DepthGraphNet: Circuit Graph Isomorphism Detection via Siamese-Graph Neural Networks

Fin Amin
Department of Electrical & Computer Engineering
North Carolina State University
0000-0003-1212-4951
Raleigh, NC, United States
samin2@ncsu.edu

Soumyadeep Chatterjee
Department of Electrical & Computer Engineering
0009-0006-0851-0058
North Carolina State University
Raleigh, NC, United States
schatte5@ncsu.edu

Paul D. Franzon
Department of Electrical & Computer Engineering
North Carolina State University
Raleigh, NC, United States
paulf@ncsu.edu

***Abstract*—The circuit graph isomorphism (CGI) problem is a fundamental computational problem in EDA that involves determining the equivalence of two circuit graphs. This issue carries great significance in the design flow of circuits, as well as with intellectual property (IP) protection efforts–a solution to this issue would have significant implications for the design process, enabling a faster and more efficient design flow. In this work, we introduce DepthGraphNet, a Siamese-Graph Neural Network (SGNN) for the CGI problem. Additionally, we show that our proposed approach runs considerably faster than classical approaches while maintaining high accuracy on a real-world circuit dataset. Furthermore, we provide theorems which help deal with the time-consuming SGNN architecture design process.**

***Index Terms*—circuit graph isomorphism, graph neural networks, siamese neural networks, siamese-graph neural networks, graph neural network architecture search, message passing layers**

I. Introduction and Motivation

The circuit graph isomorphism (CGI) problem can be described more plainly as answering "Are these two netlists[1] (graphs) identical?" To this day, there does not exist a polynomial time algorithm for solving the CGI problem. In fact, the CGI problem is known to be in the NP time-complexity class. This problem is difficult because finding an isomorphism involves testing that there exists a bijection between all nodes and edges between two graphs–a computationally expensive procedure due to the combinatorial nature of the problem.

The CGI problem arises frequently - in fact, simply altering the ordering of a netlist seemingly produces a "different" graph. For this reason, a scalable and accurate solution would have a meaningful impact on EDA. In the realm of IP protection, CGI plays a pivotal role in detecting IP theft through reverse engineering efforts. From a security perspective, this method can also be used to identify malicious components, such as trojans. At a sub-circuit level, CGI enables automated identification of functionally similar modules, which can be optimized en masse to realize PPA (power-performance-area) improvements according to application-specific requirements. This functionality can further be extended to version control during the design process, as well as in functional verification and fault identification of varying designs.

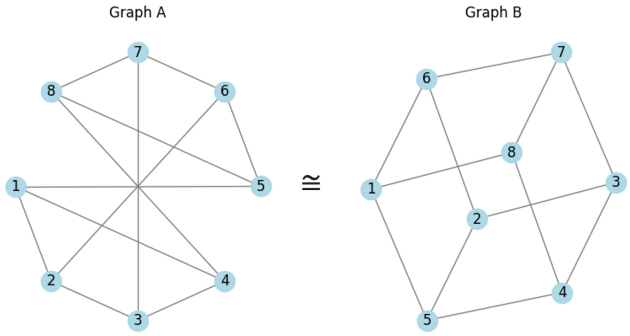

Fig. 1. Two graphs are isomorphic if they share the same structure. Notice that they may not appear to be similar [4].

In this work we investigate the deficits of existing classical approaches to the CGI problem. Afterwards, we show that SGNNs are a promising avenue for a scalable and accurate CGI detection mechanism.

Our notation is the following: U and K represent two graphs, with vertices V_u and V_k and edges E_u and E_k respectively. Additionally, we say that D_u is the diameter of graph U. We denote that two graphs are isomorphic/not isomorphic by $U \cong K$ and $U \ncong K$ respectively. When there exists a bijection between two graphs, we say that there exists a *mapping function*, M, that maps the attributes of U to K. In our work, we denote a false positive as incorrectly declaring two graphs to be isomorphic while they are not. Similarly, a false negative is declaring two graphs to be non-isomorphic while they are genuinely isomorphic.

