# Graves-CPA: A Graph-Attention Verifier Selector (Competition Contribution)

Will Leeson(✉) and Matthew B. Dwyer

University of Virginia, Charlottesville VA 22903, USA
{will-leeson,matthewbdwyer}@virginia.edu

**Abstract.** Graves-CPA is a verification tool which uses algorithm selection to decide an ordering of underlying verifiers to most effectively verify a given program. Graves-CPA represents programs using an amalgam of traditional program graph representations and uses state-of-the-art graph neural network techniques to dynamically decide how to run a set of verification techniques. The Graves technique is implementation agnostic, but it's competition submission, Graves-CPA, is built using several CPAchecker configurations as its underlying verifiers.

**Keywords:** Software Verification · Graph Attention Networks · Graph Neural Networks · Algorithm Selection

## 1 Verification Approach

Graves-CPA is an algorithm selector for software verification based on graph neural network techniques. As the tool PeSCo [14] has shown, dynamic ordering of verification techniques can result in faster and more accurate verification. Computing an ordering on techniques dynamically will incur some runtime, but an effective ordering will oftentimes make this overhead insignificant in comparison to the time saved by using a more appropriate technique. Like most algorithm selectors, Graves-CPA uses machine learning to make its selections. However, it uses graph neural networks (GNNs) so it can represent programs using traditional program abstractions, such as abstract syntax trees (ASTs). Graves-CPA uses a variant of GNNs called Graph Attention Networks (GATs) [16]. GATs use a learned attention mechanism which is trained to learn the importance of edges in a given graph.

GNNs are an emerging field in machine learning. Traditional neural networks accept input vectors, which have a fixed size and a natural ordering on elements, but graphs, in general, have neither. GNNs avoid these issues by operating on individual nodes in the graph, instead of the graph as a whole [15]. Typically, the input to a GNN is the current representation of a node $v$ and a collation of the representations of its neighboring nodes. The output is then a new representation for $v$. This process is repeated independently for all nodes in the graph. Thus, the number of nodes in the graph and order in which they are processed is irrelevant.

The Graves technique is tool agnostic [11], meaning it can be trained to select from any set of verifiers. Our competition contribution selects an ordering from the techniques utilized by CPAchecker [3], similar to PeSCo in previous competitions.

To form its selection, Graves-CPA produces a graph representation of a given program, $G$, which is based on its AST with control flow, data flow, and function call and return edges added between the tree's nodes. The AST's nodes and edges ensure the semantics of the statements in the program are maintained. Control flow edges maintain the branching and order of execution between these statements. Data flow edges explicitly relate the definitions, uses, and interactions of values in the program. $G$ is passed to a GNN, consisting of a series of GATs, which outputs a graph feature vector This feature vector is finally passed to a fully connected neural network which decides the sequence in which Graves-CPA's suite of verification techniques are run.

## 2 System Architecture

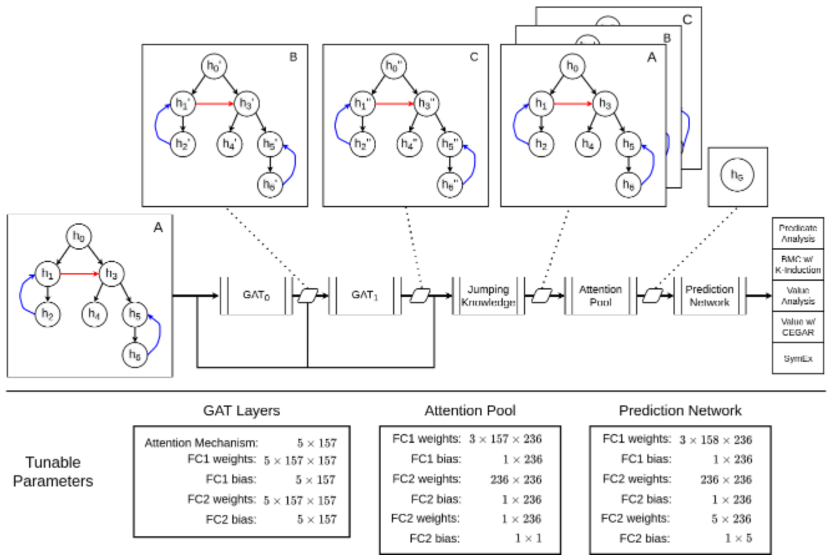
### 2.1 Graph Generation

To generate a graph from a program, Graves-CPA relies on the AST produced by the C compiler Clang [10]. Using a visitor pattern [9], Graves-CPA walks the AST to generate data flow edges and the edges of the program's Interprocedural Control Flow Graph (ICFG). Function call and return edges in the ICFG are those which can be determined purely syntactically. Using the ICFG and data flow edges, Graves-CPA produces additional data flow edges using the work-list reaching definition algorithm [1]. We limit the number of iterations of the reaching definition algorithm, making our data edges an under-approximation of possible data flow edges. Once this graph is generated, it is parsed into a list of nodes and several edge sets. Nodes represent the AST token which corresponds to them using a one-hot encoding. These nodes and edges are used as input to the GNN.

