What graph neural networks *cannot* learn: depth vs width

Andreas Loukas
andreas.loukas@epfl.ch
École Polytechnique Fédérale Lausanne

July 9, 2019

Abstract

This paper studies the capacity limits of graph neural networks (GNN). Rather than focusing on a specific architecture, the networks considered here are those that fall within the message-passing framework, a model that encompasses several state-of-the-art networks. Two main results are presented. First, GNN are shown to be Turing universal under sufficient conditions on their depth, width, node identification, and layer expressiveness. In addition, it is discovered that GNN can lose a significant portion of their power when their depth and width is restricted. The proposed impossibility statements stem from a new technique that enables the re-purposing of seminal results from theoretical computer science. This leads to lower bounds for an array of decision, optimization, and estimation problems involving graphs. Strikingly, several of these problems are deemed impossible unless the product of a GNN's depth and width exceeds the graph size; this dependence remains significant even for tasks that appear simple or when considering approximation.

1 Introduction

A fundamental question in machine learning is to determine what a model *can* and *cannot* learn. In deep learning, there has been significant research effort in establishing positive results. For instance, it is now well known that feed-forward neural networks of sufficient depth and width are universal function approximators [1, 2, 3]. More recently, we have seen the first results studying the universality of graph neural networks, i.e., neural networks that take graphs as input. Maron et al. [4] derived a universal approximation theorem over invariant functions targeted towards deep networks whose layers are linear and equivariant to permutation of their input. Universality was also shown for equivariant functions by Keriven and Peyré [5], though this time under a specific shallow architecture. Expanding upon deep sets [6], Xu et al. [7] also established the universality of a single graph neural network layer consisting of a sum aggregator, a result that was later expanded by Seo et al. [8].

Universality statements allow us to grasp the capacity of models in the limit. In theory, given enough data and the right learning algorithm, a universal network will be able to solve any task that it is presented with. Nevertheless, the insight brought by such results can also be limited. Knowing that a sufficiently large network can be used to solve any problem does not reveal much about how neural networks should be designed in practice. It also certainly cannot guarantee that said network will be able to solve a given task given a particular learning algorithm, such as stochastic gradient descent.

On the other hand, it is often easier to obtain insights about models by studying their limitations. After all, the knowledge of what *cannot* be learned by a network of specific characteristics applies independently of the training procedure. Further, by helping us comprehend the difficulty of a

task in relation to a model, impossibility results can yield practical advice on how to select model hyperparameters.

Take, for instance, the problem of graph classification. Training a graph classifier entails identifying what constitutes a class, i.e., finding properties shared by graphs in one class but not the other, and then deciding whether new graphs abide to said learned properties. However, if the aforementioned decision problem is shown to be impossible by a graph neural network of certain depth (and the test set is sufficiently diverse) then we can be certain that the same network will not learn how to classify the test set correctly, independently of which learning algorithm is employed. We should, therefore, focus on networks deeper that the lower bound when performing experiments.

1.1 Contributions

This paper studies the computational capacity limits of graph neural networks. Rather than focusing on a specific architecture, the networks considered are those that fall within the message-passing framework of Gilmer et al. [9]. This model is chosen as it is sufficiently general to encompass several state-of-the-art networks, including GCN [10], ChebyNet [11], gated graph neural networks [12], molecular fingerprints [13], interaction networks [14], molecular graph convolutions [15], among many others. I refer to such graph neural networks as GNN in the following.

The provided contributions are two-fold:

1. What GNN can learn. Section 3 derives sufficient conditions such that a GNN can compute *any* function on its input that is computable by a Turing machine. This result differs from recent universality results [4, 5] that considered approximation (rather than computability) over specific classes of functions (invariant and equivariant) and particular architectures. The claim follows in a straightforward manner by establishing the Turing equivalence of GNN with LOCAL [16, 17, 18], a classical model in distributed computing that is itself Turing universal. In a nutshell, GNN are shown to be universal if four strong conditions are met: (*i*) there are enough layers, (*ii*) said layers have sufficient width, (*iii*) nodes can uniquely distinguish each other, and (*iv*) the functions computed within each layer are sufficiently expressive.

2. What GNN cannot learn. To obtain more insight, Section 4 studies the implications of restricting the depth d and width w of GNN that do not use a readout function. Specifically, it is shown that GNN lose a significant portion of their power when the product dw is restricted. The analysis relies on a new lemma that enables the translation of impossibility results from LOCAL to GNN. The main benefit of this approach is that it allows one to re-purpose several seminal lower bounds [19, 20, 21, 22, 23] from theoretical computer science to the graph neural network setting.

Let G be the input of the neural network. Lower bounds for the following problems are presented:

- detecting whether G contains a *cycle of specific length*;
- verifying whether a given subgraph of G is *connected*, *contains a cycle*, is a *spanning tree*, is *bipartite*, is a *simple path*, corresponds to a *cut* or *Hamiltonial cycle* of G;
- approximating the *shortest path* between two vertices, the *minimum cut*, and the *minimum spanning tree*;
- finding a *maximum independent set*, a *minimum vertex cover*, or a *chromatic coloring* of G;
- computing or approximating the *diameter* and *girth* of G;

The bounds are summarized in Table 1 and the problem definitions can be found in Appendix A.

Though formulated in a graph-theoretic sense, the above problems are intimately linked to machine learning on graphs. Detection, verification, and computation problems are relevant to graph (and node)

problem	*bound*	*problem*	*bound*
cycle detection (odd)	$dw = \Omega(n/\log n)$	shortest path	$d\sqrt{w} = \Omega(\sqrt{n}/\log n)$
cycle detection (even)	$dw = \Omega(\sqrt{n}/\log n)$	maximum independent set	$dw = \Omega(n^2/\log^2 n)$ for $w = O(1)$
subgraph verification*	$d\sqrt{w} = \Omega(\sqrt{n}/\log n)$	minimum vertex cover	$dw = \Omega(n^2/\log^2 n)$ for $w = O(1)$
minimum spanning tree	$d\sqrt{w} = \Omega(\sqrt{n}/\log n)$	chromatic coloring	$dw = \Omega(n^2/\log^2 n)$ for $w = O(1)$
minimum cut	$d\sqrt{w} = \Omega(\sqrt{n}/\log n)$	girth 2-approximation	$dw = \Omega(\sqrt{n}/\log n)$
diameter estimation	$dw = \Omega(n/\log n)$	diameter 3/2-approximation	$dw = \Omega(\sqrt{n}/\log n)$

Table 1: Summary of main results. Sugraph verification entails verifying one of the following predicates for a given subgraph H of G: is connected, contains a cycle, forms a spanning tree of G, is bipartite, cuts G, is an s-t cut of G. All problems are defined in Appendix A.*

classification: knowing what properties of a graph (subgraph) a GNN cannot see informs us also about which features of a graph can it extract. Further, there have been multiple attempts to use graph neural networks in order to devise heuristics for hard or computationally expensive graph-theoretic optimization problems [24, 25], such as the ones discussed above. The presented results can then be taken as a worst-case analysis for such algorithms.