Our contributions in this work are:

1) DepthGraphNet (DGN), a SGNN architecture for detecting circuit graph isomorphisms
2) Mathematical theorems which ameliorate the extremely time-consuming process of deciding the optimal SGNN architecture
3) Code for generating synthetic datasets, our DGN, and preprocessing scripts for adjusting the dataset of

[1]We use the terms netlists and graphs interchangeably.

GNN-RE for the inductive SGNN detection setting: github.com/FinAminToastCrunch
4) An analysis of why classical approaches and other popular machine learning architectures are insufficient for the CGI detection
5) A comparison of our architecture with a state-of-the art SGNN

II. Prior Work and Observations

In this section, we split our focus into two topics. We first explain classical approaches to the CGI problem and remark on their merits and drawbacks. Afterwards we introduce the relevant components of siamese neural networks (SNNs) and graph neural networks (GNNs) then explain how they can work in a cohesive manner to solve the CGI problem. Furthermore, we remark on why convolutional neural networks (CNNs) are not suitable for this problem.

A. Classical Approaches to CGI Detection

Existing approaches fall into two categories: exact brute-force algorithms and approximation algorithms. Exact solutions always give the right answer regardless of input whereas approximate algorithms may give false positives/false negatives. We compare our SGNN against VF2 [3], Hungarian [5] and Weisfeiler-Lehman (WL) CGI detection algorithms.

The VF2 algorithm works by gradually producing M through a recursive algorithm. The nodes of U and V are mapped at each recursive call by ensuring that each respective pair of nodes mapped between U and V produce a feasible mapping with respect to future iterations. The recursive look ahead procedure causes the VF2 to have a worst-case time complexity of $\mathrm{O}(|\mathrm{V}|!|\mathrm{V}|)$ but guarantees an exact solution.

The second approach uses the Hungarian/Kuhn–Munkres Algorithm; we check whether the two input graphs U_1 and V_2 are isomorphic by finding an optimal assignment of vertices with minimal cost. The Hungarian algorithm is applied to the costs matrix of the graphs, which is in turn derived as the difference between their adjacency matrices. If the total cost of the optimal assignments is zero, it implies that a perfect one-to-one vertex matching was found between the two graphs, and thus, they are considered isomorphic. However, this method may not always produce correct results for all cases, such as with complex graphs of same sizes. It provides an approximate solution with a worst-case time complexity tending to $O(|V|^3)$.

The last classical approach, the Weisfeiler-Lehman test [10], is an approximate algorithm. We refer readers to the original paper as the details of this algorithm are quite intricate.

B. Siamese Neural Networks

SNNs are a family of neural networks which excel at learning abstract measures of similarity/dissimilarity. Sample applications include speaker identification [13], signature verification [2] and facial identity verification [11]. The input to an SNN are two samples which we want to compare and the output is a value between 0 and 1, representing dissimilar and similar respectively. For example, in the context of facial identification, the inputs could be two faces and the output is the similarity between them. SNNs are trained using *discriminative learning paradigm*[2]–the details of which we explain in a later section.

The architecture of SNNs are unorthodox compared to other models. The architecture is comprised of two phases, feature extraction followed by similarity measure. The feature extractors $F_1(x_1) \mapsto z_1$ and $F_2(x_2) \mapsto z_2$, are trained to optimally extract the necessary features from inputs to aid similarity computation. Eponymously named, the feature extractors are **siamese**–they use the same weights and therefore can be thought of as being the same network (ie. $F_1 = F_2$). For this reason, we drop the subscript from F. In the aforementioned example uses (ie. facial recognition), the feature extraction functions are typically CNNs.

The similarity measurement function $S(z_1, z_2) \mapsto \hat{y}$ is responsible for predicting the likelihood that the samples x_1 and x_2 are similar. Dense layers are frequently employed to construct this function S. These layers process the extracted features z_1 and z_2 and produce an output $\hat{y}$, which, after training via the discriminative learning paradigm, represents the estimated similarity between the samples.

