Beyond GNNs: An Efficient Architecture for Graph Problems

Pranjal Awasthi,[1] **Abhimanyu Das,** [1] **Sreenivas Gollapudi** [1]

[1] Google Research
pranjalawasthi@google.com, abhidas@google.com, sgollapu@google.com

Abstract

Despite their popularity for graph structured data, existing *Graph Neural Networks* (GNNs) have inherent limitations for fundamental graph problems such as shortest paths, k-connectivity, minimum spanning tree and minimum cuts. In these instances, it is known that one needs GNNs of high depth, scaling at a polynomial rate with the number of nodes n, to provably encode the solution space, in turn affecting their statistical efficiency. In this work we propose a new hybrid architecture to overcome this limitation. Our proposed architecture that we call as GNN$^+$ networks involve a combination of multiple parallel low depth GNNs along with simple pooling layers involving low depth fully connected networks. We provably demonstrate that for many graph problems, the solution space can be encoded by GNN$^+$ networks using depth that scales only *poly-logarithmically* in the number of nodes. This also has statistical advantages that we demonstrate via generalization bounds for GNN$^+$ networks. We empirically show the effectiveness of our proposed architecture for a variety of graph problems and real world classification problems.

Introduction

In recent years Graph Neural Networks (GNNs) have become a popular choice for learning problems over graph structured data (Hamilton, Ying, and Leskovec 2017; Kipf and Welling 2016; Veličković et al. 2017). Computation in GNNs is performed by each node sending and receiving messages along the edges of the graph, and aggregating messages from its neighbors to update its embedding vector. After a few rounds of message passing, the computed node embeddings from all the nodes are aggregated to compute the final output (Gilmer et al. 2017). This leads to a simple and elegant architecture for learning functions on graphs. On the other hand, from a theoretical and practical perspective, we also need these architectures to be *sample efficient*, i.e., learnable from a small number of training examples, where each training example corresponds to a graph. Recent works have shown that generalization in GNNs depends upon the depth of the architecture, i.e., the number of rounds of message passing, as well as the embedding size for each node in the graph (Garg, Jegelka, and Jaakkola 2020). However, this requirement is in fundamental conflict with the message passing framework. In particular, using GNNs to compute several fundamental graph problems such as *shortest paths, minimum spanning tree, min cut* etc., necessarily requires the product of the depth of the GNN and the embedding size to scale as $\sqrt{n}$ where n is the size of the graph (Loukas 2020b). This in turn places a significant statistical burden when learning these fundamental problems on large scale graphs. A natural question then is whether graph neural networks have any advantage over combinatorial algorithms for the above "simple" graph problems that lie in the complexity class P, i.e., they admit polynomial time algorithms. As has been observed in practice (Veličković et al. 2017), and we will demonstrate via experiments in Section , graph neural network based approaches can often significantly outperform combinatorial algorithms. In light of the above we ask the the following question: *Can one develop sample efficient neural network architectures for graph problems while retaining the simplicity of the message passing framework?*

Recent works have addressed the above question empirically by proposing extensions to the basic GNN framework by augmenting various pooling operations in conjunction with message passing rounds to capture more global structure (Ying et al. 2018; Simonovsky and Komodakis 2017; Fey et al. 2018). In this work we propose a theoretically principled architecture, called GNN$^+$ networks for learning graph problems. Specifically, we borrow from two fundamental paradigms in graph algorithm design namely, sketching and parallel computation, to design the proposed networks and show that these paradigms admit a neural architecture that simultaneously achieves low depth and low embedding size for many graph problems. Before we present our improved architecture, we briefly describe the standard GNN framework.

Model for GNNs. In this work we will study GNNs that fall within the message passing framework, and using notation from previous works we denote such networks as GNN$^{\mathrm{mp}}$ (Loukas 2020b). A GNN$^{\mathrm{mp}}$ network operates in the AGGREGATE and COMBINE model (Gilmer et al. 2017) that captures many popular variants such as GraphSAGE, Graph Convolutional Networks (GCNs) and GIN networks (Hamilton, Ying, and Leskovec 2017; Kipf and Welling 2016; Xu et al. 2019a). Given a graph $G = (V, E)$, let $x_i^{(k)} \in \mathbb{R}^d$ denote the feature representation of node i

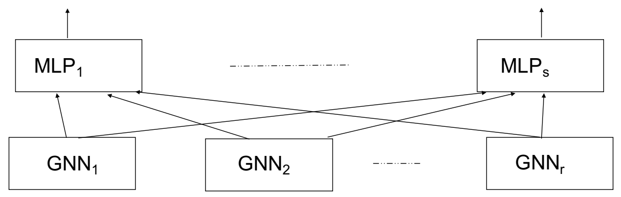


Figure 1: The basic GNN$^+$ block.

at layer k. We will refer to d as the embedding size of the network. Then we have

$$a_i^{(k-1)} = \text{AGGREGATE}(\{x_j^{(k-1)} : j \in N(i)\}) \quad (1)$$

$$x_i^{(k)} = \text{COMBINE}(x_i^{(k-1)}, a_i^{(k-1)}). \quad (2)$$

Here $N(i)$ is the set of neighbors for node i. Typically the aggregation and combination is performed via simple one or two layer full connected networks, also known as multi layer perceptrons (MLPs).