### 2.2 Prediction

To form a prediction, Graves-CPA uses a GNN, visualized in Figure 1, which consists of 2 GAT layers, a jumping knowledge layer [17], and an attention-based pooling layer [12]. The GAT layers are crucial to our technique. When propagating data through the graph, the attention mechanisms in each layer weights edges so information important to predictions is more prominent than superfluous data.

The jumping knowledge layer concatenates intermediate graph representations, denoted by A, B, and C, allowing the model to learn from each representation. The attention-based pooling layer calculates an attention value for each node in the graph. All nodes are weighted by their respective attention values and then summed together to form a graph feature vector. The combination of



GAT Layers

| | |
|---|---|
| Attention Mechanism: | 5 × 157 |
| FC1 weights: | 5 × 157 × 157 |
| FC1 bias: | 5 × 157 |
| FC2 weights: | 5 × 157 × 157 |
| FC2 bias: | 5 × 157 |

Attention Pool

| | |
|---|---|
| FC1 weights: | 3 × 157 × 236 |
| FC1 bias: | 1 × 236 |
| FC2 weights: | 236 × 236 |
| FC2 bias: | 1 × 236 |
| FC2 weights: | 1 × 236 |
| FC2 bias: | 1 × 1 |

Prediction Network

| | |
|---|---|
| FC1 weights: | 3 × 158 × 236 |
| FC1 bias: | 1 × 236 |
| FC2 weights: | 236 × 236 |
| FC2 bias: | 1 × 236 |
| FC2 weights: | 5 × 236 |
| FC2 bias: | 1 × 5 |

**Fig. 1.** GRAVES' uses a GNN comprised of 2 GAT layers, a Jumping Knowledge layer, and attention pooling layer. These layers produce a graph feature vector which a 3 layer prediction network uses to order verifiers for sequential execution. An in depth description of this architecture can be found in Leeson et al. [11].

GAT layers and the attention-based pool allows the network to weigh the importance of both edges and nodes when forming the graph feature vector. This feature vector is fed to a three layer neural network which decides the sequence of tool execution.

GRAVES-CPA was trained using data collected from running 5 configurations of the CPAchecker framework on the verification tasks from SV-COMP 2021. Labels for each configuration come from the SV-COMP score the configuration would receive for a given program minus a time penalty. Similar to CPAchecker's competition contribution, these configurations are symbolic execution [6], value analysis [7], value analysis with CEGAR [7], predicate analysis [5], and bounded model checking with k-induction [4]. To prevent GRAVES-CPA from overfitting to the SV-COMP benchmarks, we train on a subset of the dataset, only utilizing 20% of it. Like previous iterations of PeSCo, the network is trained to rank the configurations in the order in which they should be executed.

GRAVES-CPA uses the machine learning libraries PyTorch [13] and PyTorch-Geometric [8], an extension of PyTorch for graphs and other irregularly shaped data, to implement its machine learning components. GRAVES-CPA is implemented using a combination of Python, C++, and Java.

## 2.3 Execution

Using the ordering produced by the previous step, CPAchecker is run in a sequential fashion with each verification configuration. If a technique goes past a given time limit or fails to produce a result, the next technique is executed.

## 3 Strengths and Weaknesses

Graves-CPA operates on program graphs which are an abstraction of the program. Its underlying model uses this abstraction to learn what software patterns a particular verification technique excels at handling. This allows Graves-CPA to produce a dynamic ordering which should run techniques more equipped to the given problem first, reducing run time. In [11], the authors perform a qualitative study which suggests the network learns to rank verification techniques using program features an expert would use to decide between techniques.

In SV-COMP 2022 [2], there were 4,548 problems both Graves-CPA and CPA-checker reported the correct result. Graves-CPA's dynamic selection of CPA-checker's static configuration ordering allowed it to solve these problems 37 hours faster. Further, Graves-CPA was able to solve 142 problems that CPAchecker could not, due to resource constraints or other issues.