C. Graph Neural Networks

GNNs are a family of neural networks which specialize in working with graph data. An important aspect of GNNs is that they generate embeddings (or feature representations) for nodes and/or edges in a graph. Crucially, for **certain** GNN architectures these embeddings are invariant to the permutation of node labeling. In other words, relabeling the nodes of the graph won't change the embeddings, as long as the graph's structure remains the same. This property is called *invariance to graph isomorphism*, an extremely desirable property for our use case. However, although this property implies that two isomorphic graphs should produce the same embedding, it does not rule out two different graphs producing the same embedding as well. **This can cause false positives.**

Another relevant paradigm to GNNs is the message passing interface. As shown in [9], the message passing framework of GNNs is as follows:

$$x_n(t+1) = H_\theta(x_n(t), l_{neigh(n)}(t)). \tag{1}$$

Here, $l_{neigh(n)}(t)$ represents the *message* received from the node's neighbors, which is typically a function of their embeddings or labels. These embeddings, $x_n \in \mathbb{R}^{1\times q}$, encapsulate the node's semantic attributes in q dimensions, and when stacked together, form the matrix $X(t) \in \mathbb{R}^{|V|\times q}$. The function H_θ is a learnable aggregator that updates x_n by combining its current value with $l_{neigh(n)}$. Some example aggregation functions are taking the (θ-weighted) mean, max, sum, etc of $l_{neigh(n)}$. Xu et al. [12] details which aggregation functions maintain the *invariance to graph isomorphism* property.

[2]In lieu of discriminative learning, SNNs are sometimes trained using a triplet loss paradigm or contrastive loss paradigm. We exclude explaining this due to lack of relevance to our work.

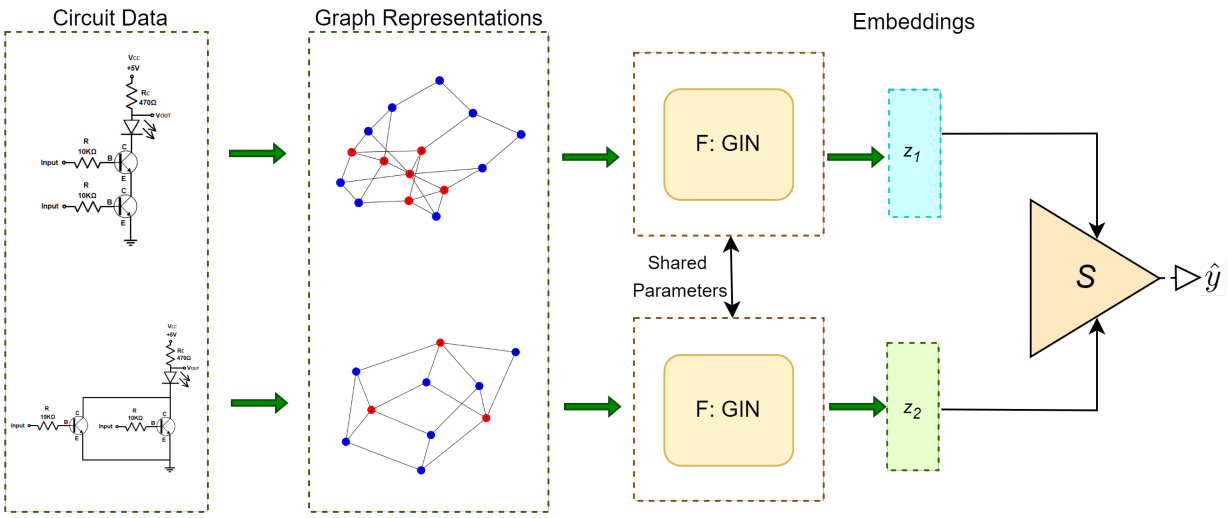

Fig. 2. The DepthGraphNet architecture allows us check graphs for isomorphism in the deductive setting. Specific implementation details are provided in section III-C. A salient feature of the DGN is that the number of message passing layers of F is decided by the graph diameter.

