WHAT GRAPH NEURAL NETWORKS CANNOT LEARN: DEPTH VS WIDTH

Anonymous authors
Paper under double-blind review

ABSTRACT

This paper studies theoretically the capacity limits of graph neural networks (GNN) falling within the message-passing framework. 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. Second, 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 repurposing of seminal results from theoretical computer science and 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 (a function of) 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 has been known for some time that feed-forward neural networks of sufficient depth and width are universal function approximators (Cybenko, 1989; Hornik et al., 1989; Lu et al., 2017). 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. (2019b) 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 & Peyré (2019). Expanding upon deep sets (Zaheer et al., 2017), Xu et al. (2018) 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. (2019).

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 might be easier to obtain insights about models by studying their limitations. After all, the knowledge of what *cannot* be computed (and thus 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 then we can be certain that the same network will not learn how to classify a sufficiently diverse test set correctly, independently of which learning algorithm is employed. We should, therefore, focus on networks deeper that the lower bound when performing experiments.

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)	$dw = \Omega(\sqrt{n}/\log n)$	max. indep. set	$dw = \Omega(n^2/\log^2 n)$ for $w = O(1)$
subgraph verification*	$d\sqrt{w} = \Omega(\sqrt{n}/\log n)$	min. vertex cover	$dw = \Omega(n^2/\log^2 n)$ for $w = O(1)$
min. spanning tree	$d\sqrt{w} = \Omega(\sqrt{n}/\log n)$	perfect coloring	$dw = \Omega(n^2/\log^2 n)$ for $w = O(1)$
min. cut	$d\sqrt{w} = \Omega(\sqrt{n}/\log n)$	girth 2-approx.	$dw = \Omega(\sqrt{n}/\log n)$
diam. estimation	$dw = \Omega(n/\log n)$	diam. $3/2$-approx.	$dw = \Omega(\sqrt{n}/\log n)$

Table 1: Summary of main results. Subgraph 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.

1.1 Main results

This paper studies theoretically the computational capacity limits of graph neural networks (GNN) falling within the message-passing framework of Gilmer et al. (2017). This model is chosen as it is sufficiently general to encompass several state-of-the-art networks, including GCN (Kipf & Welling, 2016), ChebyNet (Defferrard et al., 2016), gated graph neural networks (Li et al., 2015), molecular fingerprints (Duvenaud et al., 2015), interaction networks (Battaglia et al., 2016), molecular graph convolutions (Kearnes et al., 2016), among many others. The provided contributions are two-fold:

I. What GNN can compute. 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 compliments recent universality results (Maron et al., 2019b; Keriven & Peyré, 2019) 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 equivalence of GNN with LOCAL (Angluin, 1980; Linial, 1992; Naor & Stockmeyer, 1993), 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: there are enough layers of sufficient expressiveness and width, and nodes can uniquely distinguish each other.

II. What GNN cannot compute (and thus learn). Section 4 analyses the implications of restricting the depth d and width w of GNN that do not use a readout function. Specifically, it is proven that GNN lose a significant portion of their power when the product dw is restricted. The analysis relies on a new technique that enables repurposing impossibility results from the context of distributed computing to the graph neural network setting. Specifically, lower bounds for the following problems are presented: (i) detecting whether an input graph G contains a cycle of specific length; (ii) 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; (iii) approximating the shortest path between two nodes, the minimum cut, and the minimum spanning tree; (iv) finding a maximum independent set, a minimum vertex cover, or a perfect coloring of G; (v) 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 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 attempts to use GNN to devise heuristics graph-based optimization problems (Joshi et al., 2019; Battaglia et al., 2018), such as the ones discussed above. The presented results can then be taken as a worst-case analysis for such algorithms.

1.2 Discussion

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 its 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, detecting 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 certain NP-hard problems necessitates $d = \tilde{\Omega}(n^2)$ depth for any constant-width network.

