:         UB ← UB − 1
7:     end if
8: end for
```

The Lower Bounding Thread (T_{LB}). The pseudocode for the lower bounding thread (T_{LB}) appears in Algorithm 3. In lines 1–6, the thread searches for contrastive singletons. Neural networks were shown to be very sensitive to adversarial attacks [26] — slight input perturbations that cause misclassification (e.g., the aforementioned one-pixel attack [65]) — and this suggests that contrastive sets, and in particular contrastive singletons, exist in many cases. We observe that identifying contrastive singletons is computationally cheap: by encoding Eq. 3 as a verification query, once for each feature, we can discover all singletons; and in these queries all features but one are fixed, which empirically allows verifiers to dispatch them quickly.

The rest of T_{LB} (lines 9–13) performs a similar process, but with contrastive pairs (which do not contain contrastive singletons as one of their features). We use verification queries to identify all such pairs, and then attempt to find their MHS. We observe that finding the MHS of all contrastive pairs is the 2-MHS problem, which is a reformalization of the *minimum vertex cover* problem (see Sec. B of the appendix). Since this is an easier problem than the general MHS problem, solving it with MAX-SAT or MILP often converges quickly. In addition, the minimum vertex cover algorithm has a linear 2-approximating greedy algorithm, which can be used for finding a lower bound in cases of large feature spaces.

More formally, T_{LB} performs an efficient computation of the following bound:

$$\text{LB} = |\text{Singletons}| + |\text{MVC}(\text{Pairs})| \leq \text{MHS}(\text{Cxps}) = E_M \tag{4}$$

where MVC is the minimum vertex cover, Cxps denotes the set of all contrastive examples, and E_M is the size of the minimum explanation.

Algorithm 3 T_{LB}: Lower Bounding Thread

```
1:  for each $f \in F$ do                                   ▷ Find all singletons
2:      if Verify((F∖{f}=v,N,$Q_{\neg c}$) is SAT then
3:          Singletons ← Singletons ∪ {$f$}
4:          LB ← LB +1
5:      end if
6:  end for
7:
8:  AllPairs ← Distinct pairs of F∖Singletons
9:  for each (a,b) ∈ AllPairs do                             ▷ Find all pairs
10:     if Verify((F∖{a,b}=v,N,$Q_{\neg c}$) is SAT then
11:         Pairs ←Pairs ∪ {$(a,b)$}
12:     end if
13: end for
14: LB ← LB + MVC(Pairs)
```

It is worth mentioning that this approach can be extended to use contrastive examples of larger sizes ($k = 3, 4, \ldots$), as specified in Sec. C of the appendix. The fact that small contrastive examples, such as singletons, exist in large, state-of-the-art DNNs with large inputs [22,65] suggests that useful approximations exist in large DNNs. In our experiments, we observed that using only singletons and pairs affords good approximations, without incurring overly expensive computations by the underlying verifier.

4 Finding Minimal Explanations Efficiently

Algorithm 1 is the backbone of our approach, but it suffers from limited scalability — particularly, in T_{UB}. As the execution of T_{UB} progresses, and as additional features are "freed", the quickly growing search space slows down the underlying verifier. Here we propose three different methods for expediting this process, by reducing the number of verification queries required.

Method 1: Using Information from T_{LB}. We suggest to leverage the contrastive examples found by T_{LB} to expedite T_{UB}. The process is described in Algorithm 4. In line 3, T_{LB} is queried for the current set of contrastive singletons, which we know must be part of any minimal explanation. These are subtracted from the RemainingFeatures set (features left for T_{UB} to query), and consequently will not be added to the Free set — i.e., they are marked as part of the current explanation. In addition, for any contrastive pair (a, b) found by T_{LB}, either a or b must appear in any minimal explanation; and so, our algorithm skips checking the case where both a and b are removed from F (Line 8). (the method could also be extended to contrastive sets of greater cardinality.)

Method 2: Binary Search. Sorting the features being considered in ascending order of importance can have a significant effect on the size of the explanation found by Algorithm 2. Intuitively, a "perfect" heuristic model would assign the

Algorithm 4 T_{UB} using information from T_{LB}

```
1: Use a heuristic model to sort $F$ by ascending relevance
2: RemainingFeatures ← F∖Singletons
3: for each f ∈ RemainingFeatures do
4:     Explanation ← F∖Free
5:     if Verify((Explanation∖{f})=v,N,$Q_{\neg c}$) is UNSAT then
6:         Free ← Free ∪ {f}
7:         UB ← UB − 1
8:         Delete all features in a pair with f from RemainingFeatures
9:     end if
10: end for
```

greatest weights to all features in the minimum explanation, and so traversing features in ascending order would first discover all the features that can be removed (`UNSAT` verification queries), followed by all the features that belong in the explanation (`SAT` queries). In this case, a sequential traversal of the features in ascending order is quite wasteful, and it is much better to perform a binary search to find the point where the answer flips from `UNSAT` to `SAT`.

Of course, in practice, the heuristic models are not perfect, leading to potential cases with multiple "flips" from `SAT` to `UNSAT`, and vice versa. Still, if the heuristic is good in practice (which is often the case; see Sec. 6), these flips are scarce. Thus, we propose to perform multiple binary searches, each time identifying one `SAT` query (i.e., a feature added to the explanation). Observe that each time we hit an `UNSAT` query, this indicates that all the queries for features with lower priorities would also yield `UNSAT` — because if "freeing" multiple features cannot change the classification, changing fewer features certainly cannot. Thus, we are guaranteed to find the first `SAT` query in each iteration, and soundness is maintained. This process is described in Algorithm. 6 and in Fig.14 in the appendix.

Method 3: Local-Singleton Search. Let N be a DNN, and let x be an input point whose classification we seek to explain. As part of Algorithm 2, T_{UB} iteratively "frees" certain input features, allowing them to take arbitrary values, as it continues to search for features that must be included in the explanation. The increasing number of free features enlarges the search space that the underlying verifier must traverse, thus slowing down verification. We propose to leverage the hypothesis that input points nearby x that are misclassified tend to be clustered; and so, it is beneficial to *fix* the free features to "bad" values, as opposed to letting them take on arbitrary values. We speculate that this will allow the verifier to discover satisfying assignments much more quickly.

This enhancement is shown in Algorithm 5. Given a set Free of features that were previously freed, we fix their values according to some satisfying assignment previously discovered. Thus, the verification of any new feature that we consider is similar to the case of searching for contrastive singletons, which, as we already know, is fairly fast. See Fig. 15 in the appendix for an illustration. The process can be improved further by fixing the freed features to small neighborhoods of the previously discovered satisfying assignment (instead of its exact values), to allow some flexibility while still keeping the query's search space small.

Algorithm 5 T_{UB} using local-singleton search

```
 1: Use a heuristic model to sort $F$ by ascending relevance
 2: RemainingFeatures ← F∖Singletons
 3: for each f ∈ RemainingFeatures do
 4:     Explanation ← F∖Free
 5:     if Verify((Explanation∖{f})=v,N,$Q_{\neg c}$) is UNSAT then
 6:         Free ← Free ∪ {f}
 7:         UB ← UB − 1
 8:     else
 9:         Extract counter example C
10:         LocalSingletons ← ∅
11:         for each $f'$ ∈ RemainingFeatures do
12:             if Verify(Explanation∖$\{f'\} = C$,N,$Q_{\neg c}$) is SAT then
13:                 LocalSingletons ← LocalSingletons ∪ $\{f'\}$
14:             end if
15:         end for
16:         RemainingFeatures ← RemainingFeatures ∖ LocalSingletons
17:     end if
18: end for
```

5 Minimal Bundle Explanations

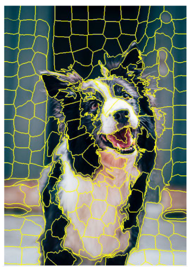


Fig. 6: Partition input's features into bundles.

So far, we presented methods for generating explanations within a given approximation ratio of the minimum explanation (Sec. 3), and for expediting the computation of these explanations (Sec. 4) — in order to improve the scalability of our explanation generation mechanism. Next, we seek to tackle the second challenge from Sec. 1, namely that these explanations may be too low-level for many users. To address this challenge, we focus on *bundles*, which is a topic well covered in the ML [64] and heuristic XAI literature [51,56] (commonly known as "super-pixels" for computer-vision tasks). Intuitively, bundles are a partitioning of the features into disjoint sets (an illustration appears in Fig. 6). The idea, which we later validate empirically, is that providing explanations in terms of bundles is often easier for humans to comprehend. As an added bonus, using bundles also curtails the search space that the verifier must traverse, expediting the process even further.

Given a feature space $F = \{1, ..., m\}$, a bundle b is just a subset $b \subseteq F$. When dealing with the set of all bundles $B = \{b_1, b_2, ...b_n\}$, we require that they form a partitioning of F, namely $F = \cup b_i$. We define a *bundle explanation* E_B for a classification instance (v, c) as a subset of bundles, $E_B \subseteq B$, such that:

$$\forall(x \in \mathbb{F}).[\wedge_{i \in \cup E_B}(x_i = v_i) \to (N(x) = c)] \tag{5}$$

The following theorem then connects bundle explanations and explicit, non-bundle explanations:

Theorem 1. *The union of features in a bundle explanation is an explanation.*

The proof directly follows from Eqs. 1 and 5. We note that this definition of bundles implies that features that are not part of the bundle explanation (i.e. features contained in *"free" bundles*) are "free" to be set to any possible value. Another possible alternative for defining bundles could be to allow features in "free" bundles to only change in the same, coordinated manner. We focus here on the former definition, and leave the alternative definition for future work.

Many of the aforementioned results and definitions for explanations can be extended to bundle explanations. In a similar manner to Eq. 5, we can define the notions of minimal and minimum bundle explanations, a contrastive bundle singleton, and contrastive bundle pairs (see Sec. D of the appendix). Theorems 1 and 2 can be extended to bundle explanations in a straightforward manner. It then follows that all bundle explanations contain all contrastive singleton bundles, and that all bundle explanations contain at least one bundle of any contrastive bundle pair.

Our method from Secs. 3 and 4 can be similarly performed on bundles rather than on features, and T_{UB} would then be used for calculating a minimal bundle explanation, rather than a minimal explanation. Regarding the aforementioned approximation ratio, we discuss and evaluate two different methods for obtaining it. The first, natural approach is to apply our techniques from Sec. 3 on bundle explanations, thus obtaining a provable approximation for a *minimum bundle explanation*. The upper bound is trivially derived by the size of the bundle explanation found by T_{UB}, whereas the lower bound calculation requires assigning a cost to each bundle, representing the number of features it contains. This is done via a known notion of *minimum hitting sets of bundles (MHSB) [7]* and using minimum *weighted* vertex cover for the approximation of contrastive bundle pairs. This method, which is almost identical to the one mentioned in Sec. 3, is formalized in Sec. E of the appendix.

The second approach is to calculate an approximation ratio with respect to a regular, non-bundle minimum explanation. The minimal bundle explanation found by T_{UB} is an upper bound on the minimum non-bundle explanation following theorem 5. For computing a lower bound, we can analyze contrastive bundle examples; extract from them contrastive non-bundle examples; and then use the duality property, compute an MHS of these contrastive examples, and derive lower bounds for the size of the minimum explanation. We formalize techniques for performing this calculation in Sec. E of the appendix.

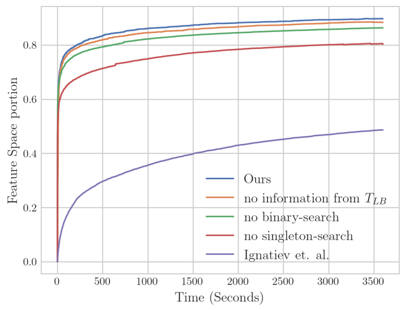
6 Evaluation

Implementation and Setup. For evaluation purposes, we created a proof-of-concept implementation of our approach as a Python framework. Currently, the framework uses the Marabou verification engine [42] as a backend, although other engines may be used. Marabou is a Simplex-based DNN verification framework that is sound and complete [3,6,40–42,69,70], and which includes support for proof production [36], abstraction [16,17,53,61,68,73], and optimization [63]; and has been used in various settings, such as ensemble selection [4], simplification [23,44] repair [24,54], and verification of reinforcement-learning based systems [2,5,18]. For sorting features by their relevance, we used the popular XAI method LIME [56]; although again, other heuristics could be used. The MVC was calculated using the classic 2-approximating greedy algorithm. All experiments reported were conducted on x86-64 Gnu/Linux-based machines, using a single Intel(R) Xeon(R) Gold 6130 CPU @ 2.10GHz core, with a 1-hour timeout.