GNN$^+$ Networks. Our proposed GNN$^+$ networks consist of one or more layers of a GNN$^+$ block shown in Figure 1. The GNN$^+$ block comprises of r parallel GNN$^{\text{mp}}$ networks follows by s parallel fully connected network modules for pooling where r and s are the hyperparameters of the architecture. More importantly we restrict the r GNN$^{\text{mp}}$ modules to share the same set of weights. Hence the parallel GNN$^{\text{mp}}$ modules only differ in the way the node embeddings are initialized. Furthermore, we restrict each GNN$^{\text{mp}}$ to be of low depth. In particular, for graphs over n nodes of maximum degree d and diameter D, we will restrict the GNN$^{\text{mp}}$ to be of depth $O(D \cdot \text{polylog}(n))$ and use an embedding size of $O(d \cdot \text{polylog}(n))$[1]. Similarly, we require the s fully connected networks to be of depth $O(D \cdot \text{polylog}(n))$ and share the network weights. Hence, they only differ in the inputs that they receive from the GNN$^{\text{mp}}$ modules in lower layers. We connect the outputs of the GNN$^{\text{mp}}$ modules to the fully connected pooling networks in a sparse manner and restrict the input size of each fully connected network to be $O((d + D) \cdot \text{polylog}(n))$. Stacking up L layers of GNN$^+$ blocks results in a GNN$^+$ network that is highly parameter efficient and in total has $O((d + D)L \cdot \text{polylog}(n))$ parameters. For such a network we call the depth as the total number of message passing rounds and the number of MLP layers used across all the L stacks. Since we restrict our MLPs and GNN$^{\text{mp}}$ blocks inside a GNN$^+$ network to be of low depth, we will often abuse notation and refer to a GNN$^+$ architecture with L stacks of GNN$^+$ blocks as a depth L architecture. Our proposed design lets us alternate between local computations involving multiple parallel GNN blocks and global post-processing stages, while still being sample efficient due to the enforced parameter sharing. We will show that optimal or near-optimal solutions to many popular graph problems can indeed be computed via a GNN$^+$ architecture. Below we briefly summarize our main results.

[1]When measuring the embedding size we will assume that each dimension can hold up to $O(\log n)$ bits.

Overview of Results. To demonstrate the generality of our proposed architecture we study several fundamental graph problems and show how to construct efficient GNN$^+$ networks to compute optimal or near optimal solutions to these problems. In particular, we will focus on degree-d graphs, i.e., graphs of maximum node degree d, with n nodes and diameter D and will construct GNN$^+$ networks of depth $O\big(D \cdot \text{polylog}(n)\big)$ and $O\big((D + d)\text{polylog}(n)\big)$ total parameters.

Shortest Paths. The first problem we consider is the fundamental graph problem of computing (approximate) all pairs shortest paths in undirected graphs. Given a graph $G = (V, E)$, let $d_G(u, v)$ be the shortest path between nodes u and v. We say that an output $\{\tilde{d}_G(u, v) : u, v \in V\}$ is an α-approximate solution if for all $u \neq v$ it holds that $d_G(u, v) \leq \tilde{d}_G(u, v) \leq \alpha d_G(u, v)$. We construct efficient GNN$^+$ networks for all pairs shortest paths with the following guarantee.

Theorem 1. *For any integer $c > 3$, there is a depth $O(D \log d + \log n)$ GNN$^+$ network with $O\big((n^{\frac{4}{c+1}} + d) \cdot$ polylog$(n)\big)$ parameters that computes c-approximate all pairs shortest paths in undirected unweighted graphs of diameter D with n nodes and maximum degree d. On the other hand, using GNN^{mp} networks to compute a c-approximate all pairs shortest paths, even on constant degree graphs with $O(\log n)$ diameter, requires either the depth or the embedding size to be $\Omega(\frac{n}{c \log n})$.*

We will show how the above result implies that GNN$^+$ has a provable sample-efficiency advantage over GNN$^{\text{mp}}$ once the approximation factor c exceeds three. As we will see later (in Theorem 5), generalization bounds for GNN$^{\text{mp}}$ and GNN$^+$ scale with the product of the depth and the number of parameters. Furthermore, in all popularly used GNN$^{\text{mp}}$ networks the number of parameters scale with the embedding size. The above result shows that the depth $\times$ embedding size required for GNN$^{\text{mp}}$ is large(linear in n). On the other hand, for real world graphs that follow power law distributions (Cohen and Havlin 2003), we will have D scaling as $O(\log n)$ and d scaling as n^ϵ for some $\epsilon \in (0, 1)$. Hence for GNN$^+$, the product of the depth and the number of parameters will typically scale sub-linearly in n, and we achieve a polynomial advantage over standard GNN$^{\text{mp}}$ networks. Furthermore, if the maximum degree grows only polylogarithmic in n, say as in random graphs, then we achieve an exponential improvement over GNN$^{\text{mp}}$ networks.

Note that for $c < 3$ lower bounds exist that rule out any sublinear sized embedding (hence networks with small embedding size) to approximate shortest paths (Matoušek 1996). The regime of $c > 3$ is of practical interest as well. In practice, computing exact shortest paths becomes impractical for large scale graphs and often combinatorial algorithms for computing c-approximate shortest paths are used. The larger the c, the faster these algorithms can answer shortest path queries. Empirical results show that choosing c to be a large constant works well in many settings (Das Sarma et al. 2010). **Distance Oracles**, which are efficient data structures that preprocess the graph and store short sketches for each node, are used in practice to answer approximate shortest path queries in real time. The larger the sketch size the better the quality of approximation (Thorup and Zwick 2005; Das Sarma et al. 2010). An important consequence of the theorem above is that the the node embeddings produced by GNN$^+$ network in Figure 2 can also serve as sketches, where the sketch size will correspond to the embedding size used in the network. In other words the output of the GNN$^+$ network can be viewed as a distance oracle too. It is natural to then ask whether in practice GNN$^+$ based sketches can outperform combinatorial algorithms. In Section we present empirical results showing that a GNN$^+$ network trained with sketch size of $\lceil \log n \rceil$, significantly outperforms the state-of-the-art combinatorial algorithm of Das Sarma et al. (2010), even when the combinatorial algorithm is allowed to use a sketch size of $8\lceil \log n \rceil$.

Connectivity Measures. Next we consider computing various graph connectivity measures. We first study the popular measure based on graph effective resistances (Chandra et al. 1996).

Definition 1 (Effective Resistance)**.** *Let G be a weighted undirected graph G with adjacency matrix A and the associated Laplacian $L = D - A$. Given an edge u, v, the effective resistance between u, v is defined as*

$$R_{u,v} = \xi_{u,v}^\top L^\dagger \xi_{u,v}.$$

Here $\xi_{u,v}$ is an n dimensional vector with $+1$ at position u, -1 at position v and zeros everywhere. $L^\dagger$ refers to the matrix pseudo-inverse.

We also study the following connectivity measure that was proposed by Panigrahy, Najork, and Xie (2012) in the context of web graphs. Given an undirected graph G, let G_p be the random graph obtained by sampling each edge independently with probability p.