To the extent of my knowledge, these are the first impossibility results for graph neural networks. The closest work is that of Xu et al. [7], which established the equivalence of GNN to the Weisfeiler-Lehman graph isomorphism test. However, the latter concerned different problems (graph isomorphism), did not propose any quantitative lower bounds, and also did not analyze the significance of depth and width.

1.2 Implications

The results of this paper carry several intriguing implications.

To start with, it is shown that the product dw of depth and width of a GNN plays a significant role in determining capacity. Solving many problems is shown to be impossible unless

$$dw = \tilde{\Omega}(n^{\delta}), \quad \text{where} \quad \delta \in (1/2, 2],$$

n is the number of nodes of the graph, and $f(n) = \tilde{\Omega}(g(n))$ is interpreted as $f(n)$ being, up to logarithmic factors, larger than $g(n)$ as n grows. This reveals a direct trade-off between the depth and width of a graph neural network.

Counter-intuitively, the dependence on n can be significant even if the problem appears local in nature or one only looks for approximate solutions. For example, to detect whether G contains a short cycle of odd length cannot be done unless $dw = \tilde{\Omega}(n)$. Approximation also does not help significantly. Computing the graph diameter requires $dw = \tilde{\Omega}(n)$ and this reduces to $dw = \tilde{\Omega}(\sqrt{n})$ if we are satisfied by any 3/2-factor approximation. Further, it is impossible to approximate within *any* constant factor the shortest path, the minimum cut, and the minimum spanning tree unless $d\sqrt{w} = \tilde{\Omega}(\sqrt{n})$. It should be remarked that all three of these problems have known polynomial time solutions.

Finally, for truly hard problems, the product of depth and width may even need to be super-linear on n. Specifically, it is shown that, even if the layers of the GNN are allowed to take exponential time, solving NP-hard problems, such as the minimum independent set, the minimum vertex cover, and the chromatic coloring, necessitates $d = \tilde{\Omega}(n^2)$ depth for any constant width network.

1.3 Limitations

There are three main limitations inherent to the present work:

Perhaps the most important limitation is that all lower bounds are of a worst-case nature. In this work, a problem is deemed impossible for a given depth and width if there exists a graph for which it cannot be solved. Therefore, the impossibility may be annulled if we take a reduced set of graphs into consideration.

Second, the analysis does not take into account the specific form of the functions used by each network layer. Instead, it is assumed that each layer is sufficiently powerful to compute any function of its input. Fortunately, this strong assumption does not limit the applicability of the presented results. This is simply because all lower bounds that hold with layers of unbounded capacity also apply to those are limited computationally. What is perhaps surprising is that, even with unbounded capacity layers, many seemingly simple problems are shown to be unsolvable by networks of small depth and width.

Lastly, in the following, it is assumed that nodes can uniquely identify each other. Since the use of identifiers is not typical in modern architectures, the assumption can be perceived as a limitation. However, similar to the unbounded computation assumption, if a problem cannot be solved by a graph neural network in the studied setting, it also cannot be solved without identifiers. Thus, the presented bounds also apply to anonymous networks.

Notation. I consider directed graphs $G = (\mathcal{V}, \mathcal{E})$ consisting of $n = |\mathcal{V}|$ nodes and $m = |\mathcal{E}|$ edges. The edge going from v_j to v_i is written as $e_{i\leftarrow j}$. The incoming neighborhood $\mathcal{I}_i$ of a node $v_i \in \mathcal{V}$ consists of all nodes v_j for which $e_{i\leftarrow j} \in \mathcal{E}$. Analogously, the outgoing neighborhood of v_i is defined as $\mathcal{O}_i = \{v_j \in \mathcal{V} : e_{j\leftarrow i} \in \mathcal{E}\}$. The degree $\deg_i$ of v_i is the sum of its incoming $\deg_i^{\text{in}}$ and outgoing $\deg_i^{\text{out}}$ degrees. Further, Δ is the maximum degree of all nodes in the graph and the graph diameter δ_G is the length of the longest shortest path between any two nodes. By adding a self-loop $e_{i\leftarrow i}$ to every node v_i, the graph $G^* = (\mathcal{V}, \mathcal{E}^*)$ is constructed whose neighborhood sets are given by $\mathcal{I}_i^* = \mathcal{I}_i \cup v_i$ and $\mathcal{O}_i^* = \mathcal{O}_i \cup v_i$.

2 The graph neural network computational model

Graph neural networks are parametric and differentiable algorithms designed for graph-structured problems. Their input usually consists of a graph and a set of node attribute vectors $a_1, \ldots, a_n$, each encoding relevant information about the role of a node v_i in a given task, as well as the node's degree. In certain situations, the input also includes edge attribute vectors $a_{i\leftarrow j}$, where $e_{i\leftarrow j} \in \mathcal{E}$.

Numerous variants of graph neural networks have been proposed. Putting aside the particularities of each, most adhere to a common logic: Each node can be thought of as a separate entity that gradually transforms its input by applying successive non-linear transformations. However, rather than acting independently, nodes also share parts of their state with their neighbors. This allows information to flow along the graph and enables the nodes to make decisions jointly.

Model 8 formalizes the graph neural network operation by placing it in a message passing computational model [9]. As seen, the computation proceeds in layers, within which two things occur:

- A message $m_{i\leftarrow j}$ is passed along each directed edge $e_{i\leftarrow j} \in \mathcal{E}$ going from v_j to v_i.
- Each node updates its internal representation $x_i^{(l)}$ by aggregating its state with the messages sent by its incoming neighbors $v_j \in \mathcal{I}_i$.

The network output can be either of two things: a vector x_i for each node v_i or a single vector x_G obtained by combining the representations of all nodes using a readout function. Vectors x_i/x_G could be scalars (node/graph regression), binary variables (node/graph classification) or multi-dimensional (node/graph embedding). I use the symbols GNN_n and GNN_g, respectively, to distinguish between the two models.

Computational model 1 GNN

Initialization: Set $x_i^{(0)} = a_i$ for all $v_i \in \mathcal{V}$.
for layer $\ell = 1, \ldots, d$ **do**
  for every edge $e_{i \leftarrow j} \in \mathcal{E}^*$ (in parallel) **do**

$$m_{i \leftarrow j}^{(\ell)} = \textsc{Msg}_\ell \left(x_i^{(\ell-1)}, x_j^{(\ell-1)}, v_i, v_j, a_{i \leftarrow j} \right)$$

  for every node $v_i \in \mathcal{V}$ (in parallel) **do**

$$x_i^{(\ell)} = \textsc{Up}_\ell \Big(\sum_{v_j \in \mathcal{I}_i^*} m_{i \leftarrow j}^{(\ell)} \Big)$$

Set $x_i = x_i^{(d)}$.
return Either x_i for every $v_i \in \mathcal{V}$ (GNN_n) or $x_G = \textsc{Read}\left(\{x_i : v_i \in \mathcal{V}\}\right)$ (GNN_g).