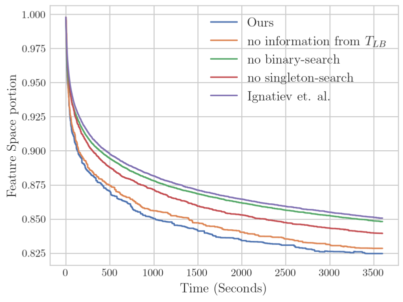
Benchmarks. As benchmarks, we used DNNs trained over the MNIST dataset for handwritten digit recognition [46]. These networks classify 28×28 grayscale images into the digits $0, \ldots, 9$. Additionally, we used DNNs trained over the Fashion-MNIST dataset [72], which classify 28×28 grayscale images into 10 clothing categories ("Dress", "Coat", etc.) For each of these datasets we trained a DNN with the following architecture: (i) an input layer (which corresponds to the image) of size 784; (ii) a fully connected hidden layer with 30 neurons; (iii) another fully connected hidden layer, with 10 neurons; and (iv) a final, softmax layer with 10 neurons, corresponding to the 10 possible output classes. The accuracy of the MNIST DNN was 96.6%, whereas that of the Fashion-MNIST DNN was 87.6%. (We note that we configured LIME to ignore the external border pixels of each input, as these are not part of the actual image.)

In selecting the classification instances to be explained for these networks, we targeted input points where the network was not confident — i.e., where the winning label did not win by a large margin. The motivation for this choice is that explanations are most useful and relevant in cases where the network's decision is unclear, which is reflected in lower confidence scores. Additionally, explanations of instances with lower confidence tend to be larger, facilitating the process of extensive experimentation. We thus selected the 100 inputs from the MNIST and the Fashion-MNIST datasets where the networks demonstrated the lowest confidence scores — i.e., where the difference between the winning output score and the runner-up class score was minimal.

Experiments. Our first goal was to compare our approach to that of Ignatiev et al. [32], which is the current state of the art in verification-based explainability of DNNs. Other approaches consider other ML types, such as descision trees [34,37], or focus on alternative definitions for abductive explanations [43,71] and are thus not comparable. Because the implementation used in [32] is unavailable, we implemented their approach, using Marabou as the underlying verifier for a fair comparison. In addition, we used the same heuristic model, LIME,



(a) Average portion of features verified to participate in the explanation.



(b) Average explanation size.

Fig. 7: Our full and ablation-based results, compared to the state of the art for finding minimal explanations on the MNIST dataset.

for sorting the input features' relevance. Fig. 7 depicts a comparison of the two approaches, over the MNIST benchmarks. The Fashion-MNIST results were similar, but since the Fashion-MNIST network had lower accuracy it tended to produce larger explanations with lower run-times, resulting in less meaningful evaluations (due to space limitations, these results appear in Fig. 10 in the appendix). We compared the approaches according to two criteria: the portion of input features whose participation in the explanation was verified, over time (part (a) of Fig. 7), and the average size of the presently obtained explanation over time, also presented as a fraction of the total number of input features (part (b)). The results indicate that our method significantly improves over the state of the art, verifying the participation of 40.4% additional features, on average, and producing explanations that are 9.7% smaller, on average, at the end of the 1-hour time limit. Furthermore, our method timed out on 10% fewer benchmarks. We regard this as compelling evidence of the potential of our approach to produce more efficient verification-based XAI.

We also looked into comparing our approach to heuristic, non-verification-based approaches, such as LIME itself; but these comparisons did not prove to be meaningful, as the heuristic approaches typically solved benchmarks very quickly, but very often produced incorrect explanations. This matches the findings reported in previous work [14,33].

Next, we set out to evaluate the contribution of each of the components implemented within our framework to overall performance, using an ablation study. Specifically, we ran our framework with each of the components mentioned in Sec. 4, i.e. (i) information exchange between T_{UB} and T_{LB}; (ii) the binary search in T_{UB}; and (iii) local-singleton search, turned off. The results on the MNIST benchmarks appear in Fig. 7; see Fig. 10 in the appendix for the Fashion-MNIST results. Our experiments revealed that each of the methods mentioned in Sec. 4 had a favorable impact on both the average portion of features verified, and the average size of the discovered explanation, over time. Fig 7a indicates that the local-singleton search method, used for efficiently proving that features are bound to be *included* in the explanation, was the most significant in reducing the number of features remained for verifying, thus substantially increasing the portion of verified features. Moreover, Fig. 7b indicates that the binary search method, which is used for grouping `UNSAT` queries and proving the *exclusion* of features from the explanation, was the most significant for more efficiently obtaining smaller-sized explanations, over time.

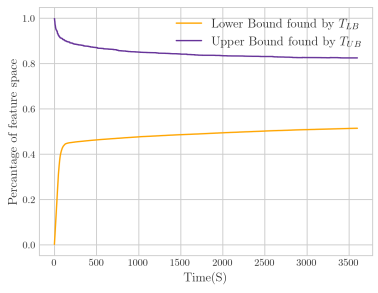


Fig. 8: Average approximation of *minimum* explanation over time.

Our second goal was to evaluate the quality of the *minimum* explanation approximation of our method (using the lower/upper bounds) over time. Results are averaged over all benchmarks of the MNIST dataset and are presented in Fig. 8 (similar results on Fashion-MNIST appear in Fig. 11 in the appendix). The upper bound represents the average size of the explanation discovered by T_{UB} over time, whereas the lower bound represents the average lower bound discovered by T_{LB} over time. It can be seen that initially, there is a steep increase in the size of the lower bound, as T_{LB} discovered many contrastive singletons. Later, as we begin iterating over contrastive pairs, the verification queries take longer to solve, and progress becomes slower. The average approximation ratio achieved after an hour was 1.61 for MNIST and 1.19 for Fashion-MNIST.

For our third experiment, we set out to assess the improvements afforded by bundles. We repeated the aforementioned experiments, this time using sets of features representing bundles instead of the features themselves. The segmentation into bundles was performed using the *quickshift* method [66], with LIME again used for assigning relevance to each bundle [56]. We approximate the sizes of the bundle explanations in terms of both the minimum bundle explanation as well as the minimum (non-bundle) explanation (as mentioned in Sec. 5 and in Sec. E of the appendix). The bundle configuration showed drastic efficiency improvements, with none of the experiments timing out within the 1-hour time limit, thus improving the portion of timeouts on the MNIST dataset by 84%. The efficiency improvement was obtained at the expense of explanation size, resulting in a decrease of 352% in the approximation ratios obtained for MNIST and 39% for Fashion-MNIST. Nevertheless, when calculating the approximation in terms of the *minimum bundle explanation*, an increase of 12% and 8% was obtained for MNIST and Fashion-MNIST (results are summarized in Table. 1 in the appendix). For a visual evaluation, we performed the same set of experiments for both bundle and non-bundle implementations, using instances with high confidence rates to obtain smaller-sized explanations that could be more easily interpreted. A sample of these results is presented in Fig. 9. Empirically, we observe that the bundle-produced explanations are less complex and more comprehensible.

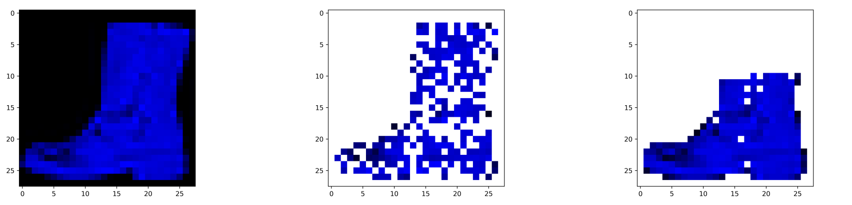

(a) Original Image (b) Explanation (c) Bundle explanation

Fig. 9: Minimal explanations and bundle explanations found by our method on the Fashion-MNIST dataset. White pixels are not part of the explanation.

Overall, we regard our results as compelling evidence that verification-based XAI can soundly produce meaningful explanations, and that our improvements can indeed significantly improve its runtime.

7 Related Work

Our work is another step in the ongoing quest for formal explainability of DNNs, using verification [20, 28, 32, 59]. Related approaches have applied enumeration of contrastive examples [31, 32], which is also an ingredient of our approach. Other approaches focus on producing abductive explanations around an epsilon environment [43, 71]. Similar work has been carried out for decision sets [34], lists [29] and trees [37], where the problem appears to be simpler to solve [37]. Our work here tackles DNNs, which are known to be more difficult to verify [40].

Prior work has also sought to produce approximate explanations, e.g., by using δ-relevant sets [38, 67]. This line of work has focused on probabilistic methods for generating explanations, which jeopardizes soundness. There has also been extensive work in heuristic XAI [51, 56, 57, 62], but here, too, the produced explanations are not guaranteed to be correct.

8 Conclusion

Although DNNs are becoming crucial components of safety-critical systems, they remain "black-boxes", and cannot be interpreted by humans. Our work seeks to mitigate this concern, by providing formally correct explanations for the choices that a DNN makes. Since discovering the minimum explanations is difficult, we focus on approximate explanations, and suggest multiple techniques for expediting our approach — thus significantly improving over the current state of the art. In addition, we propose to use bundles to efficiently produce more meaningful explanations. Moving forward, we plan to leverage lightweight DNN verification techniques for improving the scalability of our approach [50], as well as extend it to support additional DNN architectures.

References

1. M. Akintunde, A. Kevorchian, A. Lomuscio, and E. Pirovano. Verification of RNN-Based Neural Agent-Environment Systems. In *Proc. 33rd AAAI Conf. on Artificial Intelligence (AAAI)*, pages 197–210, 2019.
2. G. Amir, D. Corsi, R. Yerushalmi, L. Marzari, D. Harel, A. Farinelli, and G. Katz. Verifying Learning-Based Robotic Navigation Systems, 2022. Technical Report. `https://arxiv.org/abs/2205.13536`.
3. G. Amir, Z. Freund, G. Katz, E. Mandelbaum, and I. Refaeli. verifire: Verifying an industrial, learning-based wildfire detection system. *arXiv preprint arXiv:2212.03287*, 2022.
4. G. Amir, G. Katz, and M. Schapira. Verification-Aided Deep Ensemble Selection. In *Proc. 22nd Int. Conf. on Formal Methods in Computer-Aided Design (FMCAD)*, pages 27–37, 2022.
5. G. Amir, M. Schapira, and G. Katz. Towards Scalable Verification of Deep Reinforcement Learning. In *Proc. 21st Int. Conf. on Formal Methods in Computer-Aided Design (FMCAD)*, pages 193–203, 2021.
6. G. Amir, H. Wu, C. Barrett, and G. Katz. An SMT-Based Approach for Verifying Binarized Neural Networks. In *Proc. 27th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pages 203–222, 2021.
7. E. Angel, E. Bampis, and L. Gourvès. On the Minimum Hitting Set of Bundles Problem. *Theoretical Computer Science*, 410(45):4534–4542, 2009.
8. J. Angwin, J. Larson, S. Mattu, and L. Kirchner. Machine Bias. *Ethics of Data and Analytics*, pages 254–264, 2016.
9. G. Avni, R. Bloem, K. Chatterjee, T. Henzinger, B. Konighofer, and S. Pranger. Run-Time Optimization for Learned Controllers through Quantitative Games. In *Proc. 31st Int. Conf. on Computer Aided Verification (CAV)*, pages 630–649, 2019.