The number of layers,[3] L, in a GNN determines the 'reach' of a node's context, i.e., how far into its neighborhood the node can 'see' or gather information from. Each layer allows a node to incorporate information from nodes that are one step further away. Thus, deeper GNNs (larger L) can aggregate information from a larger neighborhood, which can be beneficial for understanding the global structure of the graph.

However, the choice of L must be made judiciously, considering the nature of the graph data and the task at hand. For tasks that primarily depend on local information (e.g., predicting properties of a node based on its immediate neighborhood), a smaller L may suffice. In contrast, tasks that require a global understanding of the graph (e.g., whole-graph classification) could benefit from a larger L.

D. Siamese Graph Neural Networks

SGNNs utilize graph neural networks for the feature extraction phase, rather than convolutional networks. That is, our graph neural network is F. The goal is to select a GNN architecture which optimally encodes the input graphs. As mentioned previously, some GNN architectures do not have the desirable invariance to graph isomorphism property, so we must ensure that we choose an architecture which has this.

According to Xu et al. [12], GNNs can be made as powerful as the Full Weisfeiler-Lehman test [10]. Using this and Theorem 1, we can conclude that we can search for GNN architectures by choosing those which are at least as expressive as the WL test.

Theorem 1. *If a Graph Neural Network (GNN) is as powerful as the Full Weisfeiler-Lehman test, then it possesses the property of Invariance to Graph Isomorphism.*

Proof. By definition, the WL test treats isomorphic graphs identically. This property is inherent to the design of the WL test. Therefore, any GNN that is as powerful as the WL test, i.e., it can distinguish between any pair of non-isomorphic graphs that the WL test can distinguish, should also exhibit this property. □

Another detail we must consider is the number of message passing layers, L, to use. This parameter is related to *which* WL test we want our GNN to be as powerful as. Prior works such as GNN-RE [1], recognize that the GNN architecture should be as powerful as the WL test, but do not consider *which WL test*. According to Theorem 2, our GNN must have $L = D$ to match the *Full* WL test.

Theorem 2. *For a Graph Neural Network (GNN) to be as powerful as the Full Weisfeiler-Lehman (WL) test, it must have as many message passing layers as the diameter, D, of the graph it is working with.*

Proof. The Weisfeiler-Lehman test is a node coloring procedure where, in each iteration, a node's color is updated based on the colors of its neighboring nodes. This is akin to a layer in a GNN, where a node's features are updated based on the features of its neighboring nodes.

After d iterations[4] of the WL test, a node's color could potentially be influenced by nodes that are d hops away. Hence, for a GNN to propagate information across a graph from a node that is d hops away, it needs at least d layers.

Considering the diameter of the graph, which is defined as the maximum shortest path between any two nodes (i.e., the maximum d over all pairs of nodes), for a GNN to potentially

[3]Some works refer to this as the "depth" of the GNN.

[4]The number of iterations of the WL test defines *which* WL test is being run. In our work, the prefix before "WL" denotes the number of iterations. Eg. 29-WL means a WL test which is run for 29 iterations.

aggregate information from any node in the graph (which is necessary for the GNN to emulate the full power of the WL test), it must have at least as many layers as the diameter of the graph. □

For CGI detection, GNNs are considerably more applicable compared to CNNs due to the aforementioned *invariance to graph isomorphism* property. However, existing works following the SGNN paradigm are scarce. The most similar existing architecture is *GREED* [8]–an architecture for computing graph/subgraph edit distance; in our experiments we show that DepthGraphNet outperforms *GREED*–the current state of the art SGNN for our problem setting. Another work, *GNN-RE* [1] attempts to use GNNs for subgraph isomorphism detection via node-labeling. We do not compare against *GNN-RE* due to their work being in the transductive setting.

