Towards Graph Neural Networks for SMT Portfolios

Jan Hůla[1,2], David Mojžíšek[1], and Mikoláš Janota[2]

[1] University of Ostrava
{jan.hula21,mojzisek.work}@gmail.com
[2] Czech Technical University in Prague
mikolas.janota@gmail.com

Solvers for *Satisfiability Modulo Theories (SMT)* are the driving force behind software verification, software testing, or software synthesis [1, 2, 3, 4]. These applications often require repeated queries to an SMT solver. This means that the quick response time of the solver is paramount.

An SMT solver receives as input a formula and responds if it is satisfiabile or not. In the case of satisfiability, a satisfying interpretation (model) of the formula is also produced. Since the problem is generally undecidable, solvers often time out or give up. Due to this hardness of the problem, different heuristics may show very different per-instance behaviour in terms of runtime and ability to find a successful solution in SMT solving. This can lead to the scenario in which there is a single best solver on average, but an algorithm selecting the best solver for a specific instance of a problem can yield a better result [5, 6]. This per-instance behaviour is hard to understand for a human, but in accordance with the current trend [7, 8, 9], we aim to predict the best solver using ML methods.

We develop an approach to solver selection in the domain of SMT using a Graph Neural Network (GNN). In contrast to related methods, GNNs do not require manual feature design as they enable discovering relevant features in the raw data. We compare several architectural choices of GNNs which are trained to predict the performance of individual solvers in the chosen benchmarks. Rather than choosing only one solver with the best prediction, we choose n best solvers, order them by the predicted score, and delegate part of the time budget to each of them. We compare our approach to a baseline, which uses bag-of-words as a representation of each formula and gradient boosted trees as a predictor. In the selected benchmarks, we show an improvement over this baseline in terms of the number of solved problems and overall solving time.

GNNs are neural networks that process inputs structured as a graph. This makes them different from other types of neural networks such as Multi-layer Perceptrons, Recurrent Neural Networks, Convolutional Neural Networks, or Transformers, which do not assume any special structure of the input. For this reason, GNNs became popular for processing all kinds of formal structures such as logical expressions, which are naturally represented as trees or directed acyclic graphs.

Most often, additional meta-information for nodes within an input graph is available. For a specific node, this information is encoded as a feature vector of a fixed size. In our case, we use the mapping from symbols corresponding to a given node to one-hot vectors.

Each layer of a GNN updates the feature vectors of all nodes by transforming and aggregating the feature vectors of its neighbour nodes. After several steps of such feature vector transformation, the final single vector is obtained by pooling, see also Figure 1.

The advantage of using a GNN is that the trained transformations are applied locally to each node and the final aggregation operator does not require a specific number of inputs. It allows the graphs to have different number of nodes and structure. Therefore, the trained GNN is applicable to arbitrary graphs. GNN architectures differ in how they achieve the layer-level node feature vector transformation and aggregation. In this contribution, we compare a

Figure 1: The steps conducted during the creation of the input graph from a given SMT formula.

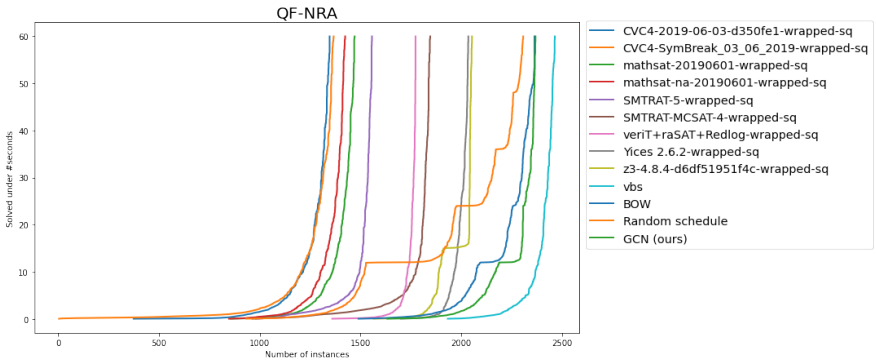

Figure 2: A cactus plot of one of our results on QF_NRA benchmark. The y-axis represents time and the x-axis the number of problems solved under the corresponding time. Virtual best solver is denoted by vbs and represents the upper bound of what could be achieved.

simple Graph Convolutional Network (GCN) [10], Graph Attention Network (GAT) [11], Graph Transformer [12] and Principal Neighbourhood Aggregation (PNA) [13].