Definition 2 (Affinity)**.** *For any two vertices u, v, and for $p \in [0, 1]$, define $A_p(u, v)$ to be the probability that u, v are connected in G_p. Then the affinity between u and v is defined as*

$$A(u, v) = \mathbb{E}_p[A_p(u, v)]$$

where the expectation is taken over p drawn from the uniform distribution in $[0, 1]$.

For the above two measures we show the following

Theorem 2. *There exists a GNN$^+$ architecture of depth $O(D \log(nd))$ and $O(D \log(nd))$ parameters, that on graphs of diameter D with n nodes and maximum degree d, approximates the above connectivity measures up to constant factors. On the other hand, using GNN^{mp} networks to compute the above measures, even approximately on constant degree $O(\log n)$ diameter graphs, necessarily requires either the depth or the embedding size $\tilde{\Omega}(\sqrt{n})$.*

Clustering, Minimum Cuts and Minimum Spanning Trees. Finally, we showcase the power of a GNN$^+$ architecture for computing other fundamental graph problems. Given an undirected graph G, the spectral clustering of G corresponds to the cut obtained by taking the sign of the eigenvector v corresponding to the second smallest eigenvalue $\lambda_2(L)$, where L is the graph Laplacian. For computing the spectral clustering via GNN$^+$ networks we show the following

Theorem 3. *There is a GNN$^+$ network of depth $O(\frac{1}{\lambda_2(L)\epsilon^2} \log n)$ and $O(d)$ parameters, that computes an ϵ-approximate spectral clustering on graphs of diameter D with n nodes and maximum degree d. On the other hand, using GNN^{mp} networks to even approximately compute the spectral clustering on constant degree $O(\log n)$ diameter graphs requires either the depth or the embedding size to be $\tilde{\Omega}(\sqrt{n})$.*

The theorem above has an inverse dependence on the second smallest eigenvalue of the graph Laplacian, i.e, $\lambda_2(L)$. In worst case scenarios $\lambda_2(L)$ could be very small thereby leading to no advantage of GNN$^+$ over GNN^{mp} networks. However, once $\lambda_2(L) \geq \frac{1}{\sqrt{n}}$, GNN$^+$ outperforms vanill GNN^{mp} networks. Next we consider the problems of computing a global minimum cut and minimum spanning trees in undirected graphs.

Theorem 4. *There exist GNN$^+$ networks of of depth $O((D + \log n) \log n)$ and $O(d)$ parameters for computing a global minimum cut (MINCUT) and minimum spanning tree (MST) on graphs of diameter D with n nodes and maximum degree d. Furthermore, using GNN^{mp} networks to compute these primitives (even approximately) on a constant degree $O(\log n)$ diameter graphs necessarily requires either the depth or the width to be $\tilde{\Omega}(\sqrt{n})$.*

As in the case with Theorem 1, the results in Theorems 2, 3 and 4 all imply that GNN$^+$ has a provable sample-efficiency advantage over GNN^{mp} in most real world settings where D scales as $O(\log n)$ and d scales sub-linearly in n.

Generalization Bounds. Our final result concerns the generalization properties of a depth L GNN$^+$ architecture. For ease of exposition, we state here the results for the case when the GNN$^+$ architecture produces a one dimensional output. More general results are presented in Appendix . Our generalization bounds depend on the depth L and the total number of parameters P in the GNN$^+$ network. Following recent work on providing generalization bounds for fully connected and convolutional neural networks (Bartlett, Foster, and Telgarsky 2017; Long and Sedghi 2020) that are based on *distance to initialization*, we consider the class $\mathcal{F}_\beta$ of depth L GNN$^+$ networks with P parameters that are at a distance β from a reference parameter configuration (typically the parameters at random initialization). Let $y \in \mathbb{R}$ denote the output of the network and consider a Lipschitz loss function $\ell(y, \hat{y})$. Then, we provide following guarantee.

Theorem 5 (Informal Theorem)**.** *Let $\ell(\hat{y}, y)$ be a Lipschitz loss function bounded in $[0, B]$. Then, given m i.i.d. samples $(G_1, y_1), (G_2, y_2), \ldots (G_m, y_m)$ generated from a distribution D, with probability at least $2/3$, it holds that for all $f \in \mathcal{F}_\beta$,*

$$\left| \hat{\mathbb{E}}_D[\ell_f] - \mathbb{E}_D[\ell_f] \right| \leq O\left(B \sqrt{\frac{P(\beta + L)}{m}} \right).$$

Here $\hat{\mathbb{E}}_D[\ell_f]$ refers to the empirical loss on the training set, and $\mathbb{E}_D[\ell_f$ refers to the expected loss over the data distribution. We refer the reader to Theorem 16 in Appendix for a formal statement and the proof. Notice that the above theorem implies that our proposed GNN$^+$ architecture for the above graph problems can indeed be trained using very few samples as opposed to the traditional GNN$^{\text{mp}}$ networks since the GNN$^+$ network requires much fewer parameters and depth. Furthermore, since a GNN$^{\text{mp}}$ network is a special case of a GNN$^+$ architecture, our analysis also leads to an improved bound on the generalization guarantees for GNN$^{\text{mp}}$ networks as well. In particular, the above theorem improves upon the recent work of Garg, Jegelka, and Jaakkola (2020) that provides generalization guarantees for training GNNs that scale with the branching factor (degree d) of the graph. Using our improved analysis we are able to remove this dependence on the branching factor. See Appendix for details.

Related Work

GNNs operate primarily in the message passing framework where nodes aggregate and combine messages from their neighbors. Several variants of this basic paradigm have been proposed, each differing in how the aggregation and combination is performed. Popular variants include GraphSAGE (Hamilton, Ying, and Leskovec 2017), Graph Convolutions Networks (Kipf and Welling 2016), GIN networks (Xu et al. 2019a), and graph pooling (Ying et al. 2018).