Relation to previous impossibility results. In contrast to universality (Maron et al., 2019b; Keriven & Peyré, 2019), the limitations of GNN have been much less studied. In particular, the bounds presented here are the first impossibility results that (i) explicitly connect GNN properties (depth and width) with graph properties and that (ii) go beyond isomorphism by addressing decision, optimization, and estimation graph problems. Two main directions of related work can be distinguished. First, Dehmamy et al. (2019) bounded the capacity of graph convolutional networks (i.e., GNN w/o messaging functions) to compute specific polynomial functions of the adjacency matrix, referred to as graph moments by the authors. Second, Xu et al. (2018) and Morris et al. (2019) independently established the equivalence of *anonymous* GNN (those that do not rely on node identification) to the Weisfeiler-Lehman (WL) graph isomorphism test. The equivalence implies that anonymous networks are blind to the many graph properties that WL cannot see: e.g., any two regular graphs with the same number of nodes and edges are identical from the perspective of the WL test (Arvind et al., 2015; Kiefer et al., 2015). Such negative results occur due to nodes being unable to distinguish between neighbors at multiple hops (see Appendix D). When nodes have the capacity to distinguish each other GNN become significantly more powerful (without necessarily sacrificing permutation in/equi-variance[1]). Still, as this work shows, even with ids certain problems remain impossible when the depth and width of the GNN is restricted. For instance, though the impossibility of cycle detection is eventually annulled with identifiers (detecting cycles is impossible in anonymous networks), to find even cycles one must now have $dw = \tilde{\Omega}(n)$.

Limitations. Perhaps the most important limitation is that all lower bounds are of a worst-case nature: 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 consider a reduced set of graphs. Second, rather than taking into account the specific parametric functions used by each layer, each layer is assumed to be sufficiently powerful to compute any function of its input. Fortunately, this strong assumption does not significantly limit the applicability of the results, simply because all lower bounds that hold with layers of unbounded capacity also apply to those limited computationally. Lastly, in the following, it is assumed that nodes can uniquely identify each other. The use of ids should not be interpreted as a dependence on the permutation of nodes: node identification is compatible with permutation invariance/equivariance as long as the network output is asked to be invariant to the particular way the ids have been assigned. In the literature, one-hot encoded node ids have found use on the transductive setting (Kipf & Welling, 2016). When attempting to learn functions across multiple graphs, ids should be ideally substituted by sufficiently discriminative node attributes (attributes that uniquely identify each node within each receptive field it belongs to can serve as ids). Nevertheless, 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 and discriminative attributes. Thus, the presented bounds also apply to (partially) 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 $e_{i \leftarrow j} \in \mathcal{E}$ and the outgoing neighborhood is defined as $\mathcal{O}_i = \{v_j \in \mathcal{V} : e_{j \leftarrow i} \in \mathcal{E}\}$. The degree of v_i is denoted by deg_i, Δ is the maximum degree of all nodes 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 $\mathcal{O}_i^* = \mathcal{O}_i \cup v_i$.

[1]A GNN whose output is in/equi-variant to how the ids were assigned is permutation in/equi-variant.

2 THE GRAPH NEURAL NETWORK COMPUTATIONAL MODEL

Graph neural networks are parametric and differentiable algorithms. 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 $e_{i \leftarrow j} \in \mathcal{E}$.