2.1 Design choices

It can be observed that the operation of a GNN is primarily determined by the *messaging*, *update*, and *readout* parametric functions.

Message and update functions. I assume that each of the Msg and Up are instantiated by feed-forward neural networks. Thus, by the universal approximation theorem and its variants [1, 2], they can be used to approximate any general function that maps vectors onto vectors, given sufficient depth and width. The use of feed-forward networks is quite common in the literature [3, 25].

Readout function. Function Read is useful when one needs to solve a global graph problem or to retrieve a representation that is invariant of the number of nodes. The function takes as an input a *multiset*, i.e., a set with (possibly) repeating elements, and returns a vector. In this particular instance, the multiset has as elements the node representations $x_1, \ldots, x_n$ learned by the preceding GNN_n. Commonly, Read is chosen to be a dimension squashing operator (combining all n vectors) followed by a feed-forward neural network. The squashing operator can be a sum [7], mean [10], or even histogram [8] over all x_i.

Depth and width. Another crucial aspect of the GNN design is choosing the network *depth* and *width*. The depth d is equal to the number of layers of the network. Larger depth means that each node has the opportunity to learn more about the rest of the graph (i.e., it has a larger receptive field). The width w of a GNN is equal to the largest dimension of state $x_i^{(l)}$ over all layers l and nodes $v_i \in \mathcal{V}$. Assuming that each variable manipulated by the network is represented in finite-precision using $p = \Theta(\log n)$ bits, each node in a GNN requires $\Theta(w \log n)$ bits to store its state. Further, to store the entire state of the forward pass of a graph neural network one needs at least $\Omega(dw \cdot n \log n)$ bits, even if the feed-forward networks instantiating MSG_l and UP_l consist of a single layer. Hence, to avoid a quadratic dependency on the number of nodes, one should choose the depth and width satisfying $dw = o(n)$.[1] Unfortunately, it will be seen in Section 4 that many real problems cannot be solved by a network that has a nearly linear memory footprint.

Node identification. The capacity of a graph neural network depends on how nodes distinguish each other: generally, it will be assumed that each node possesses a globally unique identifier. By appending

[1] The little-o notation $f(n) = o(g(n))$ means that for every positive constant c there exists a constant N such that $|f(n)| \leq cg(n)$ for all $n \geq N$.

this identifier to outgoing messages, nodes can learn to distinguish between their neighbors. To keep things concise, I overload the notation v_i to also refer to v_i's unique identifier.

3 GNN are computationally universal

As a first step, this section studies the expressive power of graph neural networks. It is demonstrated that a network is universal if four conditions are met: it has enough layers of sufficient width, nodes can uniquely distinguish each other, and the functions computed within each layer are sufficiently expressive.

The derivation is elementary and consists of two steps. In Section 3.2, it is established that GNN_n is equivalent to LOCAL, a classical model used in the study the distributed algorithms. In Section 3.3 it is noticed that LOCAL is Turing universal, meaning that it can compute anything that a Turing machine can compute when given an attributed graph as input. The universality of GNN comes as a direct consequence.

3.1 The LOCAL computational model

A fundamental question in theoretical computer science is determining what can and cannot be computed efficiently by a distributed algorithm. The LOCAL model, initially studied by Angluin [16], Linial [17], and Naor and Stockmeyer [18], provides a common framework for analyzing such algorithms.

Being inspired mainly from networked systems, in LOCAL, a graph plays a double role: it is both the input of the system and captures the network topology of the distributed system that solves the problem. In this spirit, the nodes of the graph are here both the machines where computation takes place as well as the variables of the graph-theoretic problem we wish to solve—similarly, edges model communication links between machines as well as relations between nodes.

In LOCAL, each node $v_i \in \mathcal{V}$ is given a problem specific local input and has to produce a local output. The input contains necessary the information that specifies the problem instance. It is also usually assumed that each node is aware of its degree from the start. All nodes execute the same algorithm, they are fault-free, and they may or may not be provided with unique identities.

The computation starts simultaneously and unfolds in synchronous rounds $l = 1, \ldots, d$. Three things can occur within each round:

1. Each node receives a string from its incoming neighbors.
2. Each node updates its internal state by performing some local computation.
3. Each node sends a string to every one of its outgoing neighbors.

Therefore, instantiating LOCAL entails choosing how each node will update its state and what will it send to its neighbors.

A pseudo-code description is given in Model 2. Variables $s_i^{(l)}$ and $s_{i\leftarrow j}^{(l)}$ refer respectively to the state of v_i in round l and to the message sent by v_j to v_i in the same round. Both are represented as strings. Further, $\textsc{Alg}_l^1$ and $\textsc{Alg}_l^2$ are algorithms computed locally by a Turing machine running on node v_i. Before any computation is done, each node v_i is aware of its own attribute a_i as well as of all edge attributes $\{a_{i\leftarrow j} : v_j \in \mathcal{I}_i^*\}$.

Message size restrictions. Since LOCAL is mainly used to study the impact of locality on distributed decision making, no restrictions are placed by default on the message size, which is assumed to be

Computational model 2 LOCAL (computed distributedly by each node $v_i \in \mathcal{V}$).

Initialization: Set $s^{(0)}_{i\leftarrow i} = (a_i, v_i)$ and $s^{(0)}_{i\leftarrow j} = (a_j, v_j)$ for all $e_{i\leftarrow j} \in \mathcal{E}$.
for round $\ell = 1, \dots, d$ **do**
Receive $s^{(\ell-1)}_{i\leftarrow j}$ from $v_j \in \mathcal{I}^*_i$, compute

$$s^{(\ell)}_i = \textsc{Alg}^1_\ell \left(\left\{ \left(s^{(\ell-1)}_{i\leftarrow j},\, a_{i\leftarrow j} \right) : v_j \in \mathcal{I}^*_i \right\},\, v_i \right),$$

and send $s^{(\ell)}_{j\leftarrow i} = \textsc{Alg}^2_\ell \left(s^{(\ell)}_i, v_i \right)$ to $v_j \in \mathcal{O}^*_i$.
return $s^{(d)}_i$

unbounded. More generally, it will be assumed that the message size at each round is at most b bits. This model goes by the name $\mathsf{CONGEST}_b$ in the distributed systems literature. For simplicity, in the following I refrain from distinguishing the two: unless the text mentions otherwise, I assume that also in LOCAL the message size is at most b.