Recent works have also studied the representation power of GNNs. The work of Xu et al. (2019a) demonstrates that GNNs as considered in Eq. 1 are as powerful as the Weisfeiler-Lehman test for graph isomorphism (Weisfeiler and Lehman 1968). Recent works Loukas (2020a); Chen et al. (2020) study what properties of GNNs affect their ability to distinguish among certain classes of graphs. The work of Xu et al. (2019b) studies the power of the message passing framework of GNNs in representing computations involving dynamic programming. GNN networks that can capture higher order variants of the WL test have also been proposed recently (Maron et al. 2019). Several works have also explored the limitations of GNNs for computing graph primitives. The work of Loukas (2020b) established a correspondence between the message passing GNN framework and the well studied CONGEST model of distributed computing (Peleg 2000). Based on the above correspondence it follows that in order to represent several important graph problems such as shortest paths, minimum cuts and minimum spanning tree, either the depth of the GNN or the embedding size of the nodes has to scale with the graph size at a polynomial rate. Notice that these lower bounds apply to any form of message passing framework and as a result recent works in incorporating non-symmetric node messages (Sato, Yamada, and Kashima 2019) in GNNs also run into the same barriers.

In order to address the above limitations recent works have proposed combining the GNN architecture with pooling mechanisms for aggregating more global information (Ying et al. 2018; Defferrard, Bresson, and Vandergheynst 2016; Simonovsky and Komodakis 2017; Fey et al. 2018; Bianchi, Grattarola, and Alippi 2019; Du et al. 2019; Mesquita, Souza, and Kaski 2020). For example the work of Ying et al. (2018) proposes a hierarchical approach where a GNN network is followed by a clustering step to compute higher level "nodes" to be used in the subsequent GNN operation. While these approaches show empirical promise, ours is the first work to design a principled architecture with theoretical guarantees that merges local distributed computations with global post-processing steps. The recent work of You, Ying, and Leskovec (2020) performs an empirical study of how different parameters such as the depth and the embedding size affect the performance of GNNs and how to identify a good tradeoff for a specific task at hand.

Finally, the question of generalization for GNNs has also been studied in recent works (Garg, Jegelka, and Jaakkola 2020; Zhang et al. 2020; Du et al. 2019). The most relevant to us is the recent work of Garg, Jegelka, and Jaakkola (2020) that analyzes the Rademacher complexity of GNNs with the aggregate mechanism being a simple addition and the combine mechanism being a one layer neural network. Via analyzing the Rademacher complexity the authors show that the generalization for GNNs depends on the depth, the embedding size and the branching factor (max degree) of the graph. Our improved analysis in Section extends the result of Garg, Jegelka, and Jaakkola (2020). Not only does our generalization bound apply to the more general GNN$^+$ networks, for the case of GNNs considered in (Garg, Jegelka, and Jaakkola 2020) our analysis shows that the dependence on the branching factor can be eliminated in the generalization bounds. Generalization bounds have also been proved recently for GNN based networks that use the Neural Tangent Kernel (NTK) during the aggregation and combination operations (Du et al. 2019). There is also a recent line of work that applies GNNs for tasks such as program analysis and software checking (Allamanis, Brockschmidt, and Khademi 2017; Dinella et al. 2020; Bieber et al. 2020).

Shortest Paths

In this section we provide a proof sketch of Theorem 1 showing how to construct an efficient GNN$^+$ architecture for the Shortest Paths problem. In particular we study all pairs shortest paths.

All Pairs Shortest Paths. The input is a graph $G = (V, E)$ with n nodes. The desired output is an $\binom{n}{2}$ dimensional vector containing (approximate) shortest path values between each pair of vertices. Given a graph G, let $d_G(u, v)$ be the shortest path between nodes u and v. We say that an output $\{\tilde{d}_G(u, v) : u, v \in V\}$ is an α-approximate solution if for all $u \neq v$ it holds that

$$d_G(u, v) \leq \tilde{d}_G(u, v) \leq \alpha d_G(u, v).$$

We first show that the GNN$^{\text{mp}}$ networks are highly inefficient for learning this problem.

Theorem 6. *Consider a GNN^{mp} network $\mathcal{N}$ of depth L over n nodes where each node has a representation size of B. If $\mathcal{N}$ encodes $\alpha(n)$-approximate all pairs shortest paths for graphs of constant degree and diameter bounded by $O(\log n)$ then it must hold that $B \cdot L \geq \Omega\big(\frac{n}{\alpha(n)\log n}\big)$. The lower bound holds for undirected unweighted graphs as well.*

Proof. The recent work of Loukas (2020b) established that computation in GNN$^{\text{mp}}$ networks is equivalent to the CONGEST model of computation in the design of distributed algorithms (Peleg 2000). In particular, a lower bound on the product of depth (L) and representation size (B) can be obtained by establishing the corresponding lower bound on the product of the number of rounds and the size of messages in the CONGEST model of computing. Furthermore, the result of Nanongkai (2014) shows that for any $\alpha(n)$-approximation, even on undirected graphs, the product of the number of the number of rounds and the message size must be $\Omega\left(\frac{n}{\alpha(n)\log n}\right)$. Hence the corresponding lower bound on $B \cdot L$ follows. □

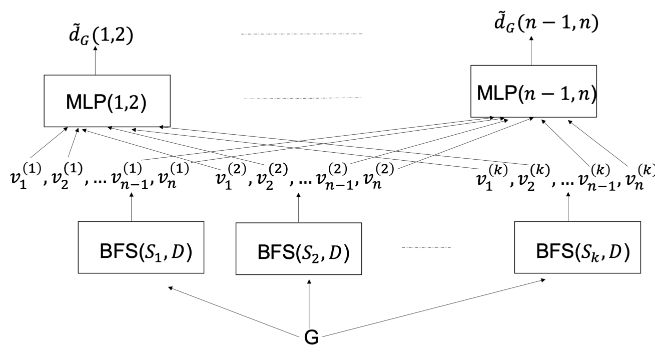
Circumventing Lower Bounds via GNN$^+$. Next we detail our proposed GNN$^+$ architecture that can encode approximate shortest paths with significantly smaller depth and embedding size.

Unweighted Graphs. To illustrate the main ideas we study the case of undirected unweighted graphs. See Appendix for the more general case of weighted graphs. The starting point of our construction is the following fundamental theorem of Bourgain (1985) regarding metric embeddings.