10. T. Baluta, S. Shen, S. Shinde, K. Meel, and P. Saxena. Quantitative Verification of Neural Networks And its Security Applications. In *Proc. 26th ACM Conf. on Computer and Communication Security (CCS)*, pages 1249–1264, 2019.
11. R. Bar-Yehuda and S. Even. A Linear-Time Approximation Algorithm for the Weighted Vertex Cover Problem. *Journal of Algorithms*, 2(2):198–203, 1981.
12. M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba. End to End Learning for Self-Driving Cars, 2016. Technical Report. `http://arxiv.org/abs/1604.07316`.
13. CACM. A Case Against Mission-Critical Applications of Machine Learning. *Communications of the ACM*, 62(8):9–9, 2019.
14. O.-M. Camburu, E. Giunchiglia, J. Foerster, T. Lukasiewicz, and P. Blunsom. Can I Trust the Explainer? Verifying Post-Hoc Explanatory Methods, 2019. Technical Report. `http://arxiv.org/abs/1910.02065`.
15. I. Dinur, V. Guruswami, S. Khot, and O. Regev. A New Multilayered PCP and the Hardness of Hypergraph Vertex Cover. In *Proc. 35th ACM Symposium on Theory of Computing*, pages 595–601, 2003.
16. Y. Elboher, E. Cohen, and G. Katz. Neural Network Verification using Residual Reasoning. In *Proc. 20th Int. Conf. on Software Engineering and Formal Methods (SEFM)*, pages 173–189, 2022.
17. Y. Elboher, J. Gottschlich, and G. Katz. An Abstraction-Based Framework for Neural Network Verification. In *Proc. 32nd Int. Conf. on Computer Aided Verification (CAV)*, pages 43–65, 2020.

18. T. Eliyahu, Y. Kazak, G. Katz, and M. Schapira. Verifying Learning-Augmented Systems. In *Proc. Conf. of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM)*, pages 305–318, 2021.
19. A. Esteva, A. Robicquet, B. Ramsundar, V. Kuleshov, M. DePristo, K. Chou, C. Cui, G. Corrado, S. Thrun, and J. Dean. A Guide to Deep Learning in Healthcare. *Nature Medicine*, 25(1):24–29, 2019.
20. T. Fel, M. Ducoffe, D. Vigouroux, R. Cadène, M. Capelle, C. Nicodème, and T. Serre. Don't Lie to Me! Robust and Efficient Explainability with Verified Perturbation Analysis, 2022. Technical Report. `http://arXivpreprintarXiv:2202.07728`.
21. A. Gainer-Dewar and P. Vera-Licona. The Minimal Hitting Set Generation Problem: Algorithms and Computation. *SIAM Journal on Discrete Mathematics*, 31(1):63–100, 2017.
22. S. Garg and G. Ramakrishnan. Bae: Bert-based adversarial examples for text classification. *arXiv preprint arXiv:2004.01970*, 2020.
23. S. Gokulanathan, A. Feldsher, A. Malca, C. Barrett, and G. Katz. Simplifying Neural Networks using Formal Verification. In *Proc. 12th NASA Formal Methods Symposium (NFM)*, pages 85–93, 2020.
24. B. Goldberger, Y. Adi, J. Keshet, and G. Katz. Minimal Modifications of Deep Neural Networks using Verification. In *Proc. 23rd Int. Conf. on Logic for Programming, Artificial Intelligence and Reasoning (LPAR)*, pages 260–278, 2020.
25. I. Goodfellow, J. Shlens, and C. Szegedy. Explaining and Harnessing Adversarial Examples, 2014. Technical Report. `http://arxiv.org/abs/1412.6572`.
26. S. Huang, N. Papernot, I. Goodfellow, Y. Duan, and P. Abbeel. Adversarial Attacks on Neural Network Policies, 2017. Technical Report. `http://arxiv.org/abs/1702.02284`.
27. IBM. The CPLEX optimizer, 2018.
28. A. Ignatiev. Towards Trustable Explainable AI. In *Proc. 29th Int. Joint Conf. on Artificial Intelligence (IJCAI)*, pages 5154–5158, 2020.
29. A. Ignatiev and J. Marques-Silva. SAT-Based Rigorous Explanations for Decision Lists. In *Proc. 24th Int. Conf. on Theory and Applications of Satisfiability Testing (SAT)*, pages 251–269, 2021.
30. A. Ignatiev, A. Morgado, and J. Marques-Silva. Propositional Abduction with Implicit Hitting Sets, 2016. Technical Report. `http://arxiv.org/abs/1604.08229`.
31. A. Ignatiev, N. Narodytska, N. Asher, and J. Marques-Silva. From Contrastive to Abductive Explanations and Back Again. In *Proc. 19th Int. Conf. of the Italian Association for Artificial Intelligence (AIxIA)*, pages 335–355, 2020.
32. A. Ignatiev, N. Narodytska, and J. Marques-Silva. Abduction-Based Explanations for Machine Learning Models. In *Proc. 33rd AAAI Conf. on Artificial Intelligence (AAAI)*, pages 1511–1519, 2019.
33. A. Ignatiev, N. Narodytska, and J. Marques-Silva. On Validating, Repairing and Refining Heuristic Ml Explanations, 2019. Technical Report. `http://arxiv.org/abs/1907.02509`.
34. A. Ignatiev, F. Pereira, N. Narodytska, and J. Marques-Silva. A SAT-Based Approach to Learn Explainable Decision Sets. In *Proc. 9th Int. Joint Conf. on Automated Reasoning (IJCAR)*, pages 627–645, 2018.
35. A. Ignatiev, A. Previti, M. Liffiton, and J. Marques-Silva. Smallest MUS Extraction with Minimal Hitting Set Dualization. In *Proc. 21st Int. Conf. on Principles and Practice of Constraint Programming (CP)*, pages 173–182, 2015.

36. O. Isac, C. Barrett, M. Zhang, and G. Katz. Neural Network Verification with Proof Production. In *Proc. 22nd Int. Conf. on Formal Methods in Computer-Aided Design (FMCAD)*, pages 38–48, 2022.
37. Y. Izza, A. Ignatiev, and J. Marques-Silva. On Explaining Decision Trees, 2020. Technical Report. http://arxiv.org/abs/2010.11034.
38. Y. Izza, A. Ignatiev, N. Narodytska, M. Cooper, and J. Marques-Silva. Efficient Explanations with Relevant Sets, 2021. Technical Report. http://arxiv.org/abs/2106.00546.
39. K. Julian, M. Kochenderfer, and M. Owen. Deep Neural Network Compression for Aircraft Collision Avoidance Systems. *Journal of Guidance, Control, and Dynamics*, 42(3):598–608, 2019.
40. G. Katz, C. Barrett, D. Dill, K. Julian, and M. Kochenderfer. Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks. In *Proc. 29th Int. Conf. on Computer Aided Verification (CAV)*, pages 97–117, 2017.
41. G. Katz, C. Barrett, D. Dill, K. Julian, and M. Kochenderfer. Reluplex: a Calculus for Reasoning about Deep Neural Networks. *Formal Methods in System Design (FMSD)*, 2021.
42. G. Katz, D. Huang, D. Ibeling, K. Julian, C. Lazarus, R. Lim, P. Shah, S. Thakoor, H. Wu, A. Zeljić, D. Dill, M. Kochenderfer, and C. Barrett. The Marabou Framework for Verification and Analysis of Deep Neural Networks. In *Proc. 31st Int. Conf. on Computer Aided Verification (CAV)*, pages 443–452, 2019.
43. E. La Malfa, A. Zbrzezny, R. Michelmore, N. Paoletti, and M. Kwiatkowska. On guaranteed optimal robust explanations for nlp models. *arXiv preprint arXiv:2105.03640*, 2021.
44. O. Lahav and G. Katz. Pruning and Slicing Neural Networks using Formal Verification. In *Proc. 21st Int. Conf. on Formal Methods in Computer-Aided Design (FMCAD)*, pages 183–192, 2021.
45. H. Lakkaraju and O. Bastani. "How do I Fool You?" Manipulating User Trust via Misleading Black Box Explanations. In *Proc. AAAI/ACM Conf. on AI, Ethics, and Society (AIES)*, pages 79–85, 2020.
46. Y. LeCun. The MNIST Database of Handwritten Digits, 1998. http://yann.lecun.com/exdb/mnist/.
47. Y. LeCun, Y. Bengio, and G. Hinton. Deep Learning. *Nature*, 521(7553):436–444, 2015.
48. C. Li and F. Manya. MaxSAT, Hard and Soft Constraints. In *Handbook of Satisfiability*, pages 903–927. IOS Press, 2021.
49. P. Liberatore. Redundancy in logic i: Cnf propositional formulae. *Artificial Intelligence*, 163(2):203–232, 2005.
50. C. Liu, T. Arnon, C. Lazarus, C. Barrett, and M. Kochenderfer. Algorithms for Verifying Deep Neural Networks, 2020. Technical Report. http://arxiv.org/abs/1903.06758.
51. S. M. Lundberg and S.-I. Lee. A Unified Approach to Interpreting Model Predictions. In *Proc. 31st Conf. on Neural Information Processing Systems (NeurIPS)*, 2017.
52. J. Marques-Silva and A. Ignatiev. Delivering Trustworthy AI through formal XAI. In *Proc. 36th AAAI Conf. on Artificial Intelligence (AAAI)*, pages 3806–3814, 2022.
53. M. Ostrovsky, C. Barrett, and G. Katz. An Abstraction-Refinement Approach to Verifying Convolutional Neural Networks. In *Proc. 20th. Int. Symposium on Automated Technology for Verification and Analysis (ATVA)*, 2022.

54. I. Refaeli and G. Katz. Minimal Multi-Layer Modifications of Deep Neural Networks. In *Proc. 5th Workshop on Formal Methods for ML-Enabled Autonomous Systems (FoMLAS)*, 2022.
55. R. Reiter. A Theory of Diagnosis from First Principles. *Artificial Intelligence*, 32(1):57–95, 1987.
56. M. Ribeiro, S. Singh, and C. Guestrin. “Why should I Trust You?” Explaining the Predictions of any Classifier. In *Proc. 22nd Int. Conf. on Knowledge Discovery and Data Mining (KDD)*, pages 1135–1144, 2016.
57. M. Ribeiro, S. Singh, and C. Guestrin. Anchors: High-Precision Model-Agnostic Explanations. In *Proc. 32nd AAAI Conf. on Artificial Intelligence (AAAI)*, 2018.
58. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra. Grad-Cam: Visual Explanations from Deep Networks via Gradient-Based Localization. In *Proc. 20th IEEE Int. Conf. on Computer Vision (ICCV)*, pages 618–626, 2017.
59. W. Shi, A. Shih, A. Darwiche, and A. Choi. On Tractable Representations of Binary Neural Networks, 2020. Technical Report. `http://arxiv.org/abs/2004.02082`.
60. A. Shih, A. Choi, and A. Darwiche. A Symbolic Approach to Explaining Bayesian Network Classifiers, 2018. Technical Report. `http://arxiv.org/abs/1805.03364`.
61. G. Singh, T. Gehr, M. Puschel, and M. Vechev. An Abstract Domain for Certifying Neural Networks. In *Proc. 46th ACM SIGPLAN Symposium on Principles of Programming Languages (POPL)*, 2019.
62. D. Smilkov, N. Thorat, B. Kim, F. Viégas, and M. Wattenberg. Smoothgrad: Removing Noise by Adding Noise, 2017. Technical Report. `http://arxiv.org/abs/1706.03825`.
63. C. Strong, H. Wu, A. Zeljić, K. Julian, G. Katz, C. Barrett, and M. Kochenderfer. Global Optimization of Objective Functions Represented by ReLU Networks. *Journal of Machine Learning*, pages 1–28, 2021.
64. D. Stutz, A. Hermans, and B. Leibe. Superpixels: An Evaluation of the State-of-the-Art. *Computer Vision and Image Understanding*, 166:1–27, 2018.
65. J. Su, D. Vargas, and K. Sakurai. One Pixel Attack for Fooling Deep Neural Networks. *IEEE Transactions on Evolutionary Computation*, 23(5):828–841, 2019.