Model 1 formalizes the graph neural network operation by placing it in a message passing computational model (Gilmer et al., 2017). As seen, the computation proceeds in layers, within which 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 and each node updates its internal representation 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$_g$).
```

It can be observed that the operation of a GNN is primarily determined by the *messaging*, *update*, and *readout* functions. I assume that $\textsc{Msg}_\ell$ and $\textsc{Up}_\ell$ are general functions that act on intermediate node representations and node ids (the notation is overloaded such that v_i refers to both the i-th node as well as its unique id). As is common in the literature (Lu et al., 2017; Battaglia et al., 2018), these functions are instantiated by feed-forward neural networks. Thus, by the universal approximation theorem and its variants (Cybenko, 1989; Hornik et al., 1989), they can approximate any general function that maps vectors onto vectors, given sufficient depth and/or width. Function $\textsc{Read}$ is useful when one needs 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. Commonly, $\textsc{Read}$ is chosen to be a dimension squashing operator, such as a sum or a mean, followed by a feed-forward neural 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 $v_i \in \mathcal{V}$. Since nodes need to be able to store their own unique ids, in the following it is assumed that each variable manipulated by the network is represented in finite-precision using $p = \Theta(\log n)$ bits (though this is not strictly necessary for the analysis).

3 SUFFICIENT CONDITIONS FOR TURING UNIVERSALITY

This section studies what graph neural networks can compute. It is demonstrated that, even without readout function, a network is computationally universal[2] if 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 entails establishing that GNN_n is equivalent to LOCAL, a classical model used in the study the distributed algorithms that is itself Turing universal.

[2]It can compute anything that a Turing machine can compute when given an attributed graph as input.

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 (1980), Linial (1992), and Naor & Stockmeyer (1993), provides a common framework for analyzing the effect of local decision. Akin to GNN, 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. 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. All nodes execute the same algorithm, they are fault-free, and they are provided with unique identities.

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. The computation starts simultaneously and unfolds in synchronous rounds $l = 1, \ldots, d$. Three things can occur within each round: each node receives a string from its incoming neighbors; each node updates its internal state by performing some local computation; and 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. Functions ALG_l^1 and 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 $\{a_{i \leftarrow j} : v_j \in \mathcal{I}_i^*\}$.

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