Theorem 7 ((Bourgain 1985))**.** *Any n-point metric (X, d) can be embedded into the Euclidean metric of dimensionality $O(\log n)$ and distortion $O(\log n)$.*

The above theorem suggests that in principle, if we only want to estimate shortest paths up to an approximation of $O(\log n)$, then we only need node embeddings of size $O(\log n)$. If there were a GNN$^{\text{mp}}$ network that could produce such embeddings, then one could simply compute the Euclidean distance between each pair of points to get the approximate shortest path. Furthermore, computing the Euclidean distance given the node embeddings can be done easily via a low depth fully connected network. Unfortunately, producing the necessary low dimensional embeddings is exactly the task for which GNN$^{\text{mp}}$ networks require large depth as proved in Theorem 6 above. While there do exist semi-definite programming based algorithms (Linial, London, and Rabinovich 1995) for computing the embeddings required for Bourgain's theorem, they are not suitable for implementation via efficient neural architectures. Instead we rely on sketching based algorithms for computing shortest path distances.

In particular, for the unweighted case we adapt the approximate shortest path procedure of Matoušek (1996) for designing an efficient network architecture. The sketch proposed in the work of Matoušek (1996) computes, for each node u, the distance of u from a random subset S of the nodes. This can be done via a simple breadth first search (BFS). Repeating this process k-times provides a k-dimensional embedding for each vertex and for an appropriate choice of k, these embeddings can be used to compute approximate shortest paths. Notice that parts of this sketch based procedure are highly amenable to be implemented in a message passing framework. Overall, the algorithm performs multiple parallel BFS subroutines to compute the embeddings. It is well known that BFS on diameter D graphs can be implemented by a GNN$^{\text{mp}}$ of depth $O(D)$. While there are other efficient sketch constructions for approximate shortest paths (Thorup and Zwick 2005; Das Sarma et al. 2010), we choose the sketch mentioned above due to its simplicity.



$\tilde{d}_G(1,2)$
$\tilde{d}_G(n-1,n)$
MLP(1,2)
MLP($n-1,n$)
$v_1^{(1)}, v_2^{(1)}, \dots v_{n-1}^{(1)}, v_n^{(1)}$
$v_1^{(2)}, v_2^{(2)}, \dots v_{n-1}^{(2)}, v_n^{(2)}$
$v_1^{(k)}, v_2^{(k)}, \dots v_{n-1}^{(k)}, v_n^{(k)}$
BFS(S_1, D)
BFS(S_2, D)
BFS(S_k, D)
G

Figure 2: The network architecture for approximate all pairs shortest paths in unweighted graphs.

Based on the above procedure, our proposed architecture is shown in Figure 2. It consists of k parallel breadth first search (BFS) modules for $k = \Theta(n^{\frac{2}{c+1}} \log n)$ for an integer $c > 3$. Module i computes the shortest path from each vertex in G to any vertex in the set S_i. The sets $S_1, S_2, \dots, S_k$ are randomly chosen subsets of the vertex set V of various sizes. In particular there are $\Theta(n^{\frac{2}{c+1}})$ subsets of size 1, $\Theta(n^{\frac{2}{c+1}})$ subsets of size 2, $\Theta(n^{\frac{2}{c+1}})$ subsets of size 2^2, and so on up to $\Theta(n^{\frac{1}{c}})$ subsets of size $2^{\lfloor \log n \rfloor}$. The BFS module i produces n distance values $v_1^{(i)}, \dots, v_n^{(i)}$. These modules are followed by $\binom{n}{2}$ fully connected networks where each module is responsible for computing the approximate shortest path distance between a pair of vertices. In particular we have $\tilde{d}_G(s,t) = \max_i |v_s^{(i)} - v_t^{(i)}|$.

Notice from the discussion in Section that the architecture in Figure 2 is a GNN$^+$ network with a single GNN$^+$ block. In the next section we will show how we can generalize to a suite of graph problems by stacking up multiple GNN$^+$ blocks. For our proposed network in Figure 2 we have the following guarantee.

Theorem 8. *For any integer $c > 3$, there exists a neural network as shown in Figure 2 of depth $O(D \log d + \log n)$ and $\tilde{O}(n^{\frac{4}{c+1}})$ parameters, that on graphs of diameter D and maximum degree d, encodes c-approximate all pairs shortest paths in G.*

The proof of the theorem above relies on the following two lemmas concerning the implementation of the BFS and the MLP modules. See Appendix for the proof.

Lemma 1. *The BFS module in Figure 2 can be implemented by a GNN of depth $O(D \log d)$, $O(1)$ total parameters and with each node having a representation size of $O(1)$.*

Lemma 2. *For any k, the MLP module in Figure 2 can be implemented by a network of depth $O(\log k)$, $O(k^2)$ total parameters.*

Minimum Cuts

To illustrate another application, in this section we design an efficient GNN$^+$ architecture for computing the minimum cut in an undirected graph. We first show in Appendix that even computing an approximate mincut using traditional GNN$^{\text{mp}}$ networks requires either the depth or the embedding size to be $\tilde{\Omega}(\sqrt{n})$. Our efficient GNN$^+$ based architecture is based on the parallel algorithm for computing mincut (Karger and Stein 1996) and is shown in Figure 9 in the Appendix. More importantly the architecture comprises of multiple layers of GNN$^+$ blocks in contrast to a single GNN$^+$ block in the case of shortest paths.

The algorithm of Karger and Stein (1996) relies on the following lemma.

Lemma 3 ((Karger and Stein 1996)). *Let $G = (V, E)$ be an undirected unweighted graph with m edges and n vertices. Then with probability at least $\frac{1}{n^2}$, a random ordering L of the edges contains a prefix L' of L such that the graph $G' = (V, L')$ contains exactly two connected components corresponding to the global minimum cut in the graph.*

Using the above, Karger and Stein (1996) proposed a Monte-Carlo randomized algorithm to compute the global minimum cut. For a given ordering L, the algorithm estimates the length of the prefix L' corresponding to the cut by using a binary search procedure. This provides the set of *active edges*, i.e., edges in the current choice L' of the prefix. Then one can run a connected components algorithm using edges in L'. If the prefix is too small, it results in more than two connected components; if it is too large it produces one connected component. If the number of connected components is two then the algorithm stops. Otherwise it recurses on the appropriate side of L'.