III. Experiments

A. Setup

All tests are run on an Nvidia RTX 3080 16 GB Laptop GPU, an Intel i7 11800H with 32GB of system memory. We run the following experiments:

1) **Time to inference with respect to number of nodes:** The goal of this experiment is to empirically evaluate the run-time complexity of various CGI algorithms. We synthetically generate 10 pairs of graphs–5 isomorphic and 5 non-isomorphic for each graph size we tested.[5]
2) **CGI detection accuracy with respect to L:** To investigate theorem 2, we perform an ablation study on DepthGraphNet and *GREED* using the RE dataset. The goal is to investigate the relationship between L and graph diameter.
3) **CGI detection accuracy on the RE dataset with respect to algorithm:** This experiment aims to amalgamate accuracy and time-to-inference metrics on the RE dataset.

B. Datasets

The experiments are run across two datasets. One of which is generated synthetically. Another is adapted from *GNN-RE* [1]. We refer to the former as the "synthetic dataset" and the later as the "RE dataset" for brevity.

The synthetic dataset was created by randomly generating graph pairs. To synthetically generate isomorphic pairs, one graph is generated at random and then has its node orderings scrambled to produce another isomorphic graph. The non-isomorphic pairs were produced by generating two random graphs which have the same number of nodes.

Importantly, the authors of *GNN-RE* produced the RE dataset using genuine subcircuit netlists so our performance on their dataset should be indicative of real-world applicability. Each component of the netlist has 34 features. To adapt the RE dataset for our inductive setting, we break up the initial dataset–which is one large, disconnected graph containing 43 sub graphs–into 43 individual graphs. The 43 graphs are sorted by $|V|$ and then delineated every other graph into the train/test set. Afterwards, for each graph in both the training set and test set, g_i we:

1) Produce k graphs which are isomorphic to g_i by scrambling their node orderings
2) Produce k graphs which are not isomorphic to g_i by using the *Deep Robust* [6] library to randomly remove a random number of edges. We then randomly add back in the same number of edges we removed. The resulting non-isomorphic graphs have the same number of edges as g_i but are perturbed (that is, the connections between nodes are randomly removed/added).
3) The resulting total of $2k$ graphs are then paired to produce our inductive CGI detection dataset, with each pair consisting of g_i, and the scrambled/altered graph. A label of 1 or 0 is given, representing isomorphic and non-isomorphic respectively.

At the end of this process, our RE dataset has size $43 \times 2k$; for our work, we set $k = 16$. The features of each node are retained during the permutation process. To avoid polluting the dataset with incorrectly labeled/generated graphs, we prevent the generation of non-isomorphic permutations of certain

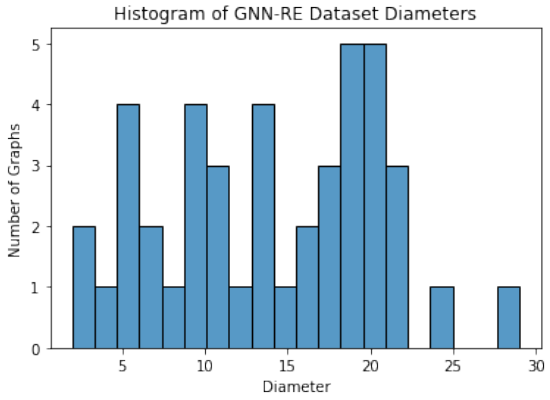

Fig. 3. A histogram of the diameters of the graphs of the GNN-RE [1] dataset. The maximum diameter is 29, average is 14, and the mode is 19 and 20.