Node identification. Similarly to the GNN model, nodes can distinguish between their neighbors and process information accordingly. This can be seen in Model 2, by noticing that both $\textsc{Alg}^1_l$ and $\textsc{Alg}^2_l$ accept as arguments node identifiers (symbolized by v_i).

3.2 Turing equivalence

The reader might have observed that LOCAL resembles closely GNN_n in its structure, with only a few minor exceptions: firstly, whereas in LOCAL an algorithm may utilize messages in any way it chooses, graph neural networks always sum received messages before any local computation. The two models also differ in the arguments of the messaging function and the choice of information representation (string versus vector).

As the following result shows, the differences between GNN_n and LOCAL are inconsequential when seen from the perspective of computational capacity.

Proposition 3.1. *The LOCAL and GNN_n computational models are Turing equivalent.*

Proof. To derive a proof, I will express the state of node v_i in the two models in the same form. It is not difficult to see that for each layer of the GNN_n one has

$$\begin{aligned}
x^{(l)}_i &= \textsc{Up}_\ell \Big(\sum_{v_j \in \mathcal{I}^*_i} m^{(\ell)}_{i\leftarrow j} \Big) && \text{(by definition)} \\
&= \textsc{Up}_\ell \Big(\sum_{v_j \in \mathcal{I}^*_i} \textsc{Msg}_\ell \left(x^{(\ell-1)}_i, x^{(\ell-1)}_j, v_i,\, v_j,\, a_{i\leftarrow j} \right) \Big) && \text{(substituted } m^{(\ell)}_{i\leftarrow j}\text{)} \\
&= \textsc{Agg}_\ell \left(\left\{ \left(x^{(\ell-1)}_i, x^{(\ell-1)}_j, v_i,\, v_j,\, a_{i\leftarrow j} \right) : v_j \in \mathcal{I}^*_i \right\} \right), && \text{(from [7, Lemma 5])}
\end{aligned}$$

where $\textsc{Agg}_\ell$ is an *aggregation function*, i.e., a map from the set of multisets onto some vector space. In the last step, I used a result of Xu et al. [7] stating that each aggregation function can be decomposed as an element-wise function over each element of the multiset, followed by summation of all elements, and then a final function.

Similarly, one may write:

$$\begin{aligned} s_i^{(\ell)} &= \textsc{Alg}_\ell^1\left(\left\{\left(s_{i\leftarrow j}^{(\ell-1)},\, a_{i\leftarrow j}\right) : v_j \in \mathcal{I}_i^*\right\},\, v_i\right) && \text{(by definition)} \\ &= \textsc{Alg}_\ell^1\left(\left\{\left(\textsc{Alg}_{\ell-1}^2\left(s_j^{(\ell-1)}, v_j\right),\, a_{i\leftarrow j}\right) : v_j \in \mathcal{I}_i^*\right\},\, v_i\right) && \text{(substituted } s_{i\leftarrow j}^{(\ell-1)}\text{)} \\ &= \textsc{Alg}_\ell\left(\left\{\left(s_j^{(\ell-1)}, v_i,\, v_j,\, a_{i\leftarrow j}\right) : v_j \in \mathcal{I}_i^*\right\}\right), \end{aligned}$$

with the last step following by restructuring the input and defining $\textsc{Alg}_\ell$ as the Turning machine that simulates the action of both $\textsc{Alg}_\ell^2$ and $\textsc{Alg}_{\ell-1}^1$.

Since one may encode any vector into a string and vice versa, w.l.o.g. one may assume that the state of each node in LOCAL is encoded as a vector x_i. Then, to complete the proof, one still needs to demonstrate that the functions

$$\textsc{Agg}\left(\{(x_i, x_j, v_i,\, v_j,\, a_{i\leftarrow j}) : v_j \in \mathcal{I}_i^*\}\right) \quad \text{and} \quad \textsc{Alg}\left(\{(x_j, v_i,\, v_j,\, a_{i\leftarrow j}) : v_j \in \mathcal{I}_i^*\}\right)$$

are equivalent (in the interest of brevity the layer/round indices have been dropped). If this holds then each layer of GNN_n is equivalent to a round of LOCAL and the claim follows.

I first note that, since its input is a multiset, $\textsc{Alg}_l$ is also an aggregation function. To demonstrate equivalence, one thus needs to show that, despite not having identical inputs, each of the two aggregation functions can be used to replace the other. For the forward direction, it suffices to show that for every aggregation function Agg there exists Alg with the same output. Indeed, one may always construct $\textsc{Alg} = \textsc{Agg} \circ g$, where g takes as input $\{(x_j, v_i,\, v_j,\, a_{i\leftarrow j}) : v_j \in \mathcal{I}_i^*\}$, identifies x_i (by searching for $v_i,\, v_i$) and appends it to each element of the multiset yielding $\{(x_i, x_j, v_i,\, v_j,\, a_{i\leftarrow j}) : v_j \in \mathcal{I}_i^*\}$. The backward direction can also be proven with an elementary construction: given Alg, one sets $\textsc{Agg} = \textsc{Alg} \circ h$, where h deletes x_i from each element of the multiset. □

3.3 Turing universality

It is well understood in machine learning that the output of any node in GNN_n is a function of the input (including the graph and attributes) within a d-radius neighborhood of the node—referred to as the receptive field. In the context of distributed algorithms, it is also known that, as long as d is larger than the graph diameter δ_G, every node in a GNN_n can effectively make decisions based on the entire graph and attributes. This elementary observation yields the following corollary.

Corollary 3.1. *GNN_n can compute any Turing computable function over attributed graphs if the following conditions are jointly met:*

- *each node is uniquely identified;*
- $\textsc{Msg}_l$ *and* $\textsc{Up}_l$ *are Turing-complete;*
- *the network consists of $d \geq \delta_G$ layers;*
- *the width is unbounded.*

Proof. In the LOCAL model the reasoning is simple: suppose that the graph is represented by a set of edges and further consider that $\textsc{Alg}_l$ amounts to a union operation. Then in $d = \delta_G$ rounds, the state of each node will contain the entire graph. The function $\textsc{Alg}_d^1$ can then be used to make the final computation. This argument also trivially holds for node attributes. The universality of GNN_n then follows by the Turing equivalence of LOCAL and GNN_n. □

So, if computation and memory is not an issue, one may construct GNN that effectively compute *any* computable function w.r.t. the graph and attributes.

Why is this result relevant? From a cursory review, it might seem that universality is an abstract result with little implication to machine learning architects. After all, the utility of GNN is usually determined not with regards to its computational capacity but with its ability to generalize to unseen examples. Nevertheless, it can be argued that universality is an essential property of a good learning model. This is for two main reasons:

First, universality guarantees that the learner is not systematically biased. In theory, given enough data and the right learning algorithm, a universal GNN will be able to solve any task that it is presented with. On the other hand, a more computationally restricted learner has blind-spots in its hypothesis space: no matter how good the optimization algorithm is, how rich the dataset, and how overparameterized the network is, there will always be functions which the learner cannot learn.