66. A. Vedaldi and S. Soatto. Quick Shift and Kernel Methods for Mode Seeking. In *Proc. 10th European Conf. on Computer Vision (ECCV)*, pages 705–718, 2008.
67. S. Waeldchen, J. Macdonald, S. Hauch, and G. Kutyniok. The Computational Complexity of Understanding Binary Classifier Decisions. *Journal of Artificial Intelligence Research*, 70:351–387, 2021.
68. S. Wang, K. Pei, J. Whitehouse, J. Yang, and S. Jana. Formal Security Analysis of Neural Networks using Symbolic Intervals. In *Proc. 27th USENIX Security Symposium*, 2018.
69. H. Wu, A. Ozdemir, A. Zeljić, A. Irfan, K. Julian, D. Gopinath, S. Fouladi, G. Katz, C. Păsăreanu, and C. Barrett. Parallelization Techniques for Verifying Neural Networks. In *Proc. 20th Int. Conf. on Formal Methods in Computer-Aided Design (FMCAD)*, pages 128–137, 2020.
70. H. Wu, A. Zeljić, G. Katz, and C. Barrett. Efficient Neural Network Analysis with Sum-of-Infeasibilities. In *Proc. 28th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pages 143–163, 2022.
71. M. Wu, H. Wu, and C. Barrett. Verix: Towards verified explainability of deep neural networks. *arXiv preprint arXiv:2212.01051*, 2022.
72. H. Xiao, K. Rasul, and R. Vollgraf. Fashion-MNist: a Novel Image Dataset for Benchmarking Machine Learning Algorithms, 2017. Technical Report. `http://arxiv.org/abs/1708.07747`.

73. T. Zelazny, H. Wu, C. Barrett, and G. Katz. On Reducing Over-Approximation Errors for Neural Network Verification. In *Proc. 22nd Int. Conf. on Formal Methods in Computer-Aided Design (FMCAD)*, pages 17–26, 2022.
74. Z. Zhou and L. Sun. Metamorphic Testing of Driverless Cars. *Communications of the ACM*, 62(3):61–67, 2019.

Appendix

A Explanations and Contrastive Examples

Lemma 1. *Every contrastive singleton is contained in all explanations.*

Proof. Let N be a classification function, (v,c) a classification instance, and $S \in F$ a contrastive singleton. Assume towards contradiction that S is not contained in some explanation $E \subseteq F$ of instance (v,c), namely $S \notin E$. This means that $E \subseteq F \setminus S$, and from Theorem 3 we can conclude that:

$$\exists(x \in \mathbb{F}).[\wedge_{i \in E}(x_i = v_i) \wedge (N(x) \neq c)]$$

which is a contradiction to Eq. 1. □

Lemma 2. *All explanations contain at least one element of every contrastive pair.*

Proof. The proof is almost identical to that of Lemma 1. Let N be a classification function, (v,c) a classification instance, $P \in F$ a contrastive pair and $E \subseteq F$ an explanation of instance (v,c). Assume towards contradiction that there does not exist any element from the pair P that is contained in E, meaning that $P \notin E$. Hence, $E \subseteq F \setminus P$, and from Theorem 3 we can conclude that:

$$\exists(x \in \mathbb{F}).[\wedge_{i \in E}(x_i = v_i) \wedge (N(x) \neq c)]$$

which is a contradiction to Eq. 1. □

See Fig. 12 for an illustration of contrastive singletons and pairs.

B Minimum Hitting Set (MHS) and Minimum Vertex Cover (MVC)

Minimum Hitting Set (MHS). Given a collection $\mathbb{S}$ of sets from a universe U, a hitting set h for $\mathbb{S}$ is a set such that $\forall S \in \mathbb{S}, h \cap S \neq \emptyset$. A hitting set h is said to be *minimal* if none of its subsets is a hitting set, and *minimum* when it has the smallest possible cardinality among all hitting sets.

K Minimum Hitting Set (K-MHS). K-MHS denotes the same problem as MHS, but when sets are at the size of at most k. This is a re-formalization of the minimum vertex cover (MVC) problem on a k-hyper-graph, where sets are treated as edges and elements in sets are treated as vertices. This implies that a 2-MHS problem is simply the classic *minimum vertex cover (MVC)* problem.

C Extending T_{LB} beyond Contrastive Pairs

The lower bounding thread, T_{LB}, (see Sec. 3) is used for computing a lower bound on the size of the minimum explanation. This is done by computing contrastive singletons and contrastive pairs, and using them in calculating an under approximation for the MHS of all contrastive examples. This approach can be extended to use contrastive examples of larger sizes ($k = 3, 4, \ldots$). Finding these contrastive examples may improve the approximation of the global minimum, but may also render the approach less scalable. In the worst case, finding all sets of size k requires performing $O(\binom{m}{k})$ queries to the underlying verifier. Since the search space becomes larger as k increases, each query may become more expensive as well. In case of a large feature spaces, if we are interested in performing an approximation via a greedy algorithm, the quality of the approximation also deteriorates as k increases. The general K-MHS problem has a polynomial k-approximating algorithm, and this bound was shown to be tight for all $k \geq 3$ [15]. Theoretically, if T_{LB} continues finding contrastive examples of larger sizes, k in the final step is the minimum k on which the MHS of all contrastive examples of size k and less are equal to the minimum explanation. The full, exact approximation can be summarized by the following formula:

$$\begin{aligned} \text{LB} &= |\text{Singletons}| + |\text{MVC(Pairs)}| \\ &\leq \sum_{\text{k=1}}^{\text{k=maxk}} |\text{K-MHS(k-Cxps)}| \\ &= \text{MHS(Cxps)} = E_M \end{aligned} \tag{6}$$

where k-Cxps denotes all contrastive examples of size k, and maxk denotes the size of k in the final iteration.

D Minimal Bundle Explanations

Let $v = (v_1, ...v_m) \in \mathbb{F}$ be an input with classification $N(v) = c$, and let B be the set of all bundles. The definition of a *minimal bundle explanation* $E_B \subseteq B$ for instance (v, c) is:

$$\begin{aligned} &(\forall (x \in \mathbb{F}).[\wedge_{i \in \cup E_B}(x_i = v_i) \to (N(x) = c)]) \wedge \\ &(\forall (j_B \in E_B).[\exists (y \in \mathbb{F}).[\wedge_{i \in \cup E_B \setminus j_B}(y_i = v_i) \wedge (N(y) \neq c)]) \end{aligned} \tag{7}$$

A *minimum bundle explanation* is a minimal bundle explanation of minimum size. We define a *contrastive bundle example* C_B for input (v, c) and the set of all bundles B, as a subset of bundles $C_B \subseteq B$ such that:

$$\exists (x \in \mathbb{F}).[\wedge_{i \in \cup B \setminus C_B}(x_i = v_i) \wedge (N(x) \neq c)] \tag{8}$$

A *contrastive bundle singleton* is defined as a contrastive bundle example of size 1, and a *contrastive bundle pair* as a contrastive bundle example of size 2 (which does not contain contrastive bundle singletons).

Minimum Hitting Set of Bundles (MHSB). We use the common definition for a *minimum hitting set of bundles (MHSB)* [7], which is as follows. Given a set of n elements $\mathcal{E} = \{e_1, e_2, ..., e_n\}$, each element e_i $(i = 1, ...n)$ has a non-negative cost c_i. A bundle b is a subset of $\mathcal{E}$. We are also given a collection $\mathcal{S} = \{S_1, S_2, ..., S_m\}$ of m sets of bundles. More precisely, each set S_j $(j = 1, ..., m)$ is composed of $g(j)$ distinct bundles $b_j^1 b_j^2, ..., b_j^{g(j)}$. A solution to MHSB is a subset $\mathcal{E}' \subseteq \mathcal{E}$ such that for every $S_j \in \mathcal{S}$, at least one bundle is covered, i.e, $b_j^l \subseteq \mathcal{E}'$ for some $l \in \{1, 2, ..., g(j)\}$. The total cost of the solution, denoted by $C(\mathcal{E}')$, is $\sum_{\{i \mid e_i \in e'\}} c_i$ (a cost of an element appearing in several bundles is counted once). The objective is to find a solution with minimum total cost.

Notice that this is a more general definition than our case, where each element (feature) inside a bundle has a cost of 1, meaning that the cost of each bundle is the number of features it contains. The proven dual MHS relationship between explanations and contrastive examples [31] can thus be trivially expanded to include bundles —i.e., that the MHSB of all contrastive bundle examples constitutes as the minimum bundle explanation and that the MHSB of all bundle explanations constitutes as a minimum bundle contrastive example.

E Approximating Bundle Explanations

Method 1: Approximating Minimum Bundle Explanations. We first discuss how to obtain a provable approximation for the minimum bundle explanation. This is a trivial extension of our method suggested in Sec. 3. The minimal bundle explanation found by T_{UB} is clearly an upper bound for the minimum bundle explanation. Regarding the lower bound computed by T_{LB}, the union of all bundle singletons is a lower bound, since every contrastive bundle singleton is contained in the minimum bundle explanation. Moreover, the minimum *weighted* vertex cover of all contrastive bundle pairs (where weights indicate the number of features in each bundle) constitutes a lower bound for the MHSB of all contrastive bundle examples. In cases of large feature spaces, a 2-approximating linear greedy algorithm can be used for the minimum weighted vertex cover [11]. Overall, the following lower bound can be calculated:

$$\begin{aligned} LB_{Bundle} &= |\cup \text{BSingletons}| + |\text{MWVC(BPairs)}| \\ &\leq |\text{MHSB}(Cxps_B)| = E_{\text{MB}} \end{aligned} \tag{9}$$

where LB_{Bundle} denotes the lower bound that is calculated for our evaluation, BSingletons denotes the set of all contrastive bundle singletons, Bpairs denotes the set of all contrastive bundle pairs, MWVC is the minimum weighted vertex cover, $Cxps_B$ denotes the set of all contrastive bundle examples, and E_{MB} is the minimum bundle explanation.

Method 2: Approximating Minimum (Non-Bundle) Explanations. The second approach is to calculate an approximation ratio with respect to a regular, non-bundle minimum explanation. The minimal bundle explanation found by

T_{UB} is an upper bound for the minimum non-bundle explanation, since the minimal bundle explanation is also an explanation by itself (Theorem 5). For obtaining the lower bound, we can analyze contrastive bundle examples found by T_{LB} for extracting contrastive non-bundle examples, and thus enabling the computation of an under-approximation to the MHS of all contrastive examples (an illustration is also depicted on Fig. 13).