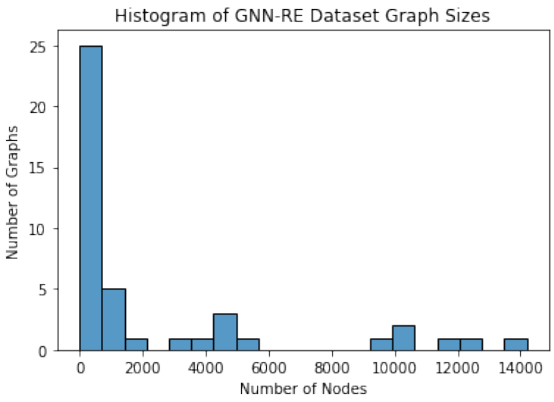

Fig. 4. The average graph size is $|V| = 2436$. The median size is $|V| = 309$.

[5]We specify graph size by setting the number of nodes when we randomly generate the graphs.

graphs and generate an additional k isomorphic permutations instead; due to the limited permutations possible, graphs with fewer than 10 nodes and fully-connected graphs are not used to generate non-isomorphic permutations. In both datasets, the resulting graphs are directed.

C. DepthGraphNet Architecture and Training

As previously mentioned, we employ a discriminative learning paradigm [7] for training. In each iteration, our data loader provides two graphs along with a label indicating whether the graphs are isomorphic or not. The discriminative learning task is designed to directly predict this binary label, encouraging the model to learn a decision boundary between isomorphic and non-isomorphic graph pairs. This is realized using the Binary Cross-Entropy (BCE) loss function:

- If the graphs are isomorphic (label 1), the BCE loss encourages the model to output a high probability for the positive class, promoting the generation of similar embeddings for these graph pairs.
- If the graphs are non-isomorphic (label 0), the BCE loss encourages the model to output a low probability for the positive class, thus differentiating the embeddings of non-isomorphic graphs from those of the isomorphic ones.

Through this learning paradigm, the network aims to directly model the conditional probability of the isomorphism label given the graph pair, learning a decision boundary in the high-dimensional embedding space. We train the model for 30 epochs using the Adam optimizer with $\texttt{lr} = \texttt{2e} - \texttt{4}$.

For F, we use the *Graph Isomorphism Network* [12] (GIN), which uses a sum aggregation function. We select $L = 29$ because 29 is the maximum diameter of the RE dataset. Hence, the name of our network is *DepthGraphNet*, denoting that the number of layers is dependent on the diameter (the "depth") of the graph. The GIN requires a neural network to help with encoding features, we use a simple 2 layer dense network with 36 neurons in each layer and a ReLU activation between them. The output of F is encoded using global sum pooling to produce an encoding for both graphs, z_1 and z_2.

Finally, the absolute difference of the encodings are are passed into S. In our work, S is composed of 3 dense layers with 32 neurons each; the first two layers use ReLU and the final layer uses a sigmoidal activation function. To summarize, the SGNN takes in two graphs, U and K and returns if they are isomorphic (1.0) or non-isomorphic (0.0).

D. Adapting GREED

For *GREED*, we trained using the same discriminative learning-paradigm and optimiser. However we set $\texttt{lr} = \texttt{8e} - \texttt{5}$ and trained for 100 epochs as using the default training hyper parameters caused poor performance. For parity, we set $\texttt{input\_dim} = 34$, $\texttt{hidden\_dim} = 36$ and $\texttt{output\_dim} = 32$. *GREED* requires two labels during training, a lower and upper bound of the graph-edit-distance; to adapt it for CGI detection, we set both labels to 1.0 or 0.0. The number of message passing layers in *GREED* is set to 29 unless stated otherwise. Note that the authors of *GREED* recommend $L = 8$ for their architecture with no further explanation. Both networks were trained with a batch size of 8. All inferencing is done on the GPU with a batch size of 1 for fairness to the classical algorithms. Additionally, we give timings of our network inferencing using a CPU in Table 5; for these timings, we use the same parameters.