Machine learning relies on the fact that training data is representative of the real world. If this is not the case, the model can easily make poor predictions. These poor decisions can be seen in competition in the 559 instances where Graves-CPA chooses an ordering that doesn't produce the correct result, but CPAchecker does. In most of these instances, Graves-CPA runs out of resources or incorrectly predicts the remaining techniques will not produce a correct result.

## 4 Tool Setup and Configuration

Graves-CPA is built on the PeSCo codebase, which in turn is built on the CPAchecker codebase, and participates in the ReachSafety and Overall categories. It can be downloaded as a fork: https://github.com/will-leeson/cpachecker. Graves-CPA requires cmake, LLVM, either make or ninja, and ant (a CPAchecker dependency) to be built and the python libraries PyTorch and PyTorch-Geometric to be executed. To build the project, simply run the shell script `setup.sh` and add our graph generation tool, `graph-builder`, to your path. Now, you may verify a program with Graves-CPA using the command:

```
scripts/cpa.sh -svcomp22-graves -spec [prop.prp] [file.c]
```

## 5 Software Project and Contributions

Graves-CPA is an open source project developed by the authors at the University of Virginia. We would like to thank the team behind the PeSCo and CPAChecker tools for allowing us to build on their work.

## Acknowledgements

We would like to thank Hongning Wang for his advice on graph neural networks and prediction systems. This material is based in part upon work supported by the U.S. Army Research Office under grant number W911NF-19-1-0054 and by the DARPA ARCOS program under contract FA8750-20-C-0507.

## References

1. Aho, A.V., Sethi, R., Ullman, J.D.: Compilers, principles, techniques. Addison wesley **7**(8), 9 (1986)
2. Beyer, D.: Progress on software verification: SV-COMP 2022. In: Proc. TACAS (2). Springer (2022)
3. Beyer, D., Dangl, M.: Strategy selection for software verification based on boolean features. In: Margaria, T., Steffen, B. (eds.) Leveraging Applications of Formal Methods, Verification and Validation. Verification. pp. 144–159. Springer International Publishing, Cham (2018)
4. Beyer, D., Dangl, M., Wendler, P.: Boosting k-induction with continuously-refined invariants. In: International Conference on Computer Aided Verification. pp. 622–640. Springer (2015)
5. Beyer, D., Keremoglu, M.E., Wendler, P.: Predicate abstraction with adjustable-block encoding. In: Formal Methods in Computer Aided Design. pp. 189–197. IEEE (2010)
6. Beyer, D., Lemberger, T.: Cpa-symexec: efficient symbolic execution in cpachecker. In: Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering. pp. 900–903 (2018)
7. Beyer, D., Löwe, S.: Explicit-state software model checking based on cegar and interpolation. In: International Conference on Fundamental Approaches to Software Engineering. pp. 146–162. Springer (2013)
8. Fey, M., Lenssen, J.E.: Fast graph representation learning with PyTorch Geometric. In: ICLR Workshop on Representation Learning on Graphs and Manifolds (2019)
9. Johnson, R., Vlissides, J.: Design patterns. Elements of Reusable Object-Oriented Software Addison-Wesley, Reading (1995)
10. Lattner, C.: Clang: a c language family frontend for llvm, https://clang.llvm.org/
11. Leeson, W., Dwyer, M.B.: Algorithm selection for software verification using graph attention networks (2022), https://arxiv.org/abs/2201.11711
12. Li, Y., Tarlow, D., Brockschmidt, M., Zemel, R.: Gated graph sequence neural networks (2017)
13. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., Chintala, S.: Pytorch: An imperative style, high-performance deep learning library. In: Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., Garnett, R. (eds.) Advances in Neural Information Processing Systems 32, pp. 8024–8035. Curran Associates, Inc. (2019), https://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf
14. Richter, C., Wehrheim, H.: Pesco: Predicting sequential combinations of verifiers. In: International Conference on Tools and Algorithms for the Construction and Analysis of Systems. pp. 229–233. Springer (2019)
15. Scarselli, F., Gori, M., Tsoi, A.C., Hagenbuchner, M., Monfardini, G.: The graph neural network model. IEEE transactions on neural networks **20**(1), 61–80 (2008)
16. Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., Bengio, Y.: Graph attention networks. arXiv preprint arXiv:1710.10903 (2017)
17. Xu, K., Li, C., Tian, Y., Sonobe, T., ichi Kawarabayashi, K., Jegelka, S.: Representation learning on graphs with jumping knowledge networks (2018)