We use GNNs for the regression task to predict different solvers runtimes for a specific instance of a problem. Rather than choosing only one solver with the best prediction, we choose n best solvers, order them by the predicted score, and delegate part of the time budget to each of them.

We test our GNN on 4 representative benchmarks: QF_NRA, UFNIA, UFNIA-CONF[1] and TPTP [14]. Figure 2 shows results for one of the considered families (non-linear arithmetic without quantifiers).

To summarize, our work has the following main contributions:

- It applies GNN to rank a portfolio of SMT solvers on a given instance according to suitability. To the best of our knowledge, this is the first time that GNN is applied in the context of SMT.
- The proposed approach schedules n best solvers rather than just picking the best one, which further improves the robustness of the approach.
- The experimental evaluation compares several GNN architectures. This will also be of use to other researchers that wish to apply GNN in the context of SMT for other tasks.

[1]We have collected the different strategies that the solver CVC5 uses to solve UFNIA formulas in the competition so we do not select the solver but the solving strategy.

References

[1] Clark W. Barrett, Roberto Sebastiani, Sanjit A. Seshia, and Cesare Tinelli. Satisfiability modulo theories. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satis ability*, volume 185 of *Frontiers in Arti cial Intelligence and Applications*, pages 825–885. IOS Press, 2009.

[2] Andrew Reynolds, Viktor Kuncak, Cesare Tinelli, Clark W. Barrett, and Morgan Deters. Refutation-based synthesis in SMT. *Formal Methods Syst. Des.*, 55(2):73–102, 2019.

[3] Leonardo de Moura and Nikolaj Bjørner. Applications and challenges in satisfiability modulo theories. In *Workshop on Invariant Generation (WING)*, volume 1, pages 1–11. EasyChair, 2012.

[4] Patrice Godefroid, Michael Y. Levin, and David A. Molnar. SAGE: whitebox fuzzing for security testing. *Commun. ACM*, 55(3):40–44, 2012.

[5] Kevin Leyton-Brown, Eugene Nudelman, Galen Andrew, Jim McFadden, and Yoav Shoham. A portfolio approach to algorithm selection. In *IJCAI*, volume 3, pages 1542–1543, 2003.

[6] Serdar Kadioglu, Yuri Malitsky, Ashish Sabharwal, Horst Samulowitz, and Meinolf Sellmann. Algorithm selection and scheduling. In *International Conference on Principles and Practice of Constraint Programming*, pages 454–469. Springer, 2011.

[7] Joseph Scott, Aina Niemetz, Mathias Preiner, Saeed Nejati, and Vijay Ganesh. MachSMT: a machine learning-based algorithm selector for SMT solvers. *Tools and Algorithms for the Construction and Analysis of Systems*, 12652:303, 2020.

[8] Daniel Selsam, Matthew Lamm, Benedikt Bünz, Percy Liang, Leonardo de Moura, and David L Dill. Learning a sat solver from single-bit supervision. *arXiv preprint arXiv:1802.03685*, 2018.

[9] Chris Cameron, Rex Chen, Jason Hartford, and Kevin Leyton-Brown. Predicting propositional satisfiability via end-to-end learning. In *Proceedings of the AAAI Conference on Arti cial Intelligence*, volume 34, pages 3324–3331, 2020.

[10] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[11] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[12] Vijay Prakash Dwivedi and Xavier Bresson. A generalization of transformer networks to graphs. *arXiv preprint arXiv:2012.09699*, 2020.

[13] Gabriele Corso, Luca Cavalleri, Dominique Beaini, Pietro Liò, and Petar Veličković. Principal neighbourhood aggregation for graph nets. *arXiv preprint arXiv:2004.05718*, 2020.

[14] Geoff Sutcliffe. The TPTP Problem Library and Associated Infrastructure. From CNF to TH0, TPTP v6.4.0. *Journal of Automated Reasoning*, 59(4):483–502, 2017.