IV. Results

According to Fig. 5, the run times of all classical algorithms are poor compared to the two SGNN approaches. This result is significant as it underscores the need for more scalable approaches to CGI detection. Predictably, VF2 has the slowest run-time due to being a brute-force exact algorithm. The two approximate algorithms, Hungarian and 29-WL offer speed-improvements but may still be infeasible for very large graphs. On the other hand, the SGNNs offer logarithmic time complexities–an extremely desirable property for scalable application. Another observation is that *GREED* runs at most 0.30 seconds faster than DGN. We surmise that this difference is caused by the caching of the dataset within GPU memory rather than due to architectural/run-time differences. This conclusion is bolstered by the fact that the inference-time differences do not increase as the graph size increases.

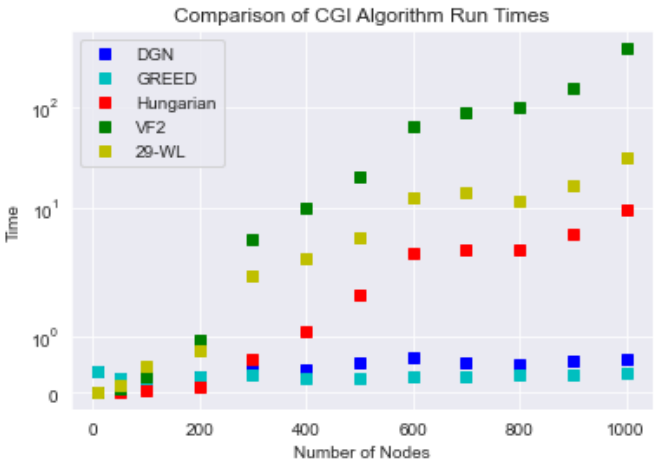

Fig. 5. The time to inference of various CGI algorithms with respect to graph size. Timings are computed over 10 graph pairs per number of nodes.

The relationship between L and graph diameter is apparent in Fig. 6. For both architectures, accuracy plateaus once L exceeds the mean of the graph diameters. This provides empirical evidence to theorem 2. Moreover, this observation serves as a guide for SGNN architects–L should not be greater than the maximum graph diameter and there are diminishing benefits in increasing L beyond the mode of the graphs' diameters.

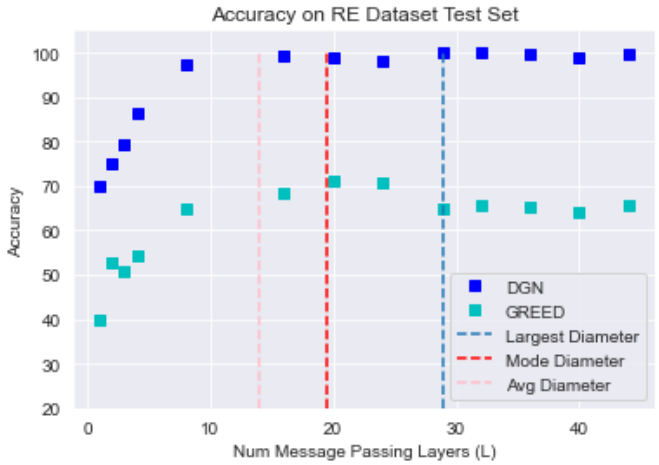

Fig. 6. The average accuracy of DepthGraphNet and *GREED* over 5 trials. DepthGraphNet achieves its best average test accuracy of 100% at $L = 29$ and *GREED* achieves best average test accuracy at $L = 20$.

TABLE I
ACCURACY AND INFERENCE TIME ON RE DATASET'S TEST SET

Method	Accuracy (%)	Time (s)
DGN (ours)	**100.0**%	$19s_{GPU}/43s_{CPU}$
GREED$^{*}{}_{L=24}$	82.14%	$\mathbf{18s}_{GPU}/62s_{CPU}$
Hungarian	54.76%	233s
29-WL	**100.0**%	262s
VF2**	100.0%**	at least 1200s**

* To show *GREED* in the best light, we show results for $L = 24$ as they attained their peak accuracy in this configuration.
** VF2 took a prohibitively long amount of time to inference on the entirety of the RE dataset. We report results for the graphs it inferenced on after 20 minutes.