```
**Initialization:** Set $s_{i \leftarrow i}^{(0)} = (a_i, v_i)$ and $s_{i \leftarrow j}^{(0)} = (a_j, v_j)$ for all $e_{i \leftarrow j} \in \mathcal{E}$.
for round $\ell = 1, \ldots, d$ do
    Receive $s_{i \leftarrow j}^{(\ell-1)}$ from $v_j \in \mathcal{I}_i^*$, compute
        $s_i^{(\ell)} = \text{ALG}_\ell^1 \left( \left\{ \left( s_{i \leftarrow j}^{(\ell-1)}, a_{i \leftarrow j} \right) : v_j \in \mathcal{I}_i^* \right\}, v_i \right)$,
    and send $s_{j \leftarrow i}^{(\ell)} = \text{ALG}_\ell^2 \left( s_i^{(\ell)}, v_i \right)$ to $v_j \in \mathcal{O}_i^*$.
return $s_i^{(d)}$
```

3.2 Turing universality

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).

Yet, as the following theorem shows, the differences between GNN_n and LOCAL are inconsequential when seen from the perspective of computational capacity:

Theorem 3.1. *The LOCAL and GNN_n computational models are equivalent when functions MSG_l and UP_l are Turing complete for every layer ℓ.*

This equivalence enables us to reason about the power of graph neural networks by building on the well-studied properties of LOCAL. In particular, it is well known in distributed computing that, as long as the number of rounds d of a distributed algorithm is larger than the graph diameter δ_G, every node in a LOCAL can effectively make decisions based on the entire graph and attributes. Together with Theorem 3.1, the above imply that, if computation and memory is not an issue, one may construct a GNN_n that effectively computes *any* computable function w.r.t. the graph and attributes.

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; MSG_l and UP_l are Turing-complete for every layer ℓ; the network consists of $d \geq \delta_G$ layers; and the width is unbounded.*

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 does not have 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 a non universal learner cannot learn. *Second, a universality result provides 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 two 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.

The universality of GNN$_g$. Though a universality result could also be easily derived for networks with a readout function, the latter is not included as it deviates from how graph neural networks are meant to function: 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.

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. In addition, it will be assumed that nodes do not have access to a random generator (so that the output is deterministic).

4.1 How to repurpose lower bounds

Since LOCAL is mainly used to study the impact of locality, by default no restrictions are placed on the message size. However, to be able to examine the role of width, it will be useful to also assume that the message size in LOCAL at each round is at most b bits. This model goes by the name CONGEST in the distributed computing literature (Peleg, 2000).

The following theorem shows us how to translate impossibility results from CONGEST to GNN$_n$:

Theorem 4.1. *If a problem P cannot be solved in less than d rounds in CONGEST 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.*

The $p = \Theta(\log n)$ factor corresponds to the length of the binary representation of every variable—the precision needs to depend logarithmically on n for the node ids to be unique.

With this result in place, the following sections re-state several known lower bounds in terms of a GNN$_n$'s depth and width.

4.2 Impossibility results for decision problems

I first consider problems where one needs to decide whether a given graph satisfies a certain property (Feuilloley & Fraigniaud, 2016). Concretely, given a decision problem P and a graph G, the GNN$_n$ should output

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

The network then accepts the premise if the logical conjunction of $\{x_1, \ldots, x_n\}$ is *true* and rejects it otherwise. Such problems are intimately connected to graph classification: classifying a graph entails identifying what constitutes a class from some training set *and* using said learned definition to decide the label of 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, even with ids, cycle detection remains a relatively hard problem:

Corollary 4.1 (Repurposed from (Drucker et al., 2014; Korhonen & Rybicki, 2018))**.** *There exists graph G on which every GNN_n of width w requires depth at least $d = \Omega(\sqrt{n}/(w \log n))$ and $d = \Omega(n/(w \log n))$ to detect if G contains a cycle C_k for even $k \geq 4$ and odd $k \geq 5$, respectively.*

Whereas an anonymous GNN cannot detect cycles (e.g., distinguish between two C_3 vs one C_6 (Maron et al., 2019a)), it seems that with ids the product of depth and width should exhibit an at least linear dependence on n. The intuition behind this bound can be found in Appendix C.