Second, a universality result allows us to get a glimpse on how the complexity of the learner's hypothesis space is affected by different design choices. For instance, Corollary 3.1 puts forth four necessary conditions for universality: the GNN should be sufficiently deep and very wide, nodes should be able to uniquely and consistently identify each other, and finally, the functions utilized in each layer should be sufficiently complex.

The following section delves further into the importance of these universality conditions. It will be shown that GNN lose a significant portion of their power when conditions of Corollary 3.1 are relaxed.

4 Impossibility results as a function of depth and width

This section analyzes the effect of depth and width in the computational capacity of a graph neural network. The impossibility results presented are of a worst-case flavor: a problem will be deemed impossible for a given depth and width if there exists a graph for which it cannot be solved. Thus, the impossibility may be annulled if one takes a reduced set of graphs into consideration.

To obtain meaningful bounds, I focus on networks without a readout function. The reason is simple: given a sufficiently powerful readout function, a GNN_g of $d = 1$ depth and $O(\Delta)$ width can be used to compute any Turing computable function. The nodes should simply gather one hop information about their neighbors; the readout function can then reconstruct the problem input based on the collective knowledge and apply any computation necessary. Arguably, this is an unsatisfactory result as it deviates from how graph neural networks are meant to function. In addition, it will be assumed that the node and edge attributes can only encode the problem input (i.e., they do not reveal anything about the problem solution) and that nodes do not have access to a random generator (so that the output is deterministic).

Before presenting the main findings, it is apt to mention the following well-known impossibility result: a network whose depth is strictly smaller than the graph diameter δ_G cannot compute quantities that are inherently global [26]. For instance, the nodes cannot learn to count the total number of nodes/edges in G, and they cannot approximate any non-trivial[2] function $f(a_1, \ldots, a_n)$ of node attributes. The limitation of this folklore result is that it does not expose a relation between network depth and width, as well as that it requires one to understand if a given problem is global or not.

4.1 How to transfer lower bounds between LOCAL and GNN_n

I introduce the following result:

[2]Non-trivial here means that the partial derivatives of f w.r.t. its inputs are not uniformly zero.

Lemma 4.1. *If a problem P cannot be solved in less than d rounds in* LOCAL *using messages of at most b bits, then P cannot be solved by a* GNN_n *of width $w \leq (b - \log_2 n)/p = O(b/\log n)$ and depth d.*

Proof. First note that, since the GNN_n and LOCAL models are Turing equivalent, if no further memory/width restrictions are placed, an impossibility for one implies also an impossibility for the other. It can also be seen in the proof of Proposition 3.1 that there is a one to one mapping between the internal state of each node at each level between the two models (i.e., variables $x_i^{(l)}$ and $s_i^{(l)}$). As such, impossibility results that rely on restrictions w.r.t. state size also transfer between the models.

To proceed, I demonstrate that a depth lower bound in the LOCAL model with bounded *message size* also implies the existence of a lower bound in the LOCAL model with a bounded *state size*—with this result in place, the proof of the main claim follows directly. As in the statement of the lemma, one starts by assuming that P cannot be solved in less than d rounds when messages are bounded to be at most b bits. Then, for the sake of contradiction, it is supposed that there exists an algorithm $A \in \mathsf{LOCAL}$ that can solve P in less than d rounds with a state of at most c bits, but unbounded message size. I argue that the existence of this algorithm also implies the existence of a second algorithm A' whose messages are bounded by $c + \log_2 n$: since each message $s_{j \leftarrow i}^{(l)}$ is the output of a universal Turing machine $\textsc{Alg}_l^2$ that takes as input the tuple $(s_i^{(l)}, v_i)$, algorithm A' directly sends the input and relies on the universality of $\textsc{Alg}_{l+1}^1$ to simulate the action of $\textsc{Alg}_l^2$. The message size bound follows by adding the size c of the state with that of representing the node id ($\log_2 n$ bits suffice for unique node identifiers). This line of reasoning leads to a contradiction when $c \leq b - \log_2 n$, as it implies that there exists an algorithm (namely A') that can solve P in less than d rounds while using messages of at most b bits. Hence, no algorithm whose state is less than $b - \log_2 n$ bits can solve P in LOCAL. □

The significance of Lemma 4.1 is that it shows us how to translate impossibility results from the well studied LOCAL model to GNN_n. Intriguingly, the lemma reveals a connection between the message size in LOCAL and the width of a GNN_n: if a problem cannot be solved in LOCAL with messages of b bits and d rounds, then also no d depth GNN_n whose width is at most $O(b/\log n)$, will be able to solve it. The $p = \Theta(\log n)$ factor corresponds to the length of the binary representation of every variable.[3]

4.2 Decision problems

I first consider problems where one needs to decide whether a given graph satisfies a certain property [27]. Concretely, given a decision problem P and a graph G, the GNN_n should output

$$x_i \in \{\mathit{true}, \mathit{false}\} \quad \text{for all} \quad v_i \in \mathcal{V}.$$

The network then accepts the premise if the logical conjunction of $\{x_1, \ldots, x_n\}$ is *true* and rejects it otherwise.

Such decision problems are of particular interest to machine learning as they are intimately connected to graph classification. A graph classification problem entails identifying what constitutes a class from some training set *and* using said learned definition to classify graphs sampled from the test set. Instead, I will suppose that the class definition is known to the classifier and focus on the corresponding decision problem. As a consequence, every lower bound presented below for a decision problem must also be respected by a GNN_n classifier that attains zero error on the corresponding graph classification problem.

Subgraph detection. In this type of problems, the objective is to decide whether G contains a subgraph belonging to a given family. I focus specifically on detecting whether G contains a cycle C_k, i.e., a simple undirected graph of k nodes each having exactly two neighbors. As the following result shows, cycle detection is a harder problem than one might have guessed:

[3] The precision needs to depend logarithmically on n for the node identifiers to be unique.

Corollary 4.1 (Cycle detection [21, 22])**.** *There exists graph G on which every GNN_n of width w requires depth at least*

$$d = \Omega\left(\frac{\sqrt{n}}{w \log n}\right) \quad \text{and} \quad d = \Omega\left(\frac{n}{w \log n}\right)$$

to detect if G contains a cycle C_k for even $k \geq 4$ and odd $k \geq 5$, respectively.