For example, it is straightforward that every contrastive bundle singleton is a contasrive example by itself, and thus at least one of the bundle's elements is contained in a minimum explanation. Likewise, for every contrastive bundle pair (b_1, b_2) there exist at least two subsets, $s_1 \subseteq b_1$ and $s_2 \subseteq b_2$, such that $s_1 \cup s_2$ is a contrastive example. This means that a regular minimum (unweighted) vertex cover can be calculated by the following approximation (used for our evaluation):

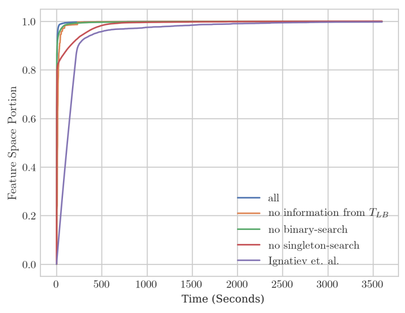
$$|\text{BSingletons}| + |\text{MVC}(\text{BPairs})| \leq E_M \tag{10}$$

An additional, optional approach for tightening the bound even more is to search for contrastive examples of features within each bundle. This can be done for every contrastive bundle singleton, by calculating the MHS of all contrastive examples within that certain bundle. Since bundles typically consist of small feature sets, it may be computationally feasible to compute the MHS of all features within each bundle. If not, the procedure that we suggested in Sec. 3 can be repeated for each bundle. More precisely, we propose to iterate on all features and pairs in each bundle, to find all contrastive singletons and pairs within that bundle, and then to calculate a lower bound by solving an unweighted vertex cover problem. Further, we can perform a similar process (of either calculating the MHS or performing the lower bound procedure from Sec. 3) on the union of all contrastive bundle pairs, as well. Notice that by doing so, we do not need to iterate on the entire set of all pairs, since features that are within the same contrastive bundle are necessarily not contrastive pairs (because otherwise, that bundle would be a contrastive bundle singleton). Thus, we can arrive at the following bound:

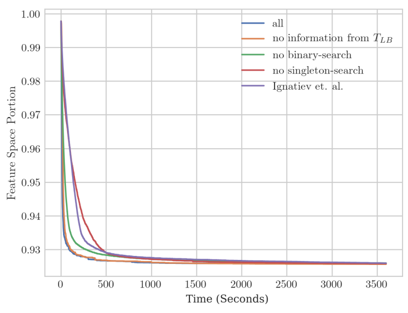
$$\begin{aligned}
|\text{BSingletons}| &+ |\text{MVC}(\text{BPairs})| \\
&\leq |\sum_{\text{BS} \subseteq \text{BSingletons}} \text{LB}(\text{BS})| + |\text{LB}(\cup \text{BPairs})| \\
&\leq \sum_{\text{BS} \subseteq \text{BSingletons}} |\text{MHS}(\text{BS})| + |\text{MHS}(\cup \text{Bpairs})| \\
&\leq E_M
\end{aligned} \tag{11}$$

F Additional Evaluation

Fashion-MNIST Evaluation. Figs. 10 and 11 depict the results of our evaluation using the Fashion-MNIST network.



(a) Average portion of features verified to be included/excluded from explanation



(b) Average size of explanation

Fig. 10: Our full and ablated results compared to the state-of-the-art for finding minimal explanations over the Fashion-MNIST dataset

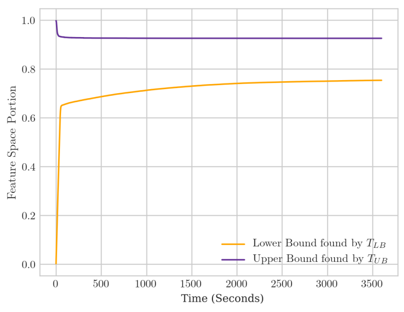


Fig. 11: Average approximation of *minimum* explanation over time on the Fashion-MNIST dataset

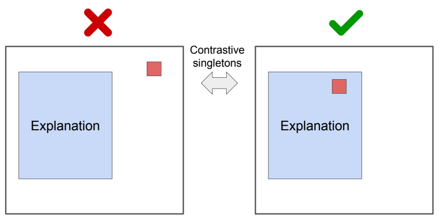
Minimal Bundle Explanation Experiments. Table 1 summarizes the results of our bundle explanation experiments.

Table 1: Bundle and non-bundle implementation results, evaluated using the following criteria: (i) the proportion of completed tasks within a one hour time limit; (ii) the average run-time per task, in seconds; (iii) the average final approximation ratio found of the minimum bundle explanation: E_{MB} (lower bound obtained per Eq. 10); and (iv) the average final approximation found of the minimum explanation: E_M (lower bound obtained per Eq. 9).

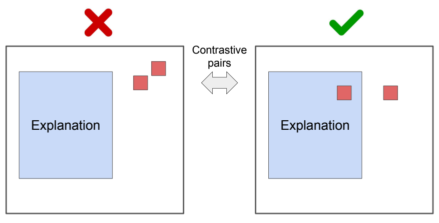
Dataset	**Criteria**	**Bundle Explanation**	**Non-Bundle Explanation**
MNIST	Completed Portion	100%	16%
	Run-time (Seconds)	115.3	3264.5
	E_{MB} Approx.	1.49	-
	E_M Approx.	5.13	1.61
Fashion-MNIST	Completed Portion	100%	100%
	Run-time (Seconds)	119.6	140.82
	E_{MB} Approx.	1.15	-
	E_M Approx.	1.62	1.23

G Additional Figures

We present here additional illustrations that demonstrate contrastive examples (specifically, contrastive singletons and pairs) (Fig. 12), an illustration of the binary search heuristic 14 and the local singleton search heuristic 15 for expediting the search for minimal explanations, and minimal bundle explanation approximations (Fig. 13). Additionally, we attach an illustration of one-pixel attacks, which can be regarded as contrastive singletons (Fig. 16).



(a) Contrastive singletons



(b) Contrastive pairs

Fig. 12: An illustration of contrastive singletons and contrastive pairs. Every singleton is contained in every explanation, and every pair holds at least one feature in every explanation.

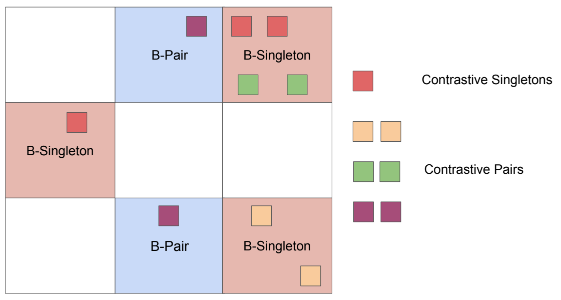


Fig. 13: An illustration of a bundle explanation. B-Singletons and B-Pairs represent contrastive bundle singletons and pairs. A lower bound can be computed either in terms of bundles, or in terms of the features that comprise them.

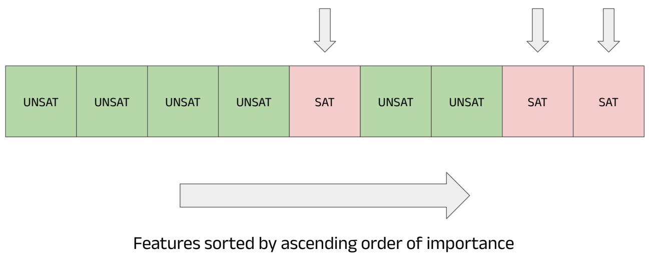


Fig. 14: Features are sorted in increasing order of importance. `SAT` queries indicate features included in the explanation. We perform a sequence of binary searches to identify these `SAT` queries.

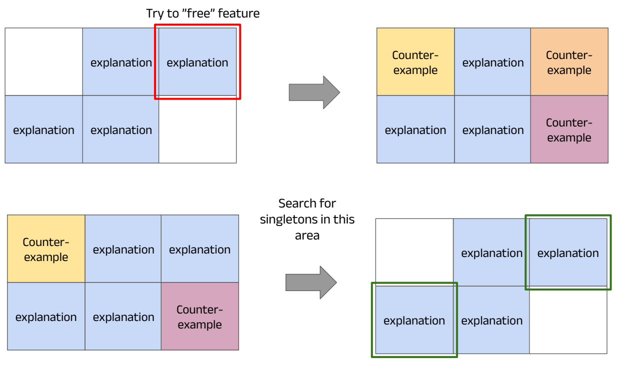


Fig. 15: An illustration of the local-singleton search. Elements in blue represent features that are currently in the explanation (not “freed”), and are thus set to the input value. We try to “free” a feature, find a counterexample, and search for *local singletons* in the nearby area of that counter-example. These features are bound to be part of the explanation, hence reducing the number of features that remained for verifying.

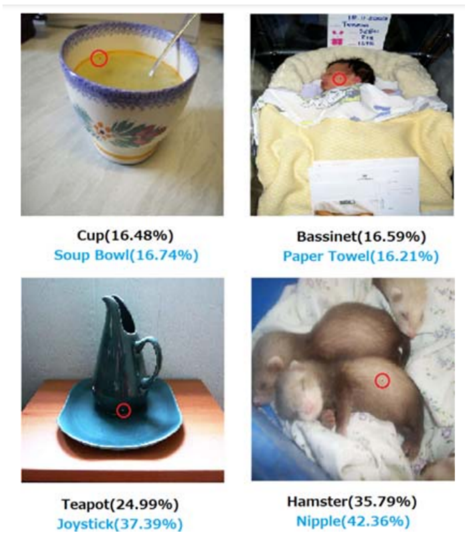


Fig. 16: Examples of one-pixel-attacks on the Imagenet database, borrowed from [65]. These represent *contrastive singletons*.

H Additional Pseudo-Codes

We present here additional pseudo-codes, specifically of the binary search heuristic 6 for expediting the search for minimal explanations.

Algorithm 6 T_{UB} using binary-search

```
1: Use a heuristic model to sort $F$ by ascending relevance
2: $L = 0$
3: $R = |F| - 1$
4: **while** $L \leq |F| - 1$ **do**
5:     **while** $L \leq R$ **do**                    ▷ The inner loop is a single binary search
6:         Mid ← $\frac{L-R}{2}$
7:         Explanation ←F∖Free
8:         **if** Verify((Explanation∖{L,L+1,...,Mid})=v,N,$Q_{\neg c}$) is UNSAT **then**
9:             Free ← Free ∪ {L,L+1,...,Mid}
10:            UB ← UB −|{$L$, $L$+1, ...,Mid}|
11:            $L$ ← Mid+1
12:        **else**
13:            $R$ ← Mid−1
14:        **end if**
15:    **end while**
16:    $L \leftarrow L + 1$
17:    $R \leftarrow |F| - 1$
18: **end while**
```

About Limits In Formal Verification of Graph Neural Networks

Marco Sälzer and Martin Lange

School of Electr. Eng. and Computer Science, University of Kassel, Germany
`marco.saelzer, martin.lange@uni-kassel.de`

Introduction

Deep learning methods, especially their most prominent representative (deep) neural networks, have proved to be very successful in a broad range of applications, delivering impressive results Unsurprisingly, deep learning methods are also used in safety-critical applications like early disease detection [6] or driving assistants [2], which naturally leads to the need for safety certificates. Currently very popular are so called Graph Neural Networks (GNN), a framework for computing functions over graphs. While there is ongoing work on (formal) verification of GNN (see [4] for an overview), there is no work so far giving reliable decidability or complexity bounds of corresponding problems Thus, there are currently no fundamental results that frame and guide the development of GNN verification algorithms.

We made a first contribution to this topic in [8] by establishing first (un-)decidability results regarding the verification problem of common safety properties of so called Message Passing Neural Networks (MPNN), a popular GNN model. We present these in more detail in the following section. After this we discuss promising next steps to unveil further decidability and complexity bounds of GNN verification, using recently established results about the expressiveness of different GNN models.

Fundamental Limits in Formal Verification of MPNN

As it turns out, it is particular difficult to establish broadly applicable decidability or complexity bounds for GNN verification. The reason is that the GNN framework is highly heterogeneous. Wu et al. [9] give a complex taxonomy of models belonging to the GNN framework. Despite the differences between different GNN models, their domain, roughly described as the class of vector-labelled graphs, leads to a variety of different tasks, like classifying specific nodes, specific edges or classifying whole graphs.

In order to address the aforementioned difficulties with the GNN framework, our work [8] focuses on the presumably most popular group of GNN models and applications: spatial-based GNN, represented by MPNN, and node as well as whole-graph classification.

Theorem 1. *[8] The problem of formally verifying output reachability properties of graph-classifying MPNN (with reasonable specifications) is undecidable. On the other hand, the equivalent problem for node-classifying MPNN becomes decidable as soon as we restrict it to graphs of bounded degree.*

We refer to [8] for technical details. In addition, this work presents similar results about a popular safety property called adversarial robustness.

Expressiveness of GNN Through the Lens of Formal Verification

The applicability of our findings in [8] has some obvious limitations: first, the undecidability or decidability results apply without further ado only to GNN models which are at least as expressive respectively at most as expressive as MPNN.

Recently, there has been interest in questions about the expressibility of GNN, comprehensively summarized in [3]. Here, expressibility means the ability to distinguish pairs of nodes respectively pairs of whole graphs. Results from this research can roughly be divided into two directions. First, the expressive power of (mainly node-classifying) GNN is characterized by the Weisfeiler-Lemann algorithm [7], a well-known algorithm for testing graph isomorphism. Second, the expressibility of GNN is characterised in terms of finite variable counting logic [1]. For our purpose, a logical characterisation seems more promising as their is an obvious connection between the satisfiability problem (SAT) of logics and the verification problem of output reachability properties of GNN.