Subgraph verification. Suppose that the network 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 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 is a direct consequence of the seminal work by Sarma et al. (2012):

Corollary 4.2 (Repurposed from (Sarma et al., 2012))**.** *There exists a graph G on which every GNN_n of width w requires depth at least $d = \Omega(\sqrt{\frac{n}{w \log^2 n}} + \delta_G)$ to verify if some subgraph H of G is connected, contains a cycle, forms a spanning tree of G, is bipartite, is a cut of G, or is an s-t cut of G. Furthermore, the depth should be at least $d = \Omega\left(\left(\frac{n}{w \log n}\right)^{\gamma} + \delta_G\right)$ with $\gamma = \frac{1}{2} - \frac{1}{2(\delta_{G'} - 1)}$ to verify if H is a Hamiltonian cycle or a simple path.*

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 Impossibility results for 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 (Repurposed from (Sarma et al., 2012; Ghaffari & Kuhn, 2013))**.** *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 $d = \Omega(\sqrt{\frac{n}{w \log^2 n}})$ and $d' = \Omega((\frac{n}{w \log n})^{\gamma})$ with $\gamma = \frac{1}{2} - \frac{1}{2(\delta_{G'} - 1)}$, respectively, to* *approximate within any constant factor: the minimum cut problem, the shortest s-t path problem, or the minimum spanning tree problem.*

Thus, 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.

NP-hard problems. So what about truly hard problems? Clearly, one cannot expect a GNN to solve an NP-hard time in polynomial time[3]. 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 depth and width. 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 (Repurposed from (Censor-Hillel et al., 2017))**.** *There exists a graph G on which every GNN_n of width $w = O(1)$ requires depth at least $d = \Omega(n^2/\log^2 n)$ to solve: the minimum vertex cover problem; the maximum independent set problem; the perfect 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 Impossibility results for 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 (Repurposed from (Frischknecht et al., 2012))**.** *There exists a graph G on which every GNN_n of width w requires depth at least $d = \Omega(n/(w \log n) + \delta_G)$ to exactly compute the graph diameter δ_G and $d = \Omega(\sqrt{n}/(w \log n) + \delta_G)$ to approximate the graph diameter and girth within a factor of $3/2$ 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 computing it. Yet, in both cases, one cannot achieve this using a GNN_n whose capacity dw is constant. 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 can be derived also for graphs with $O(n \log n)$ edges (Abboud et al., 2016). For the case of exact computation, the lower bound is explained in Appendix C.

5 Conclusion

This work studied theoretically 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 whose size is independent of n can be significantly limited in what they can learn.

[3]Unless P=NP.

REFERENCES

Amir Abboud, Keren Censor-Hillel, and Seri Khoury. Near-linear lower bounds for distributed distance computations, even in sparse networks. In *International Symposium on Distributed Computing*, pp. 29–42. Springer, 2016.

Dana 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, pp. 82–93, New York, NY, USA, 1980. ACM. ISBN 0-89791-017-6. doi: 10.1145/800141.804655. URL `http://doi.acm.org/10.1145/800141.804655`.

Vikraman Arvind, Johannes Köbler, Gaurav Rattan, and Oleg Verbitsky. On the power of color refinement. In *International Symposium on Fundamentals of Computation Theory*, pp. 339–350. Springer, 2015.

Peter Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, et al. Interaction networks for learning about objects, relations and physics. In *Advances in neural information processing systems*, pp. 4502–4510, 2016.

Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

Keren Censor-Hillel, Seri Khoury, and Ami Paz. Quadratic and near-quadratic lower bounds for the congest model. *arXiv preprint arXiv:1705.05646*, 2017.

Benny Chor and Oded Goldreich. Unbiased bits from sources of weak randomness and probabilistic communication complexity. *SIAM Journal on Computing*, 17(2):230–261, 1988.

George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in neural information processing systems*, pp. 3844–3852, 2016.

Nima Dehmamy, Albert-László Barabási, and Rose Yu. Understanding the representation power of graph neural networks in learning graph topology. *arXiv preprint arXiv:1907.05008*, 2019.

Andrew Drucker, Fabian Kuhn, and Rotem 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.

David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P Adams. Convolutional networks on graphs for learning molecular fingerprints. In *Advances in neural information processing systems*, pp. 2224–2232, 2015.