We implement the above algorithm using a GNN$^+$ architecture of depth $O(\log m)$ as shown in Figure **??** where each pair of (GNN,Update Prefix) blocks is a GNN$^+$ block and corresponds to a particular stage of the above binary search procedure. During each stage one needs to perform a connected component subroutine. This can be done via BFS and is implemented in the GNN module. The GNN is followed by the UpdatePrefix module that is an MLP implementing the logic of selecting the appropriate side of the permutation to recurse on.

More formally, at each of the $O(\log m) = O(\log n)$ stages, each vertex in the GNN$^{\text{mp}}$ maintains a list of which of its connecting edges are *active*, their position in the initial ordering L, and an integer value in $[1, m]$ corresponding to the current choice of the prefix L'. This in total requires $O(d)$ embedding size. The goal next is to infer whether the number of connected components induced by the active edges is one, two, or more than two. This in turn decides the part of the ordering the next stage will focus on. The computation of connected components can be carried out using at most two breadth first searches and hence via $O(D)$ rounds of a GNN$^{\text{mp}}$ network, and a low depth MLP. Following this intuition we arrive at the proposed architecture in Figure 9. Formally, we have the following guarantee.

Theorem 9. *There is a GNN$^+$ architecture (Figure **??**) of depth $O(D \log^2 n)$ and $O(d + \log n)$ parameters that, on graphs with maximum degree d and diameter D, produces the minimum cut.*

Experiments

We show the efficacy of GNN$^+$ on the aforementioned graph problems: Shortest Paths, Effective Resistance, Affinity, MINCUT , and MST, and compare to a state-of-the-art GNN$^{\text{mp}}$ model (Xu et al. 2019a). Additionally, we also present results on evaluating our proposed architecture on real world graph classification problems. **Dataset.** We generated synthetic random graphs between 500 and 1000 nodes (n). For the affinity measure, we used dense graphs with 250 nodes to have a reasonable number of alternate paths between any two end points. In particular, we generate graph s from the Erdos-Renyi model $G(n, p)$ with edge sampling probability $p = \frac{\alpha}{n}$. Specifically, we set α to be a constant in $[1, 100]$ to capture varying degrees of sparsity. For each n, p we generate $30,000$ training examples consisting of tuples of the form $(G, s, t, d(s, t))$ where G is a random graph drawn from $G(n, p)$, s, t are two vertices uniformly drawn at random and $d(s, t)$ is one of shortest path value, effective resistance, or affinity between the two vertices. In the case of min cut and minimum spanning tree, we generate tuples (g, v_G) where v_G corresponds to the value of the minimum spanning tree or the global minimum cut.

Models and Configurations. For our baseline GNN$^{\text{mp}}$ implementation, we used the GIN model proposed in Xu et al. (2019a). This has been empirically shown (Xu et al. 2019a; Loukas 2020b; Errica et al. 2020) to be a state-of-the-art GNN$^{\text{mp}}$ model on several datasets. GIN updates feature representations $x_v^{(k)}$ of each node v at iteration k as: $x_v^{(k)} = \text{MLP}\Big((1+\epsilon^{(k)}) \cdot x_v^{(k-1)} + \sum_{u \in N(v)} x_u^{(k-1)}\Big)$, where MLP refers to a Multi-Layer Perceptron, $N(v)$ is the set of neighbors of v, and ϵ is a learnable parameter. For problems that involved weighted graphs (e.g. MST), we incorporated edge weights into the GIN update equation by using the weighted sum of neighbor representations.

Our GNN$^+$ implementation used the same GIN implementation as its internal GNN$^{\text{mp}}$ block. All graphs in our experiments were undirected. For both baseline and GNN$^+$, we used node degree as the input node features for MINCUT and MST. For Shortest Paths, Effective Resistance and Affinity, we set input node features to be Booleans indicating if the node is a source/destination node or not. See Appendix for further details.

In GNN$^+$, the embeddings across different nodes and different GNNmp blocks need to be “pooled” together and passed onto the MLP modules to get the embeddings for next layer (see e.g. Figure 1). For our theoretical constructions we designed specific pooling operations for specific problems, that show the existence of sample efficient networks. For our practical implementation, we use standard sum and max pooling operations.

Results. We compare the test mean squared errors for the two models across the five datasets. For all the five problem, Table 1 lists the test MSEs and corresponding standard deviations for the two models. As a sanity check, we also plot the

Problem	Label Variance	Avg. MSE (GNN^{mp})	Avg. MSE (GNN^{+})
Shortest Path	7.482	0.975 ± 0.031	0.849 ± 0.022
Effective Resistance	7.949	0.397 ± 0.025	0.187 ± 0.008
Affinity	3.030	0.0025 ± 0.00017	$0.0018 \pm 1.8e{-}05$
MST	4637.4	1011.39 ± 106.94	733.901 ± 30.97
MinCut	11.964	0.963 ± 0.110	0.694 ± 0.07

Table 1: Performance of the GNN^{mp}and GNN^{+} architectures

Dataset	Test Acc. (GNN^{mp})	Test Acc. (GNN^{+})
IMDB-BINARY	0.742 ± 0.09	0.769 ± 0.02
IMDB-MULTI	0.523 ± 0.06	0.527 ± 0.04
COLLAB	0.802 ± 0.02	0.816 ± 0.004
PROTEINS	0.760 ± 0.008	0.765 ± 0.015
NCI1	0.849 ± 0.004	0.851 ± 0.003
PTC	0.686 ± 0.02	0.708 ± 0.018
MUTAG	0.876 ± 0.016	0.898 ± 0.012

Table 2: Performance of the GNN^{mp}and GNN^{+} architectures

variance of the labels in our datasets, which corresponds to the MSE obtained by a naive model that predicts the mean label. We observe significant gains in accuracy of anywhere between 15% relative MSE improvement over the GNN^{mp} baseline (for Shortest Paths) to as much as 108% relative MSE improvement (for Effective Resistance). Note that the naive mean predictor's MSE is at least an order of magnitude larger than all the MSE values for GNN^{mp} and GNN^{+} (except for the MSTdataset, where it is around five times larger - we suspect that the weighted graphs involved in this dataset make this a harder problem).