To further motivate this approach, consider the following known connection between node-classifying MPNN and a guarded, two-variable fragment of first order logic, called graded modal logic (GC_2).

Theorem 2. *[1] For each query expressed by some $\varphi \in \mathrm{GC}_2$ there is a node-classifying MPNN expressing the same query.*

This means, that each class of graph-node pairs recognizable by some $\varphi \in \mathrm{GC}_2$ is also recognizable by some node-classifying MPNN. This result in combination with the fact that SAT of GC_2 is known to be PSPACE-hard [5], leads us to the following conjecture.

Conjecture 1. The problem of verifying output-reachability and adversarial robustness properties of node-classifying MPNN is PSPACE-hard.

However, these logical characterizations do not directly yield results for upper complexity or decidability bounds. The reason is that GNN can express properties of the type "node v has twice as many neighbours with label P as with label Q". Such properties cannot be expressed in standard first order logic.

References

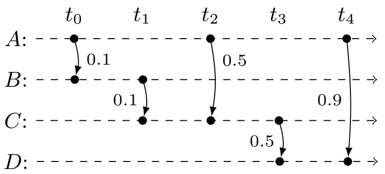
1. Barceló, P., Kostylev, E.V., Monet, M., Pérez, J., Reutter, J.L., Silva, J.P.: The logical expressiveness of graph neural networks. In: 8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020. OpenReview.net (2020), https://openreview.net/forum?id=r1lZ7AEKvB
2. Grigorescu, S.M., Trasnea, B., Cocias, T.T., Macesanu, G.: A survey of deep learning techniques for autonomous driving. J. Field Robotics (2020). https://doi.org/10.1002/rob.21918
3. Grohe, M.: The logic of graph neural networks. In: 36th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021, Rome, Italy, June 29 - July 2, 2021. https://doi.org/10.1109/LICS52264.2021.9470677
4. Günnemann, S.: Graph neural networks: Adversarial robustness. In: Wu, L., Cui, P., Pei, J., Zhao, L. (eds.) Graph Neural Networks: Foundations, Frontiers, and Applications, pp. 149–176. Springer Singapore (2022)
5. Kazakov, Y., Pratt-Hartmann, I.: A note on the complexity of the satisfiability problem for graded modal logics. In: Proceedings of the 24th Annual IEEE Symposium on Logic in Computer Science, LICS 2009, 11-14 August 2009, Los Angeles, CA, USA. pp. 407–416. IEEE Computer Society (2009). https://doi.org/10.1109/LICS.2009.17
6. Litjens, G., Kooi, T., Bejnordi, B.E., Setio, A.A.A., Ciompi, F., Ghafoorian, M., van der Laak, J.A.W.M., van Ginneken, B., Sánchez, C.I.: A survey on deep learning in medical image analysis. Medical Image Anal. (2017). https://doi.org/10.1016/j.media.2017.07.005
7. Morris, C., Lipman, Y., Maron, H., Rieck, B., Kriege, N.M., Grohe, M., Fey, M., Borgwardt, K.M.: Weisfeiler and leman go machine learning: The story so far. CoRR (2021), https://arxiv.org/abs/2112.09992
8. Sälzer, M., Lange, M.: Fundamental limits in formal verification of message-passing neural networks. In: The Eleventh International Conference on Learning Representations ICLR (2023), https://openreview.net/forum?id=WlbG820mRH-, to appear
9. Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., Yu, P.S.: A comprehensive survey on graph neural networks. IEEE Trans. Neural Networks Learn. Syst. (2021). https://doi.org/10.1109/TNNLS.2020.2978386

On Q-Learning Routes in Uncertain Delay-Tolerant Networks

Pedro R. D'Argenio[1,2], Juan A. Fraire[1,3], Arnd Hartmanns[4], and Fernando Raverta[1,2]

[1] CONICET, Córdoba, Argentina
[2] Universidad Nacional de Córdoba, Córdoba, Argentina
[3] INRIA, Lyon, France
[4] University of Twente, Enschede, The Netherlands

Problem context. Delay-tolerant networks (DTNs) are time-evolving networks lacking continuous and instantaneous end-to-end connectivity [3,9]. They in particular appear in deep-space [1] and near-Earth communication [2]. DTN communication protocols work with data *bundles* that flow from node to node in a store-carry-and-forward fashion. The possible communication episodes—the *contacts*—and their reliabilities (with message loss randomly resulting from disturbances, hardware failures, imprecise planning, etc.) are known a priori. To maximize the end-to-end delivery probability, a bounded network-wide number of message copies are allowed. The resulting multi-copy routing optimization problem is naturally modelled as a Markov decision process with distributed information. Consider the example contact plan with nodes A, B, C, and D on the right. It spans five time slots, t_0 to t_4. The contacts in each slot are depicted by an arrow labelled with the contact failure probability. Suppose we want to transmit a bundle from A to D, allowing two copies throughout the network. A state of the MDP consists of the number of copies that each node holds at a given time slot. Initially, node A has both copies, and three options: send one copy to B, send two copies to B, or keep the two copies. The first option maximises the end-to-end delivery probability. Then, in t_2, node A can store its remaining copy or send it to C. The optimal decision is to store if C has the other copy and send otherwise. However, A cannot know whether the second copy is in B or C. This type of problem, in which decisions in an MDP associated to a distributed system may only be based on *local* knowledge, is known as *distributed scheduling* [4,10,11], and is closely related to the model of decentralised partially observable Markov decision processes (Dec-POMDPs) [14].



Existing approaches. The optimal *global* scheduler can be computed using any probabilistic model checker (PMC) [8,12,13]. Yet these tools are not specialised to DTNs, running into scalability limits, and only solve the global-information case. The two existing MDP-based approaches for optimal DTN routing under uncertain contact plans with *local* information are RUCoP [15] and lightweight scheduler sampling (LSS) [5,7]. We recently evaluated them [6] in terms of the obtained message delivery probabilities as well as the computational effort using

	PMC			Q-learning (100 k episodes)						LSS (100 k schedulers)					
	global			global			local			global			local		
	p	$\|S\|$	t	p	$\|S\|$	t	p	$\|S\|$	t	p	$\|S\|$	t	p	$\|S\|$	t
example	0.493	131	0 s	0.489	37	1 s	0.456	14	1 s	0.495	0	6 s	0.472	0	6 s
walker	0.4375	64 k	1 s	0.375	< 25 k	41 s	0.440	< 250	46 s	0.246	0	170 s	0.379	0	269 s

an extensive benchmark set. The two approaches are both useful to solve the problem, but expose a trade-off between scalability and solution quality. While RUCoP delivers routes with 3-10% higher delivery probabilities, LSS needs only 30-40% of the computational effort. LSS also delivers a viable solution for the most complex topologies, on which RUCoP runs out of memory.

Q-learning DTN routes. RUCoP and LSS are extremal in the sense that the former is memory-limited due to is exhaustive state space exploration, while the latter is time-limited as a purely simulation-based approach that needs to sample as many schedulers as possible. We are currently experimenting with the use of reinforcement learning techniques as a middle ground that stores information about relevant states only, but hopefully achieves better probabilities than LSS due to the extra storage (albeit offering the same weak formal guarantee only). For the global-information case, we implemented standard table-based Q-learning on the MDP's state space. In the local-information case, we have a multi-agent reinforcement learning (MARL) problem. We currently solve this in a straightforward way by equipping every node in the DTN with its own table-based Q-learner that operates on its local states, i.e. state vectors comprising only the values of those model variables visible to the respective node. The good-for-distributed-scheduling restriction [5, Sect. 3.2] that our MDPs satisfy entails that each node N's view is not corrupted by actions chosen by other nodes performing transitions between any pair of states locally different for N.

Preliminary results. We have exercised our current prototype implementation on the cases with unreliable communication considered in [5]. The table on top of this page shows the obtained message delivery probabilities p, memory requirements (in terms of states stored $|S|$), and runtimes t of the different methods. For the example contact plan (as shown on the first page), the optimal local-information probability is 0.4645. We see that the distributed Q-learning approach delivers outstanding (essentially optimal with global information, whereas the local-information *walker* optimum is unknown) results while requiring very little memory due to storing local states. The local state spaces are small, with the state space explosion resulting from their parallel composition.

Open questions. Our current approach to MARL using concurrent, independent Q-learning for the individual nodes is arguably naïve. There is a large corpus of work on MARL, and different restrictions of Dec-POMDPs that lead to vastly different complexities and limitations of learning approaches [14, Sect. 15.5.3]. It is not currently clear where our setting of simple distributed schedulers for good-for-distributed-scheduling MDPs lies. We hope to engage with the LiVe community to help us answer these questions, and learn about state-of-the-art alternatives to our current approach.

References

1. Burleigh, S., Hooke, A., Torgerson, L., Fall, K., Cerf, V., Durst, B., Scott, K., Weiss, H.: Delay-tolerant networking: An approach to interplanetary Internet. Comm. Mag. **41**(6), 128–136 (Jun 2003). https://doi.org/10.1109/MCOM.2003.1204759
2. Caini, C., Cruickshank, H., Farrell, S., Marchese, M.: Delay- and disruption-tolerant networking (DTN): An alternative solution for future satellite networking applications. Proceedings of the IEEE **99**(11), 1980–1997 (Nov 2011). https://doi.org/10.1109/JPROC.2011.2158378
3. Cerf, V., Burleigh, S., Hooke, A., Torgerson, L., Durst, R., Scott, K., Fall, K., Weiss, H.: Delay-tolerant networking architecture. RFC 4838, RFC Editor (April 2007), http://www.rfc-editor.org/rfc/rfc4838.txt
4. Cheung, L., Lynch, N.A., Segala, R., Vaandrager, F.W.: Switched PIOA: parallel composition via distributed scheduling. Theor. Comput. Sci. **365**(1-2), 83–108 (2006). https://doi.org/10.1016/j.tcs.2006.07.033, https://doi.org/10.1016/j.tcs.2006.07.033
5. D'Argenio, P.R., Fraire, J.A., Hartmanns, A.: Sampling distributed schedulers for resilient space communication. In: 12th International NASA Formal Methods Symposium (NFM). LNCS, vol. 12229, pp. 291–310. Springer (2020). https://doi.org/10.1007/978-3-030-55754-6_17
6. D'Argenio, P.R., Fraire, J.A., Hartmanns, A., Raverta, F.D.: Comparing statistical and analytical routing approaches for delay-tolerant networks. In: 19th International Conference on Quantitative Evaluation of Systems (QEST). LNCS, vol. 13479, pp. 337–355. Springer (2022). https://doi.org/10.1007/978-3-031-16336-4_17
7. D'Argenio, P.R., Legay, A., Sedwards, S., Traonouez, L.M.: Smart sampling for lightweight verification of Markov decision processes. Int. J. Softw. Tools Technol. Transf. **17**(4), 469–484 (2015). https://doi.org/10.1007/s10009-015-0383-0
8. Dehnert, C., Junges, S., Katoen, J.P., Volk, M.: A Storm is coming: A modern probabilistic model checker. In: 29th International Conference on Computer Aided Verification (CAV). LNCS, vol. 10427, pp. 592–600. Springer (2017). https://doi.org/10.1007/978-3-319-63390-9_31
9. Fall, K.: A delay-tolerant network architecture for challenged Internets. In: 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications. pp. 27–34. SIGCOMM'03, ACM, New York, NY, USA (2003). https://doi.org/10.1145/863955.863960
10. Giro, S.: On the Automatic Verification of Distributed Probabilistic Automata with Partial Information. Ph.D. thesis, Universidad Nacional de Córdoba, Argentina (2010)
11. Giro, S., D'Argenio, P.R., Fioriti, L.M.F.: Distributed probabilistic input/output automata: Expressiveness, (un)decidability and algorithms. Theor. Comput. Sci. **538**, 84–102 (2014). https://doi.org/10.1016/j.tcs.2013.07.017
12. Hartmanns, A., Hermanns, H.: The Modest Toolset: An integrated environment for quantitative modelling and verification. In: 20th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS). LNCS, vol. 8413, pp. 593–598. Springer (2014). https://doi.org/10.1007/978-3-642-54862-8_51
13. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: Verification of probabilistic real-time systems. In: 23rd International Conference on Computer Aided Verification (CAV). LNCS, vol. 6806, pp. 585–591. Springer (2011)

14. Oliehoek, F.A.: Decentralized POMDPs. In: Wiering, M.A., van Otterlo, M. (eds.) Reinforcement Learning, Adaptation, Learning, and Optimization, vol. 12, pp. 471–503. Springer (2012). https://doi.org/10.1007/978-3-642-27645-3_15
15. Raverta, F.D., Demasi, R., Madoery, P.G., Fraire, J.A., Finochietto, J.M., D'Argenio, P.R.: A Markov decision process for routing in space DTNs with uncertain contact plans. In: 6th IEEE International Conference on Wireless for Space and Extreme Environments (WiSEE). pp. 189–194 (2018). https://doi.org/10.1109/WiSEE.2018.8637330

Towards Formal Specification of Reinforcement Learning

Mahsa Varshosaz[1], Mohsen Ghaffari[1],
Einar Broch Johnsen[2], and Andrzej Wąsowski[1]

[1] IT University of Copenhagen, Copenhagen, Denmark,
[2] University of Oslo, Oslo, Norway

The development process for reinforcement learning (RL) applications is exploratory rather than systematic, which reduces reuse between applications and increases the chances of introducing errors into particular implementations, lowering trustworthiness and effectiveness. This is especially important in areas where reinforcement learning is used to control physical devices (e.g., embedded or cyber-physical systems, and robots).