Laurent Feuilloley and Pierre Fraigniaud. Survey of distributed decision. *arXiv preprint arXiv:1606.04434*, 2016.

Silvio Frischknecht, Stephan Holzer, and Roger 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.

Mohsen Ghaffari and Fabian Kuhn. Distributed minimum cut approximation. In *International Symposium on Distributed Computing*, pp. 1–15. Springer, 2013.

Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George 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.

Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

Chaitanya K Joshi, Thomas Laurent, and Xavier Bresson. An efficient graph convolutional network technique for the travelling salesman problem. *arXiv preprint arXiv:1906.01227*, 2019.

Bala Kalyanasundaram and Georg Schintger. The probabilistic communication complexity of set intersection. *SIAM Journal on Discrete Mathematics*, 5(4):545–557, 1992.

Steven Kearnes, Kevin McCloskey, Marc Berndl, Vijay Pande, and Patrick Riley. Molecular graph convolutions: moving beyond fingerprints. *Journal of computer-aided molecular design*, 30(8): 595–608, 2016.

Nicolas Keriven and Gabriel Peyré. Universal invariant and equivariant graph neural networks. *arXiv preprint arXiv:1905.04943*, 2019.

Sandra Kiefer, Pascal Schweitzer, and Erkal Selman. Graphs identified by logics with counting. In *International Symposium on Mathematical Foundations of Computer Science*, pp. 319–330. Springer, 2015.

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

Janne H Korhonen and Joel Rybicki. Deterministic subgraph detection in broadcast congest. In *21st International Conference on Principles of Distributed Systems (OPODIS 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493*, 2015.

Nathan Linial. Locality in distributed graph algorithms. *SIAM Journal on Computing*, 21(1):193–201, 1992.

Zhou Lu, Hongming Pu, Feicheng Wang, Zhiqiang Hu, and Liwei Wang. The expressive power of neural networks: A view from the width. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 30*, pp. 6231–6239. Curran Associates, Inc., 2017. URL `http://papers.nips.cc/paper/7203-the-expressive-power-of-neural-networks-a-view-from-the-width.pdf`.

Haggai Maron, Heli Ben-Hamu, Hadar Serviansky, and Yaron Lipman. Provably powerful graph networks. *arXiv preprint arXiv:1905.11136*, 2019a.

Haggai Maron, Ethan Fetaya, Nimrod Segol, and Yaron Lipman. On the universality of invariant networks. *arXiv preprint arXiv:1901.09342*, 2019b.

Christopher Morris, Martin Ritzert, Matthias Fey, William L Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 4602–4609, 2019.

Moni Naor and Larry J. Stockmeyer. What can be computed locally? In *STOC*, 1993.

D. Peleg. *Distributed Computing: A Locality-Sensitive Approach*. Society for Industrial and Applied Mathematics, 2000. doi: 10.1137/1.9780898719772.

Atish Das Sarma, Stephan Holzer, Liah Kor, Amos Korman, Danupon Nanongkai, Gopal Pandurangan, David Peleg, and Roger Wattenhofer. Distributed verification and hardness of distributed approximation. *SIAM Journal on Computing*, 41(5):1235–1265, 2012.

Younjoo Seo, Andreas Loukas, and Nathanael Peraudin. Discriminative structural graph classification. *arXiv preprint arXiv:1905.13422*, 2019.

Jukka Suomela. Survey of local algorithms. *ACM Computing Surveys (CSUR)*, 45(2):24, 2013.

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

Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Ruslan R Salakhutdinov, and Alexander J Smola. Deep sets. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 30*, pp. 3391–3401. Curran Associates, Inc., 2017. URL `http://papers.nips.cc/paper/6931-deep-sets.pdf`.

A Graph theory definitions

The main graph-theoretic terms encountered 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.
- *Hamiltonian cycle*: a cycle of length n
- *(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).
- *s-t cut*: a subgraph of G such that removing all subgraph edges from G will leave the nodes s and t of G disconnected.
- *(shortest) path*: a simple 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 simple 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.
- *(perfect) 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 perfect 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 Deferred Proofs

B.1 Proof of Theorem 3.1

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

$$\begin{aligned}
x_i^{(l)} &= \textsc{Up}_\ell\Big(\sum_{v_j \in \mathcal{I}_i^*} m_{i\leftarrow j}^{(\ell)} \Big) && \text{(by definition)} \\
&= \textsc{Up}_\ell\Big(\sum_{v_j \in \mathcal{I}_i^*} \textsc{Msg}_\ell \left(x_i^{(\ell-1)}, x_j^{(\ell-1)}, v_i,\, v_j,\, a_{i\leftarrow j} \right) \Big) && \text{(substituted } m_{i\leftarrow j}^{(\ell)}) \\