Remarkably, DGN achieves perfect accuracy on the test set while only requiring 80,513 parameters. This is especially impressive considering the $L = 29$ equivalent *GREED* model uses 120,729 parameters but fails to perform as well. In fact, even the $L = 24$ *GREED* model has more parameters (100,929) than DGN but fails to achieve parity in accuracy. The timing results from table I reinforce our earlier conclusions from Fig. 5. Both SGNNs inference over **12x faster** compared to the fastest classical counterparts. Furthermore, to prove our speedup is not solely due to GPU parallelization, we show our inference times are superior even when using CPU.

A. Conclusion

Our results bolster the conclusions of the theorems provided–we show empirically that increasing the number of message passing layers beyond the diameter of a graph gives diminishing returns. Conveniently, because computing the diameter of a graph runs in polynomial time, SGNN designers are now provided with an effective feature to guide their designs. Another desirable property of our approach is the *fast* training times; using the aforementioned training scheme described in III-C, DepthGraphNet finished training in **less than 5 minutes**.

Future extensions of our work could include the exploration of how the quantity of neurons utilized in our dense layers influences the number of node features. A logical subsequent topic for exploration would be sub-graph isomorphism. With excellent performance and scalability, our study evidences the significant potential of our DGN as a practical solution for real-world applications.

REFERENCES

[1] Lilas Alrahis, Abhrajit Sengupta, Johann Knechtel, Satwik Patnaik, Hani Saleh, Baker Mohammad, Mahmoud Al-Qutayri, and Ozgur Sinanoglu. Gnn-re: Graph neural networks for reverse engineering of gate-level netlists. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 41(8):2435–2448, 2021.

[2] Jane Bromley, Isabelle Guyon, Yann LeCun, Eduard Säckinger, and Roopak Shah. Signature verification using a" siamese" time delay neural network. *Advances in neural information processing systems*, 6, 1993.

[3] L.P. Cordella, P. Foggia, C. Sansone, and M. Vento. A (sub)graph isomorphism algorithm for matching large graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(10):1367–1372, 2004.

[4] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring network structure, dynamics, and function using networkx. In Gaël Varoquaux, Travis Vaught, and Jarrod Millman, editors, *Proceedings of the 7th Python in Science Conference*, pages 11 – 15, Pasadena, CA USA, 2008.

[5] Harold W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics (NRL)*, 52, 1955.

[6] Yaxin Li, Wei Jin, Han Xu, and Jiliang Tang. Deeprobust: A pytorch library for adversarial attacks and defenses. *arXiv preprint arXiv:2005.06149*, 2020.

[7] Tao Lu, Qiang Zhou, Wenhua Fang, and Yanduo Zhang. Discriminative metric learning for face verification using enhanced siamese neural network. *Multimedia Tools and Applications*, 80:8563–8580, 2021.

[8] Rishabh Ranjan, Siddharth Grover, Sourav Medya, Venkatesan Chakaravarthy, Yogish Sabharwal, and Sayan Ranu. Greed: A neural framework for learning graph distance functions. In *Advances in Neural Information Processing Systems*, 2022.

[9] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009.

[10] Boris Weisfeiler and Andrei Leman. The reduction of a graph to canonical form and the algebra which appears therein. *nti, Series*, 2(9):12–16, 1968.

[11] Haoran Wu, Zhiyong Xu, Jianlin Zhang, Wei Yan, and Xiao Ma. Face recognition based on convolution siamese networks. In *2017 10th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI)*, pages 1–5. IEEE, 2017.

[12] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018.

[13] Yichi Zhang, Meng Yu, Na Li, Chengzhu Yu, Jia Cui, and Dong Yu. Seq2seq attentional siamese neural networks for text-dependent speaker verification. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6131–6135. IEEE, 2019.