RL research often focuses on evaluating the quality of obtained policies. However, the results of even the most carefully designed evaluation experiments have little value unless the evaluated methods are implemented correctly. Despite this, techniques and tools for systematic quality assurance of RL applications are rare in the field.

In a RL problem, an agent learns from experience to achieve a goal. The agent interacts with an environment, receiving rewards for its actions. The goal is to learn a policy which maximizes the cumulative reward. As an example, consider a car moving that needs to stop before reaching a static obstacle (Fig. 1). A RL algorithm for the car should learn to halt before the obstacle and to avoid unnecessary sharp braking. Throughout the learning process, the car starts from different states, i.e., positions and velocities, and selects actions, values of deceleration, and observes the reward and change of state. Sometimes the car stops before the obstacle, sometimes it crashes. The algorithm estimates the long-term effect of taking an action in a specific state. The estimate is updated by considering new rewards observed from the agent's interactions with the environment. This update step constitutes the core of each RL algorithm; the details of how and when to perform the update vary depending on the specific algorithm.

In this talk, we present our ongoing work on a direct formal specification of correctness for reinforcement learning problem definitions and the *learning algorithms* themselves, as opposed to the policies that they output. We show how to use the resulting specification to automatically test reinforcement learning systems and show an example of a property one can verify using the specification. We focus on table-based temporal difference (TD) learning methods [2], a large and well-established class of *model-free* methods. Instead of needing a complete model of the environment, these methods learn from experience by sampling executions. They update an estimate of state action values during an execution using prior estimates (bootstrapping).

We pay special attention to the update step in the algorithms commonly described using update diagrams [2]. We present a compositional, domain-specific language for describing update diagrams (and hence, update steps), which are generally defined informally in the RL literature. As an example, SARSA [1] is a TD learning algorithm which learns an optimal policy by considering a number of episodes. Each episode is a sequence of states S_i and actions A_i, generated based on the agent and environment

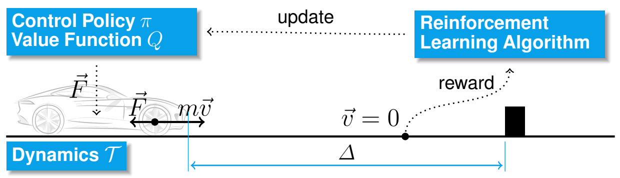


Fig. 1: Example: A car moves with velocity $\vec{v}$ towards a fixed obstacle at distance Δ, learning how to brake. The control policy chooses a deceleration with which to brake. The agent receives a reward based on the location where stopped and updates the policy

interaction, leading to a final state. In table-based SARSA, a function, Q, is used to assign long term reward values to each pair of state and action. The goal of SARSA is to obtain an optimal policy by updating the Q function iteratively based on the rewards observed in the episodes. In each step of the algorithm, an action is selected using an ε-greedy policy, that is A_t that maximises the Q value in the current state S_t is selected (with a probability 1-ε). The value for the pair of state and action is updated using the difference of the old estimate and the new estimate obtained by considering the new observed reward and the Q value for the next state S_{t+1} and selected action A_{t+1} based on the policy. The rate by which this difference effects the old estimate is called learning rate (α). The update step of SARSA is commonly described as follows (γ defines the long term reward discount) [2]:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) - \alpha(R_t + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t))$$

Implementing the update step of the algorithm using the above description requires additional knowledge and understanding intricacies which are implicit such as the relation of the policy and Q function. This is more problematic for algorithms that involve similar updates but with multiple steps. To capture each component involved in the update step of the TD algorithms, we provide a term language and a denotational semantics to precisely define the meaning of each step. As an example the term describing SARSA in the provided language is $\mathsf{sample}^\gamma\ \mathsf{Update}^\alpha\ \mathsf{sample}$, where each of sample^γ and Update^α, represent sampling for selection of actions and updating the Q values, with formally defined semantics. Due to commonalities between steps of TD learning algorithms, this modular specification can be used for testing different TD learning algorithms. We discuss how providing this formal semantics of TD learning components enables reuse of partial specifications between RL algorithms and creating a configurable test harness that can be tailored and extended to different applications.

References

1. Rummery, G.A., Niranjan, M.: On-line Q-learning using connectionist systems. Technical Report CUED/F-INFENF/TR (1994), https://cir.nii.ac.jp/crid/1573668924277769344
2. Sutton, R.S.: Learning to predict by the methods of temporal differences. Mach. Learn. **3**(1), 9–44 (1988). https://doi.org/10.1023/A:1022633531479

Learning Through Imitation by using Formal Verification

Avraham Raviv, Eliya Bronshtein, Or Reginiano, Michelle Aluf-Medina, and Hillel Kugler

Faculty of Engineering, Bar-Ilan University, Ramat Gan, Israel
{ravivav1, hillelk}@biu.ac.il

Abstract. Reinforcement-Learning-based solutions have achieved many successes in numerous complex tasks. However, their training process may be unstable, and achieving convergence can be difficult, expensive, and in some instances impossible. We propose herein an approach that enables the integration of strong formal verification methods in order to improve the learning process as well as prove convergence. During the learning process, formal methods serve as experts to identify weaknesses in the learned model, improve it, and even lead it to converge. By evaluating our approach on several common problems, which have already been studied and solved by classical methods, we demonstrate the strength and potential of our core idea of incorporating formal methods into the training process of Reinforcement Learning methods.

1 Introduction

Reinforcement Learning (RL) [1] is a paradigm of machine learning focused on training intelligent agents leading to a strategy of selecting actions in an environment to maximize their cumulative reward. RL involves an agent, a set of states ($\mathcal{S}$), and a set of actions per state ($\mathcal{A}$). The transition of an agent from state to state is effected by performing an action $a \in \mathcal{A}$. Each (s, a) pair, i.e. action execution in a specific state, provides the agent with a reward (r), represented by a numerical score.

The agent's goal is to maximize total reward over a complete trajectory, a set of N steps defined by $\tau = \{a_0, s_0, r_0, ..., a_n, s_n, r_n\}$. Formally, the goal is to find the steps that maximize the following expression:

$$\sum_{t=1}^{N} r_t$$

Despite its success in solving real-world problems, such as autonomous driving [2], natural language processing [3], genetics algorithms [4], and more, RL faces several significant challenges. The algorithms do not use pure labeled data, so they have to explore it themselves. A major challenge with RL solutions is that the state distribution changes as policies change. Using some exploration policy, sampling many states and actions, and then using that model to improve the policy will result in a revised distribution over states. The model may have been fairly accurate for the previous distribution, but there is no guarantee that it will be accurate for the updated distribution as well.

Rather than trying to learn only from sparse rewards or manually specifying a reward function, Imitation Learning [5] provides us with a set of demonstrations from an expert (typically a human). Following and imitating the expert's decisions, the agent attempts to learn the optimal policy. Behavioral cloning [6] is the simplest type of imitation learning, which uses supervised learning to mimic the expert's policy. Using imitation learning methods tackles the mentioned challenge since learning from an expert reduces the changes in the environmental distribution and allows for more stable and accurate learning. It is unfortunate that experts can be expensive, and sometimes they are simply not available. One can use Inverse Learning [7] to reduce the dependencies on experts, but it still requires prior data.

In a recent published paper [8], we propose a method of learning through imitation in which formal verification tools serve as experts. To demonstrate the effectiveness of our approach, we examine several common tasks in the field of RL. At the workshop we will discuss new empirical evaluation beyond results reported in [8] including handling of stochastic agents.

References

1. Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.
2. B Ravi Kiran, Ibrahim Sobh, Victor Talpaert, Patrick Mannion, Ahmad A Al Sallab, Senthil Yogamani, and Patrick Pérez. Deep reinforcement learning for autonomous driving: A survey. *IEEE Transactions on Intelligent Transportation Systems*, 2021.
3. Caiming Xiong, Victor Zhong, and Richard Socher. Dcn+: Mixed objective and deep residual coattention for question answering. *arXiv preprint arXiv:1711.00106*, 2017.
4. Daniel S Weile and Eric Michielssen. Genetic algorithm optimization applied to electromagnetics: A review. *IEEE Transactions on Antennas and Propagation*, 45(3):343–353, 1997.
5. Ahmed Hussein, Mohamed Medhat Gaber, Eyad Elyan, and Chrisina Jayne. Imitation learning: A survey of learning methods. *ACM Computing Surveys (CSUR)*, 50(2):1–35, 2017.
6. Dean A Pomerleau. Alvinn: An autonomous land vehicle in a neural network. *Advances in neural information processing systems*, 1, 1988.
7. Pieter Abbeel and Andrew Y Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, page 1, 2004.
8. Avraham Raviv, Eliya Bronshtein, Or Reginiano, Michelle Aluf-Medina, and Hillel Kugler. Learning through imitation by using formal verification. In Leszek Gasieniec, editor, *SOFSEM 2023: Theory and Practice of Computer Science - 48th International Conference on Current Trends in Theory and Practice of Computer Science, SOFSEM 2023, Nový Smokovec, Slovakia, January 15-18, 2023, Proceedings*, volume 13878 of *Lecture Notes in Computer Science*, pages 342–355. Springer, 2023.

Safe Policy Improvement for POMDPs

Marnix Suilen, Thiago D. Simão, Nils Jansen

Radboud University, Nijmegen, The Netherlands

Reinforcement learning (RL) is the standard approach to solve sequential decision-making problems when environment dynamics are unknown [8]. By interacting with the environment and receiving feedback in the form of a reward, an agent learns to optimize its behavior. However, direct interaction with the environment as done in the standard RL paradigm, is not always desirable or possible, as is the case in, for instance, robotics or healthcare applications.

In *offline RL*, the agent does not interact with the environment but instead is given access to a fixed data set of past interactions with the environment using a so-called *behavior policy*. Using this data set and behavior policy, the agent must learn a new policy. Safe policy improvement (SPI) addresses the problem of how to learn such a new policy such that it outperforms the behavior policy with a high probability [9]. More precisely, SPI methods provide a formal probabilistic guarantee that states that with high probability the newly learned policy has a higher value than the behavior policy, up to a small error.