&= \textsc{Agg}_\ell \left(\left\{ \left(x_i^{(\ell-1)}, x_j^{(\ell-1)}, v_i,\, v_j,\, a_{i\leftarrow j} \right) : v_j \in \mathcal{I}_i^* \right\} \right), && \text{(from (Xu et al., 2018, Lemma 5))}
\end{aligned}$$

where AGG_ℓ 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. Xu et al. (2018) 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:

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

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

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

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

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, 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 $\text{ALG} = \text{AGG} \circ g$, 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 $\text{AGG} = \text{ALG} \circ h$, where h deletes x_i from each element of the multiset.

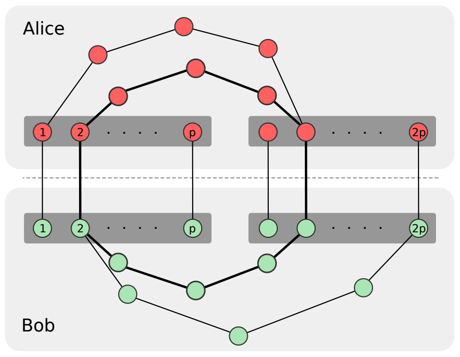
B.2 Proof of Corollary 3.1

In the LOCAL model the reasoning is simple: suppose that the graph is represented by a set of edges and further consider that 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 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 equivalence of LOCAL and GNN_n.

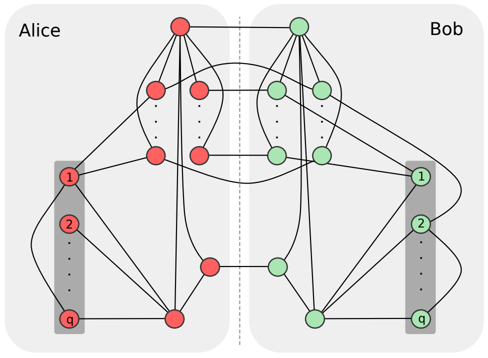
B.3 Proof of Theorem 4.1

First note that, since the GNN_n and LOCAL models are 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 Theorem 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 (in terms of bits) also transfer between the models.

To proceed, I demonstrate that a depth lower bound in the CONGEST model (i.e., 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 $A \in$ 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 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 ALG_{l+1}^1 to simulate



(a) 10-cycle lower bound graph



(b) diameter lower bound graph

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

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 ids). 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, and the width of GNN_n has to be at least $(b - \log_2 n)/p$.

C An illustration of the lower bounds for cycle detection and diameter estimation

A common technique for obtaining lower bounds in the CONGEST 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 (Kalyanasundaram & Schintger, 1992; Chor & Goldreich, 1988).

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 in total in CONGEST. By Theorem 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.

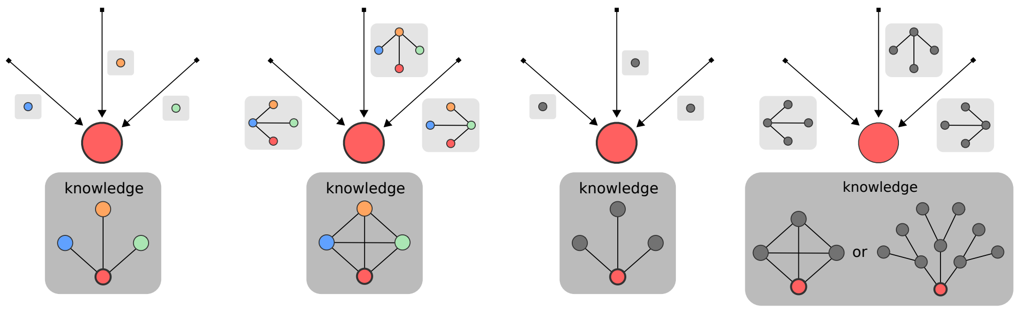
(a) In the construction of Korhonen & Rybicki (2018), each player starts from a complete bipartite graph of $p = \sqrt{q}$ nodes (nodes annotated in dark grey) with nodes numbered from 1 to $2p$. The nodes with the same id are connected yielding a cut of size $2p$. Each player then uses its secret (there are as many bits as bipartite edges) to decide which of the bipartite edges will be deleted (corresponding to zero bits). Remaining 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 are both one. It can then be shown that $n = \Theta(p^2)$ from which it follows that: CONGEST 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. (2016), 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, About et al. proved that $G(s_a, s_b)$ has diameter at least five 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.

D THE COST OF ANONYMITY

There is a striking difference between the power of anonymous networks and those in which nodes have the capacity to uniquely identify each other (e.g., based on ids or discriminative node attributes) (Suomela, 2013).

To illustrate this phenomenon, I consider a toy example where a node is tasked with reconstructing the graph topology in the LOCAL model. In the left, Figure 2 depicts the red node's knowledge after two rounds (equivalent to a GNN_n having $d = 2$) when each node has a unique identifier (color). At the end of the first round, each node is aware of its neighbors and after two rounds the entire graph has been successfully reconstructed in red's memory.



(a) nodes *can* identify each other (this work) (b) nodes *cannot* identify each other (WL test).

Figure 2: *Toy example of message exchange from the perspective of the red node. The arrows show where each received message comes from and the message content is shown in light gray boxes. Red's knowledge of the graph topology is depicted at the bottom.*

In the right subfigure, nodes do not possess ids (as in the analysis of (Xu et al., 2018; Morris et al., 2019)) and thus cannot distinguish which of their neighbors are themselves adjacent. As such, the red node cannot tell whether the graph contains cycles: after two rounds there are at least two plausible topologies that could explain its observations.