We posit that these test MSE gains directly stem from the sample-efficiency of GNN^{+} due to low depth and parameters, as captured in Theorems 1,2 and 4 - the most compact GNN^{+} networks that can represent these problems are smaller than the corresponding most compact GNN^{mp} networks. By Theorem 5, such networks will have smaller generalization errors. In Appendix , we also plot the test MSE as a function of number of epochs to show that our models also converge faster than the baseline GNN^{mp}, though we do not have any theoretical justification supporting this observation.

Distance Oracles. Here we evaluate the effectiveness of embeddings produced by GNN^{mp} and GNN^{+} networks to serve as effective distance oracles. A distance oracle is a data structure that can answer approximate shortest path queries quickly and accurately after doing some pre-processing on the graph. There exist many combinatorial algorithms to construct such oracles by pre-processing the graph to compute a k-dimensional sketch (Das Sarma et al. 2010; Thorup and Zwick 2005). At query time the distance between two vertices is then approximated by some ℓ_p distance between the sketches. We evaluate the ability of the embeddings produced by GNNs to serve as effective sketches. For this purpose we implement a state-of-the-art combinatorial sketch computation algorithms of Das Sarma et al. (2010). For sketch sizes in the range $\{\lceil \log n \rceil, 2\lceil \log n \rceil, \ldots, 8\lceil \log n \rceil\}$, where $n = 1000$, we run the combinatorial algorithm to produce the sketch of the required size. At the same time we train a GNN^{mp} and a GNN^{+} network (to minimize MSE as discussed above) to produce embeddings of the same size as that of the sketch. Figure 3 shows the performance of the neural networks as compared to the combinatorial algorithm. As can be see, at embedding size of $3\lceil \log n \rceil$, GNN^{+} networks start outperforming the combinatorial algorithm when run with sketch size $5\lceil \log n \rceil$. Furthermore, GNN^{+} at embedding size of $\lceil \log n \rceil$, already outperforms the combinatorial algorithms for all larger values of the sketch sizes.

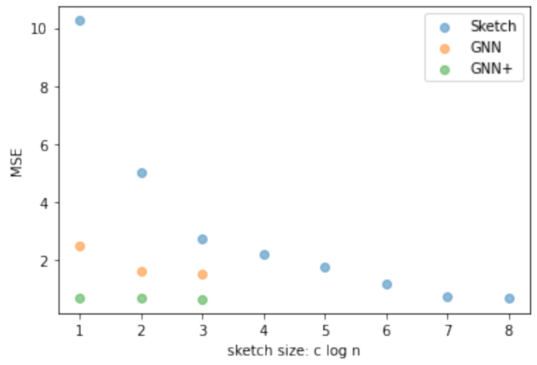


Figure 3: Performance of GNN^{mp} and GNN^{+} as compared to the combinatorial distance oracle of Das Sarma et al. (2010).

Real World Data. While our theoretical results apply to problems such as shortest paths it is a natural question to ask how our proposed GNN^{+} architecture performs on other tasks such as classification. We experiment with the following real world datasets (Yanardag and Vishwanathan 2015) that have been used in recent works for evaluating various GNN architectures (Xu et al. 2019a): 1) IMDB-BINARY and 2) IMDB-MULTI datasets: These are movie collaboration datasets with nodes as actors and the class label being the genre. 3) COLLAB: This is a scientific collaboration dataset with three classes. 4) PROTEINS: This is a bioinformatics dataset with 3 class labels. 5) PTC, 6) NCI1 and 7) MUTAG: These are various datasets of chemical compounds with two class labels each.

We train our GNN^{+} proposed architecture on these graphs using the cross-entropy loss and as before compare with the GIN architecture of (Xu et al. 2019a). We use the same input node features as in (Xu et al. 2019a) and use the same experimental methodology as that for synthetic graphs above. In particular, when tuning hyperparameter tuning we allow the GNN^{mp} architecture to explore depth up to 9 whereas the GNN^{+} architecture is tuned by restricting the depth upto 3. See Appendix for a comparison of the depth used by GNN^{mp} vs. GNN^{+}. The results are summarized in Table 1. As can be seen, in each instance GNN^{+} either outperforms or matches the performance of the GNN^{mp} architecture in terms of final test accuracy. In Appendix we also provide an analysis of the depth used by GNN^{+} and GNN^{mp} on the different datasets.

References

Allamanis, M.; Brockschmidt, M.; and Khademi, M. 2017. Learning to represent programs with graphs. *arXiv preprint arXiv:1711.00740*.

Arora, R.; Basu, A.; Mianjy, P.; and Mukherjee, A. 2016. Understanding deep neural networks with rectified linear units. *arXiv preprint arXiv:1611.01491*.

Babai, L.; Frankl, P.; and Simon, J. 1986. Complexity classes in communication complexity theory. In *27th Annual Symposium on Foundations of Computer Science (sfcs 1986)*, 337–347. IEEE.

Bartlett, P. L.; Foster, D. J.; and Telgarsky, M. J. 2017. Spectrally-normalized margin bounds for neural networks. In *Advances in Neural Information Processing Systems*, 6240–6249.

Bianchi, F. M.; Grattarola, D.; and Alippi, C. 2019. Mincut pooling in graph neural networks. *arXiv preprint arXiv:1907.00481*.

Bieber, D.; Sutton, C.; Larochelle, H.; and Tarlow, D. 2020. Learning to Execute Programs with Instruction Pointer Attention Graph Neural Networks. *arXiv preprint arXiv:2010.12621*.

Bourgain, J. 1985. On Lipschitz embedding of finite metric spaces in Hilbert space. *Israel Journal of Mathematics*, 52(1-2): 46–52.

Chandra, A. K.; Raghavan, P.; Ruzzo, W. L.; Smolensky, R.; and Tiwari, P. 1996. The electrical resistance of a graph captures its commute and cover times. *Computational Complexity*, 6(4): 312–340.

Chen, Z.; Chen, L.; Villar, S.; and Bruna, J. 2020. Can graph neural networks count substructures? *arXiv preprint arXiv:2002.04025*.

Cohen, R.; and Havlin, S. 2003. Scale-free networks are ultrasmall. *Physical review letters*, 90(5): 058701.