Our Contribution: SPI in POMDPs

SPI has been studied extensively over the past couple of years, leading to efficient approaches that exploit the given knowledge of the behavior policy [3] or the structure in the underlying environment [6]. Yet, all previous SPI approaches assume the underlying environment is modeled as a fully observable Markov decision process (MDP) [4].

In our recently published work [7], we present a first approach to the SPI problem under *partial observability*, that is, environments modeled as partially observable MDPs (POMDPs) [2]. Specifically, we apply the SPI with baseline bootstrapping (SPIBB) [3] in a POMDP setting.

We note that a POMDP is in fact a fully observable, infinitely growing, *history MDP* [5]. In this history MDP, states are sequences of observations and actions that encode the agent's past interactions. Directly applying SPIBB or other SPI algorithms to this history MDP is practically infeasible since the state space is infinite. We deal with this problem in the following way.

First, we represent finite-memory policies as finite-state controllers (FSCs). We choose some memory size for our FSC, and then estimate the finite product of that FSC and the data set. This gives us a maximum-likelihood estimate of a finite part of the (infinite) history MDP. This finite part is a fully observable MDP on which we can run standard safe policy improvement methods such as SPIBB [3]. This approach is amenable to any POMDP, yet, in order to preserve the theoretical guarantees from SPIBB, we need the following assumption.

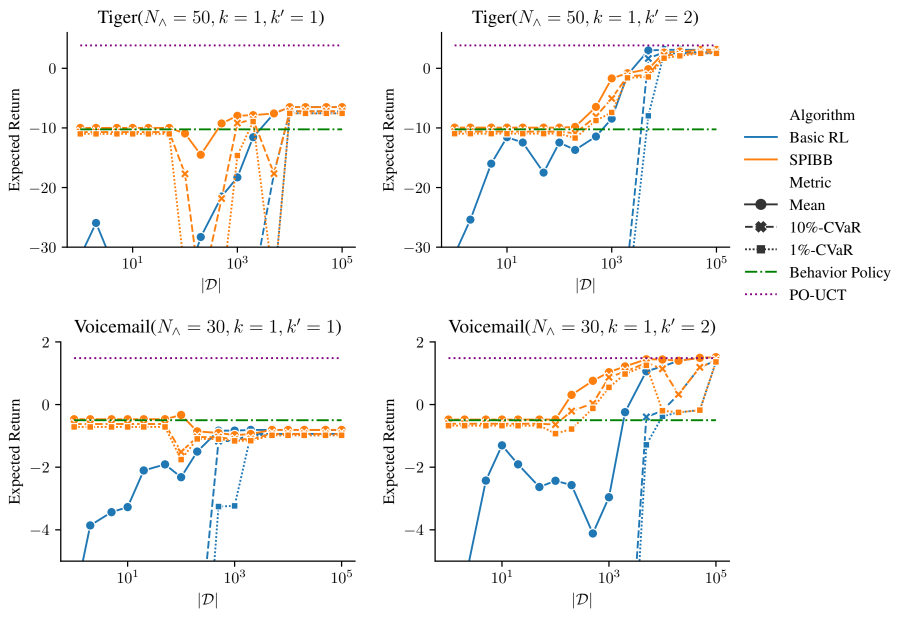


Fig. 1: SPI on the Tiger problem [2] and the Voicemail problem [10].

If we assume the chosen amount of memory is sufficient for an optimal policy in our POMDP, we have the aforementioned probabilistic improvement guarantee on the resulting policy. We formalize this assumption via a stochastic bisimulation [1] that states that the infinite histiory MDP has a finite bisimulation quotient. This assumption is, however, unlikely to hold in practice. As such, we evaluate our method on POMDP environments where the assumption does not hold and still observe good performance in practice.

Experimental Results

We evaluate our method on several well-established POMDP environments across different hyperparameters. An excerpt is presented in Figure 1.

The baseline policies used are memoryless ($k = 1$), and for the target policies we consider memoryless policies ($k' = 1$) and finite-memory policies that look one observation back ($k' = 2$). Note that these memory sizes are insufficient to satisfy our assumption in these environments. We observe, however, that already for small finite-memory policies ($k' = 2$) we are able to infer well-performing policies in both environments for relatively small data sets already ($|\mathcal{D}| \geq 10^3$). Increasing the memory size of the policies also makes them more stable, as seen in the conditional value at risk (CVaR) values ($k' = 1$ versus $k' = 2$).

Furthermore, we see the value of applying SPI methods such as SPIBB when comparing with a basic RL approach, as SPI methods will stick to the baseline policy, instead of learning from scratch.

References

1. Givan, R., Dean, T.L., Greig, M.: Equivalence notions and model minimization in markov decision processes. Artif. Intell. **147**(1-2), 163–223 (2003)
2. Kaelbling, L.P., Littman, M.L., Cassandra, A.R.: Planning and acting in partially observable stochastic domains. Artif. Intell. **101**(1-2), 99–134 (1998)
3. Laroche, R., Trichelair, P., des Combes, R.T.: Safe policy improvement with baseline bootstrapping. In: ICML. Proceedings of Machine Learning Research, vol. 97, pp. 3652–3661. PMLR (2019)
4. Puterman, M.L.: Markov Decision Processes: Discrete Stochastic Dynamic Programming. Wiley Series in Probability and Statistics, Wiley (1994)
5. Silver, D., Veness, J.: Monte-carlo planning in large pomdps. In: NIPS. pp. 2164–2172. Curran Associates, Inc. (2010)
6. Simão, T.D., Spaan, M.T.J.: Safe policy improvement with baseline bootstrapping in factored environments. In: AAAI. pp. 4967–4974. AAAI Press (2019)
7. Simão, T.D., Suilen, M., Jansen, N.: Safe policy improvement for pomdps via finite-state controllers. In: AAAI. AAAI Press (2023)
8. Sutton, R.S., Barto, A.G.: Reinforcement learning - an introduction. Adaptive computation and machine learning, MIT Press (1998)
9. Thomas, P.S., Theocharous, G., Ghavamzadeh, M.: High confidence policy improvement. In: ICML. JMLR Workshop and Conference Proceedings, vol. 37, pp. 2380–2388. JMLR.org (2015)
10. Williams, J.D., Young, S.J.: Partially observable markov decision processes for spoken dialog systems. Comput. Speech Lang. **21**(2), 393–422 (2007)

1–2–3–Go! Explainable Policies for Arbitrarily Large Parametrized Markov Decision Processes via Decision-Tree Generalization

Muqsit Azeem[1], Alexandros Evangelidis[1], Jan Křetínský[1], Mohammadsadegh Mohagheghi[2], Stefanie Mohr[1], and Maximilian Weininger[1]

[1] Technical University of Munich, Munich, Germany
[2] Vali-e-Asr University of Rafsanjan, Rafsanjan, Iran

Abstract. Despite the advances in probabilistic model checking, the scalability of the verification methods as well as the explainability of the results remains limited. In particular, when instantiating parametrized Markov decision processes (MDPs) even with moderate values, the state space often becomes extremely large; similarly, the policy, if any is produced, can be too large an object to be understood, or even implemented. To tackle these issues, we propose a simple learning-based approach to construct often near-optimal policies and their interpretable representations as decision trees, even for huge MDPs. The main idea is to generalize optimal strategies from smaller to larger instantiations, using decision-tree learning.

Markov decision processes (MDPs) are *the* model for combining non-determinism and probabilistic uncertainty. MDPs come with a rich theory and algorithmics developed over several decades with mature verification tools arising 20 years ago [KNP02] and proliferating since then [BHK$^+$20]. Despite all this effort, the *scalability* of the methods is considerably worse than those used for the verification of non-deterministic systems with no probabilities, even for basic problems.

Synthesizing *practically good policies* is often sufficient and also the only way when the systems are too large. Consequently, many *best-effort* approaches, such as reinforcement learning (RL) [SB98] and lightweight statistical model checking [BDHS18] simply try to find a good policy while giving only empirically good chances to be close to the optimum. This is perfectly sufficient in the setting of policy synthesis, where a good enough, but not necessarily optimal, controller is sought. However, the quality of the results rely on certain assumptions: RL results suffer when the rewards in the model are sparse (e.g., in the case of reachability) and lightweight statistical model checking requires near-optimal policies to be frequent. The mentioned techniques yield policies that are: (i) far from optimum when the system is extremely large; and (ii) not explainable. We tackle the former issue *through* tackling the latter.

Looking at the structure of large MDPs in standard benchmark sets [HKP$^+$19], it is not so surprising that their huge sizes are typically not due to astronomically large human-written code, but rather because the MDPs are *parametrized* (e.g., by the number of participants in a protocol) and then instantiated with large values.

This paper thus proposes a new technique for dealing with large parametrized MDPs. We focus on probabilistic reachability (i) for simplicity and (ii) because it is a fundamental building block for many other problems.

The *main idea* is very simple: First, we construct optimal policies for the given parametrized MDP by instantiating it with smaller numbers, and then we generalize this information to the instantiation with the desired larger number. It is important to note that we do not focus on generalizing the values of the states across different parametrizations, rather we generalize the corresponding decisions (i.e., the policy itself). This is because, while there is some regularity in the state space, it can be hard to capture. To this end, we need a compact, symbolic representation of a policy, which translates to policies over *all* instantiations. While binary decision diagrams (BDDs) may be an obvious candidate, it has been shown that *decision trees (DTs)* are more appropriate if adequately used [BCC$^+$15,AJK$^+$21], since they capture naturally the structure of the decisions and provide an explainable policy. Moreover, *because* the policies have this symbolic form, they can be applied to any instantiation; and *because* they capture the essence of the decisions, not just a list of state-action pairs, they are able to generalize well. Our new "generalization" policy synthesis method, which has its own merit as it provides another means for scalability, thus works *via* explainability (utilizing the literature on small decision-tree representation). Explainable representations are important for further steps in the system development, such as validating the model, implementing the policy, certifying the whole system, or patching it after deployment, see e.g., [AJK$^+$21].

References

[AJK+21] Pranav Ashok, Mathias Jackermeier, Jan Kretínský, Christoph Weinhuber, Maximilian Weininger, and Mayank Yadav. dtcontrol 2.0: Explainable strategy representation via decision tree learning steered by experts. In *TACAS (2)*, volume 12652 of *Lecture Notes in Computer Science*, pages 326–345. Springer, 2021.

[BCC+15] Tomás Brázdil, Krishnendu Chatterjee, Martin Chmelik, Andreas Fellner, and Jan Kretínský. Counterexample explanation by learning small strategies in markov decision processes. In Daniel Kroening and Corina S. Pasareanu, editors, *Computer Aided Verification - 27th International Conference, CAV 2015, San Francisco, CA, USA, July 18-24, 2015, Proceedings, Part I*, volume 9206 of *Lecture Notes in Computer Science*, pages 158–177. Springer, 2015.

[BDHS18] Carlos E. Budde, Pedro R. D'Argenio, Arnd Hartmanns, and Sean Sedwards. A statistical model checker for nondeterminism and rare events. In *TACAS (2)*, volume 10806 of *Lecture Notes in Computer Science*, pages 340–358. Springer, 2018.

[BHK+20] Carlos E. Budde, Arnd Hartmanns, Michaela Klauck, Jan Kretínský, David Parker, Tim Quatmann, Andrea Turrini, and Zhen Zhang. On correctness, precision, and performance in quantitative verification - QComp 2020 competition report. In *ISoLA (4)*, volume 12479 of *Lecture Notes in Computer Science*, pages 216–241. Springer, 2020.

[HKP+19] Arnd Hartmanns, Michaela Klauck, David Parker, Tim Quatmann, and Enno Ruijters. The quantitative verification benchmark set. In *TACAS (1)*, volume 11427 of *Lecture Notes in Computer Science*, pages 344–350. Springer, 2019.

[KNP02] Marta Z. Kwiatkowska, Gethin Norman, and David Parker. PRISM: probabilistic symbolic model checker. In *Computer Performance Evaluation / TOOLS*, volume 2324 of *Lecture Notes in Computer Science*, pages 200–204. Springer, 2002.

[SB98] Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning.* Cambridge, MA, USA, 1st edition, 1998.