It is striking that, even for small cycles, the product of depth and width should exhibit an almost linear dependence on the number of nodes (see also Appendix B). This is not necessarily the case for every problem: indeed, one may solve the k-tree detection problem (i.e., does G contain an subgraph that is isomorphic to a pre-specified tree of k nodes?) in $O(k\,2^k)$ rounds [22, 28], yielding a bound that is independent of n when k is a constant.

Subgraph verification. Suppose that GNN_n is given a subgraph $H = (\mathcal{V}_H, \mathcal{E}_H)$ of G in its input. This could, for instance, be achieved by selecting the attributes of each node and edge to be a one-hot encoding of their membership on $\mathcal{V}_H$ and $\mathcal{E}_H$, respectively. The question considered is whether the neural network can verify a certain property of H. More concretely, we are interested in whether there exists a graph neural network that can successfully verify H as belonging to a specific family of graphs (possibly defined w.r.t. G). In contrast to the standard decision paradigm, here every node should reach the same decision—either accepting or rejecting the hypothesis.

The following result can be obtained as a consequence of the seminal work by Sarma et al. [19]:

Corollary 4.2 (Subgraph verification [19])**.** *There exists a graph G on which every GNN_n of width w requires depth at least*

$$d = \Omega\left(\sqrt{\frac{n}{w \log^2 n}} + \delta_G\right)$$

to verify if some subgraph $H = (\mathcal{V}_H, \mathcal{E}_H)$ of G:

- *is connected, i.e., there is a finite path between every pair of nodes in $\mathcal{V}_H$; or*
- *contains a cycle, i.e., H contains a subgraph with all nodes having degree equal to 2; or*
- *forms a spanning tree of G, i.e., H is a tree consisting of n nodes; or*
- *is bipartite. i.e., $\mathcal{V}$ can be partitioned into two sets such that no edge in $\mathcal{E}_H$ has both endpoints in the same set; or*
- *is a cut of G, i.e., deleting all edges $\mathcal{E}_H$ disconnects G; or*
- *is an s-t cut, such that removing all edges $\mathcal{E}_H$ from G will leave the nodes s and t disconnected.*

Furthermore, the depth should be at least

$$d = \Omega\left(\left(\frac{n}{w \log n}\right)^{\frac{1}{2} - \frac{1}{2(\delta_{G'} - 1)}} + \delta_G\right)$$

to verify if some subgraph H:

- *is a Hamiltonian cycle of G, i.e., a simple cycle of length n; or*
- *is a simple path, i.e., all nodes have degree 2 except from the two endpoint nodes whose degree is one.*

Therefore, even if one knows where to look in G, verifying whether a given subgraph meets a given property can be non-trivial, and this holds for several standard graph-theoretic properties. For instance, if we constrain ourselves to networks of constant width (something very desirable in terms of memory complexity), detecting whether a subgraph is connected can, up to logarithmic factors, require $\Omega(\sqrt{n})$ depth in the worst case.

4.3 Optimization problems

I turn my attention to the problems involving the exact or approximate optimization of some graph-theoretic objective function. From a machine learning perspective, the considered problems can be interpreted as node/edge classification problems: each node/edge is tasked with deciding whether it belongs to the optimal set or not. Take, for instance, the maximum independent set, where one needs to find the largest cardinality node set, such that no two of them are adjacent. Given only information identifying nodes, GNN_n will be asked to classify each node as being part of the maximum independent set.

Polynomial-time problems. Let me first consider three problems that possess known solutions. To make things easier for the GNN_n, I relax the objective and ask for an approximate solution rather than optimal. An algorithm (or neural network) is said to attain an α-approximation if it produces a feasible output whose utility is within a factor α of the optimal. Let OPT be the utility of the optimal solution and ALG that of the α-approximation algorithm. Depending on whether the problem entails minimization or maximization, the ratio ALG/OPT is at most α and at least $1/\alpha$, respectively.

According to the following corollary, it is non-trivial to find good approximate solutions:

Corollary 4.3 ([19], see also [29])**.** *There exists graphs G and G' of diameter $\delta_G = \Theta(\log n)$ and $\delta_{G'} = O(1)$ on which every GNN_n of width w requires depth at least*

$$d = \Omega\left(\sqrt{\frac{n}{w \log^2 n}}\right) \quad \text{and} \quad d' = \Omega\left(\left(\frac{n}{w \log n}\right)^{\frac{1}{2} - \frac{1}{2(\delta_{G'}-1)}}\right),$$

respectively, to approximate within any constant factor:

- *the minimum cut problem; or*
- *the shortest s-t path problem; or*
- *the minimum spanning tree problem.*

This is arguably disappointing: even for simple problems (complexity-wise) in the worst case a constant width GNN_n should be almost $\Omega(\sqrt{n})$ deep even if the graph diameter is exponentially smaller than n. It should be remarked that graphs with $\Theta(\log n)$ diameter are rather commonly encountered in network science and are generated by various standard random graph models [30], such as the Erdös–Rényi [31], the Barabasi-Albert [32], and the exponential cutoff [33] models, among others.

NP-hard problems. So what about truly hard problems? Clearly, one cannot expect a GNN to solve an NP-hard time in polynomial time[4]. However, it might be interesting as a thought experiment to consider a network whose layers take exponential time on the input size—e.g., by selecting the $\textsc{Msg}_l$ and $\textsc{Up}_l$ functions to be feed-forward networks of exponential width and or depth. Could one ever expect such a GNN_n to arrive at the optimal solution?

The following corollary provides necessary conditions for three well-known NP-hard problems:

Corollary 4.4 ([23])**.** *There exists a graph G on which every GNN_n of width $w = O(1)$ requires depth at least*

$$d = \Omega\left(\frac{n^2}{\log^2 n}\right)$$

to solve:

- *the minimum vertex cover problem; or*

[4]Unless P=NP.

- *the maximum independent set problem; or*
- *the chromatic coloring problem.*

According to Corollary 4.4, even if each layer of the network is allowed to take exponential time, the network depth should be much larger than the graph diameter $\delta_G = O(n)$ to have any chance of finding the optimal solution.

4.4 Estimation problems

Finally, I will consider problems that involve the exact or approximate estimation of some real function that takes as an input the graph and attributes. Estimation problems are related to graph embedding: the network should produce some graph-related quantity in its output. A key difference is that, whereas canonically graph embedding involves the use of a readout function, here the output is produced identically by every node.

The following corollary concerns the computation of two well-known graph invariants: the diameter δ_G and the girth. The latter is defined as the length of the shortest cycle and is infinity if the graph has no cycles.