Das Sarma, A.; Gollapudi, S.; Najork, M.; and Panigrahy, R. 2010. A sketch-based distance oracle for web-scale graphs. In *Proceedings of the third ACM international conference on Web search and data mining*, 401–410.

Defferrard, M.; Bresson, X.; and Vandergheynst, P. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in neural information processing systems*, 3844–3852.

Dinella, E.; Dai, H.; Li, Z.; Naik, M.; Song, L.; and Wang, K. 2020. Hoppity: Learning graph transformations to detect and fix bugs in programs. In *International Conference on Learning Representations (ICLR)*.

Du, S. S.; Hou, K.; Salakhutdinov, R. R.; Poczos, B.; Wang, R.; and Xu, K. 2019. Graph neural tangent kernel: Fusing graph neural networks with graph kernels. In *Advances in Neural Information Processing Systems*, 5723–5733.

Errica, F.; Podda, M.; Bacciu, D.; and Micheli, A. 2020. A Fair Comparison of Graph Neural Networks for Graph Classification. *ICLR*.

Fey, M.; Eric Lenssen, J.; Weichert, F.; and Müller, H. 2018. Splinecnn: Fast geometric deep learning with continuous b-spline kernels. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 869–877.

Forster, S.; and Nanongkai, D. 2018. A faster distributed single-source shortest paths algorithm. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, 686–697. IEEE.

Garg, V. K.; Jegelka, S.; and Jaakkola, T. 2020. Generalization and Representational Limits of Graph Neural Networks. *ICML*.

Gilmer, J.; Schoenholz, S. S.; Riley, P. F.; Vinyals, O.; and Dahl, G. E. 2017. Neural message passing for quantum chemistry. *arXiv preprint arXiv:1704.01212*.

Giné, E.; and Guillou, A. 2001. On consistency of kernel density estimators for randomly censored data: rates holding uniformly over adaptive intervals. In *Annales de l'IHP Probabilités et statistiques*, volume 37, 503–522.

Hamilton, W.; Ying, Z.; and Leskovec, J. 2017. Inductive representation learning on large graphs. In *Advances in neural information processing systems*, 1024–1034.

Karger, D. R.; and Stein, C. 1996. A new approach to the minimum cut problem. *Journal of the ACM*, 43: 601–640.

Kipf, T. N.; and Welling, M. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*.

Linial, N.; London, E.; and Rabinovich, Y. 1995. The geometry of graphs and some of its algorithmic applications. *Combinatorica*, 15(2): 215–245.

Long, P. M.; and Sedghi, H. 2020. Generalization bounds for deep convolutional neural networks. arXiv:1905.12600.

Loukas, A. 2020a. How hard is to distinguish graphs with graph neural networks? Technical report.

Loukas, A. 2020b. What graph neural networks cannot learn: depth vs width. *ICLR*.

Maron, H.; Ben-Hamu, H.; Serviansky, H.; and Lipman, Y. 2019. Provably powerful graph networks. In *Advances in Neural Information Processing Systems*, 2156–2167.

Matoušek, J. 1996. On the distortion required for embedding finite metric spaces into normed spaces. *Israel Journal of Mathematics*, 93(1): 333–344.

Mesquita, D.; Souza, A. H.; and Kaski, S. 2020. Rethinking pooling in graph neural networks. *arXiv preprint arXiv:2010.11418*.

Nanongkai, D. 2014. Distributed approximation algorithms for weighted shortest paths. In *Proceedings of the forty-sixth annual ACM symposium on Theory of computing*, 565–573.

Panigrahy, R.; Najork, M.; and Xie, Y. 2012. How user behavior is related to social affinity. In *Proceedings of the fifth ACM international conference on Web search and data mining*, 713–722.

Peleg, D. 2000. *Distributed computing: a locality-sensitive approach*. SIAM.

Sarma, A. D.; Holzer, S.; Kor, L.; Korman, A.; Nanongkai, D.; Pandurangan, G.; Peleg, D.; and Wattenhofer, R. 2012. Distributed verification and hardness of distributed approximation. *SIAM Journal on Computing*, 41(5): 1235–1265.

Sato, R.; Yamada, M.; and Kashima, H. 2019. Approximation ratios of graph neural networks for combinatorial problems. In *Advances in Neural Information Processing Systems*, 4081–4090.

Simonovsky, M.; and Komodakis, N. 2017. Dynamic edge-conditioned filters in convolutional neural networks on graphs. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 3693–3702.

Sollin, M. 1965. La trace de canalisation. *Programming, Games, and Transportation Networks*.

Thorup, M.; and Zwick, U. 2005. Approximate distance oracles. *Journal of the ACM (JACM)*, 52(1): 1–24.

Veličković, P.; Cucurull, G.; Casanova, A.; Romero, A.; Lio, P.; and Bengio, Y. 2017. Graph attention networks. *arXiv preprint arXiv:1710.10903*.

Weisfeiler, B.; and Lehman, A. A. 1968. A reduction of a graph to a canonical form and an algebra arising during this reduction. *Nauchno-Technicheskaya Informatsia*, 2(9): 12–16.

Xu, K.; Hu, W.; Leskovec, J.; and Jegelka, S. 2019a. How Powerful are Graph Neural Networks? *ICLR*.

Xu, K.; Li, J.; Zhang, M.; Du, S. S.; Kawarabayashi, K.-i.; and Jegelka, S. 2019b. What Can Neural Networks Reason About? *arXiv preprint arXiv:1905.13211*.

Yanardag, P.; and Vishwanathan, S. 2015. Deep Graph Kernels. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '15, 1365–1374.

Ying, Z.; You, J.; Morris, C.; Ren, X.; Hamilton, W.; and Leskovec, J. 2018. Hierarchical graph representation learning with differentiable pooling. In *Advances in neural information processing systems*, 4800–4810.

You, J.; Ying, Z.; and Leskovec, J. 2020. Design space for graph neural networks. *Advances in Neural Information Processing Systems*, 33.

Zhang, S.; Wang, M.; Liu, S.; Chen, P.-Y.; and Xiong, J. 2020. Fast Learning of Graph Neural Networks with Guaranteed Generalizability: One-hidden-layer Case. In *International Conference on Machine Learning*, 11268–11277. PMLR.