Corollary 4.5 ([20])**.** *There exists a graph G on which every GNN_n of width w requires depth at least:*

- $d = \Omega\left(\frac{n}{w \log n} + \delta_G\right)$ *to exactly compute the graph diameter δ_G;*
- $d = \Omega\left(\frac{\sqrt{n}}{w \log n} + \delta_G\right)$ *to approximate the graph diameter and girth within a factor of $\frac{3}{2}$ and 2, respectively.*

Term δ_G appears in the lower bounds because both estimation problems require global information. Further, approximating the diameter within a 3/2 factor seems to be simpler than estimating it exactly. Yet, in both cases, cannot expect to achieve this using a GNN_n whose memory footprint remains linear on n. As a final remark, the graphs giving rise to the lower bounds of Corollary 4.5 have constant diameter and $\Theta(n^2)$ edges. However, similar bounds are known also for sparse graphs, i.e., graphs with $O(n \log n)$ edges [34]. For the case of exact computation, the lower bound is explained in Appendix B.

5 Conclusion

This work studied the capacity limits of graph neural networks without a readout function. Several impossibility results were presented for graph-theoretic decision, optimization, and estimation problems. It was discovered that the product of a GNN's depth and width plays a prominent role in determining network capacity. Strikingly, the condition $dw = \tilde{\Omega}(n)$ was found necessary for seemingly simple problems, such as odd cycle detection and diameter estimation. Overall, these results suggest that networks with constant memory footprint (i.e., with $dw = o(n)$) can be significantly limited in what they can learn.

Acknowledgements. I would like to thank the Swiss National Science Foundation (SNSF) for supporting this work in the context of the project "*Deep Learning for Graph-Structured Data*", grant number PZ00P2 179981.

A Graph theoretic problems

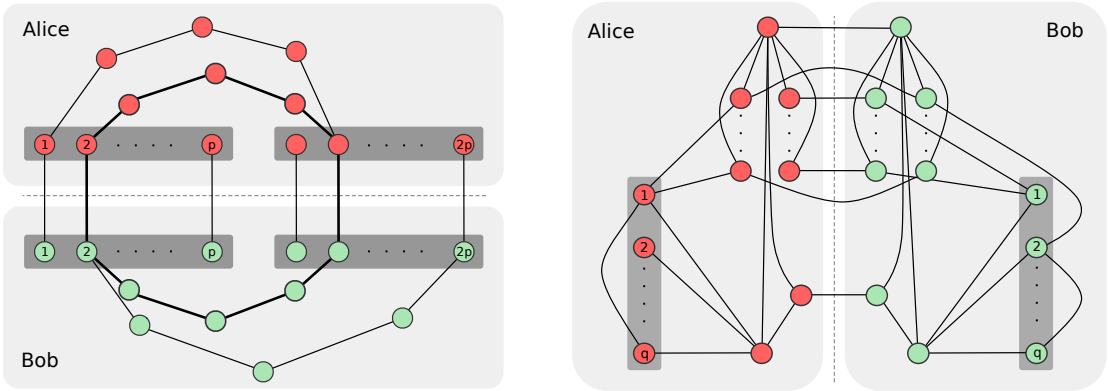
The main problems considered in this work are:

- *k-cycle detection*: a k-cycle is a subraph of G consisting of k nodes, each with degree two. The k-cycle detection problem entails determining if G contains a k-cycle.
- *(minimum) spanning tree*: a spanning tree is a tree subgraph of G consisting of n nodes. The minimum spanning tree problem entails finding the spanning tree of G of minimum weight (the weight of a tree is equal to the sum of its edge weights).
- *(minimum) cut*: a cut is a subgraph of G that when deleted leaves G disconnected. The minimum cut problem entails finding the cut of minimum weight (the weight of a cut is equal to the sum of its edge weights).
- *(shortest) path*: a path is subgraph of G where all nodes have degree 2 except from the two endpoint nodes whose degree is one. The shortest path problem entails finding the path of minimum weight that connects two given nodes (the weight of a path is equal to the sum of its edge weights).
- *(maximum) independent set*: an independent set is a set of nodes in a graph, no two of which are adjacent. The maximum independent set problem entails finding the set of maximum cardinality.
- *(minimum) vertex cover*: a vertex cover of G is a set of nodes such that each edge of G is incident to at least one node in the set. The minimum vertex cover problem entails finding the set of minimum cardinality.
- *(chromatic) coloring*: a coloring of G is a labeling of the nodes with distinct colors such that no two adjacent nodes are colored using same color. The chromatic coloring problem entails finding a coloring with the smallest number of colors.
- *diameter estimation*: the diameter δ_G of G equals the length of the longest shortest path. The diameter estimation problem entails computing δ_G of G.
- *girth estimation*: the girth of G equals the length of the shortest cycle. It is infinity if no cycles are present. The girth estimation problem entails computing the girth of G.

B Lower bound techniques

A common technique for obtaining lower bounds in the LOCAL model is by reduction to the *set-disjointness* problem in two-player communication complexity: Suppose that Alice and Bob are each given some secret string (s_a and s_b) of q bits. The two players use the string to construct a set by selecting the elements from the base set $\{1, 2, \ldots, q\}$ for which the corresponding bit is one. It is known that Alice and Bob cannot determine whether their sets are disjoint or not without exchanging at least $\Omega(q)$ bits.

The reduction involves constructing a graph that is partially known by each player. Usually, Alice and Bob start knowing half of the graph (red and green induced subgraphs in Figure 1). The players then use their secret string to control some aspect of their private topology (subgraphs annotated in dark gray). Let the resulting graph be $G(s_a, s_b)$ and denote by cut the number of edges connecting the subgraphs controlled by Alice and Bob. To derive a lower bound for some problem P, one needs to prove that a solution for P in $G(s_a, s_b)$ would also reveal whether the two sets are disjoint or not. Since each player can exchange at most $O(b \cdot \text{cut})$ bits per round, at least $\Omega(q/(b \cdot \text{cut}))$ rounds are needed



(a) 10-cycle lower bound graph

(b) diameter lower bound graph

Figure 1: Examples of graphs giving rise to lower bounds.

in total in LOCAL. By Lemma 4.1, one then obtains a $d = \Omega(q/(w \log n \cdot \text{cut}))$ depth lower bound for GNN_n.

The two examples in Figure 1 illustrate the graphs $G(s_a, s_b)$ giving rise to the lower bounds for even cycle detection and diameter estimation. To reduce occlusion, only a subset of the edges are shown.

(a) In the construction of Korhonen and Rybicki [22], each player starts from a complete bipartite graph of $p = \sqrt{q}$ nodes (nodes annotated in dark grey) with nodes numbered from 1 to $2p$. The nodes with the same id are connected yielding a cut of size $2p$. Each player then uses its secret (there are as many bits as bipartite edges) to decide which of the bipartite edges will be deleted and which will remain (not shown here). Deleted edges are substituted by a path of length $k/2 - 1$. This happens in a way that ensures that $G(s_a, s_b)$ contains a cycle of length k (half known by Alice and half by Bob) if and only if the two sets are disjoint: the cycle will pass through nodes t and $p + t$ of each player to signify that the t-th bits of s_a and s_b coincide. It can then be shown that $n = \Theta(p^2)$ from which it follows that: LOCAL requires at least $d = \Omega(q/(b \cdot \text{cut})) = \Omega(n/(b \cdot p)) = \Omega(\sqrt{n}/b)$ bits to decide if there is a cycle of length k; and GNN_n has to have $d = \Omega(\sqrt{n}/(w \log n))$ depth to do the same.

(b) In the construction of Abboud et al. [34], each string consists of $q = \Omega(n)$ bits. The strings are used to encode the connectivity of subgraphs annotated in dark gray: an edge exists between the red nodes i and q if and only if the i-th bit of s_a is one (and similarly for green). Due to the graph construction, the cut between Alice and Bob has $O(\log q)$ edges. Moreover, $G(s_a, s_b)$ has diameter at least 5 if and only if the sets defined by s_a and s_b are disjoint. This implies that $d = \Omega(n/(w \log^2 n))$ depth is necessary to compute the graph diameter in GNN_n.

References

[1] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of control, signals and systems*, vol. 2, no. 4, pp. 303–314, 1989.

[2] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural networks*, vol. 2, no. 5, pp. 359–366, 1989.

[3] Z. Lu, H. Pu, F. Wang, Z. Hu, and L. Wang, "The expressive power of neural networks: A view from the width," in *Advances in Neural Information Processing Systems 30* (I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, eds.), pp. 6231–6239, Curran Associates, Inc., 2017.

[4] H. Maron, E. Fetaya, N. Segol, and Y. Lipman, "On the universality of invariant networks," *arXiv preprint arXiv:1901.09342*, 2019.

[5] N. Keriven and G. Peyré, "Universal invariant and equivariant graph neural networks," *arXiv preprint arXiv:1905.04943*, 2019.

[6] M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Poczos, R. R. Salakhutdinov, and A. J. Smola, "Deep sets," in *Advances in Neural Information Processing Systems 30* (I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, eds.), pp. 3391–3401, Curran Associates, Inc., 2017.

[7] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?," *arXiv preprint arXiv:1810.00826*, 2018.

[8] Y. Seo, A. Loukas, and N. Peraudin, "Discriminative structural graph classification," *arXiv preprint arXiv:1905.13422*, 2019.

[9] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 1263–1272, JMLR. org, 2017.

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

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

[12] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel, "Gated graph sequence neural networks," *arXiv preprint arXiv:1511.05493*, 2015.

[13] D. K. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams, "Convolutional networks on graphs for learning molecular fingerprints," in *Advances in neural information processing systems*, pp. 2224–2232, 2015.

[14] P. Battaglia, R. Pascanu, M. Lai, D. J. Rezende, *et al.*, "Interaction networks for learning about objects, relations and physics," in *Advances in neural information processing systems*, pp. 4502–4510, 2016.

[15] S. Kearnes, K. McCloskey, M. Berndl, V. Pande, and P. Riley, "Molecular graph convolutions: moving beyond fingerprints," *Journal of computer-aided molecular design*, vol. 30, no. 8, pp. 595–608, 2016.

[16] D. Angluin, "Local and global properties in networks of processors (extended abstract)," in *Proceedings of the Twelfth Annual ACM Symposium on Theory of Computing*, STOC '80, (New York, NY, USA), pp. 82–93, ACM, 1980.

[17] N. Linial, "Locality in distributed graph algorithms," *SIAM Journal on Computing*, vol. 21, no. 1, pp. 193–201, 1992.

[18] M. Naor and L. J. Stockmeyer, "What can be computed locally?," in *STOC*, 1993.

[19] A. D. Sarma, S. Holzer, L. Kor, A. Korman, D. Nanongkai, G. Pandurangan, D. Peleg, and R. Wattenhofer, "Distributed verification and hardness of distributed approximation," *SIAM Journal on Computing*, vol. 41, no. 5, pp. 1235–1265, 2012.

[20] S. Frischknecht, S. Holzer, and R. Wattenhofer, "Networks cannot compute their diameter in sublinear time," in *Proceedings of the twenty-third annual ACM-SIAM symposium on Discrete Algorithms*, pp. 1150–1162, Society for Industrial and Applied Mathematics, 2012.

[21] A. Drucker, F. Kuhn, and R. Oshman, "On the power of the congested clique model," in *Proceedings of the 2014 ACM symposium on Principles of distributed computing*, pp. 367–376, ACM, 2014.

[22] J. H. Korhonen and J. Rybicki, "Deterministic subgraph detection in broadcast congest," *arXiv preprint arXiv:1705.10195*, 2017.

[23] K. Censor-Hillel, S. Khoury, and A. Paz, "Quadratic and near-quadratic lower bounds for the congest model," *arXiv preprint arXiv:1705.05646*, 2017.

[24] C. K. Joshi, T. Laurent, and X. Bresson, "An efficient graph convolutional network technique for the travelling salesman problem," *arXiv preprint arXiv:1906.01227*, 2019.

[25] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, *et al.*, "Relational inductive biases, deep learning, and graph networks," *arXiv preprint arXiv:1806.01261*, 2018.

[26] J. Suomela, "Survey of local algorithms," *ACM Computing Surveys (CSUR)*, vol. 45, no. 2, p. 24, 2013.

[27] L. Feuilloley and P. Fraigniaud, "Survey of distributed decision," *arXiv preprint arXiv:1606.04434*, 2016.

[28] G. Even, O. Fischer, P. Fraigniaud, T. Gonen, R. Levi, M. Medina, P. Montealegre, D. Olivetti, R. Oshman, I. Rapaport, *et al.*, "Three notes on distributed property testing," in *31st International Symposium on Distributed Computing (DISC 2017)*, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

[29] M. Ghaffari and F. Kuhn, "Distributed minimum cut approximation," in *International Symposium on Distributed Computing*, pp. 1–15, Springer, 2013.

[30] D. Chakrabarti and C. Faloutsos, "Graph mining: Laws, generators, and algorithms," *ACM Comput. Surv.*, vol. 38, June 2006.

[31] P. Erdös and A. Rényi, "On the evolution of random graphs," *Publ. Math. Inst. Hung. Acad. Sci*, vol. 5, no. 1, pp. 17–60, 1960.

[32] A.-L. Barabási and R. Albert, "Emergence of scaling in random networks," *science*, vol. 286, no. 5439, pp. 509–512, 1999.

[33] M. E. Newman, "The structure of scientific collaboration networks," *Proceedings of the national academy of sciences*, vol. 98, no. 2, pp. 404–409, 2001.

[34] A. Abboud, K. Censor-Hillel, and S. Khoury, "Near-linear lower bounds for distributed distance computations, even in sparse networks," in *International Symposium on Distributed Computing*, pp. 29–42, Springer, 2016.