We Cannot Guarantee Safety: The Undecidability of Graph Neural Network Verification

Marco Sälzer  
University of Kassel  
Germany  
marco.saelzer@uni-kassel.de

Martin Lange  
University of Kassel  
Germany  
martin.lange@uni-kassel.de

Abstract

Graph Neural Networks (GNN) are commonly used for two tasks: (whole) graph classification and node classification. We formally introduce generically formulated decision problems for both tasks, corresponding to the following pattern: given a GNN, some specification of valid inputs, and some specification of valid outputs, decide whether there is a valid input satisfying the output specification. We then prove that graph classifier verification is undecidable in general, implying that there cannot be an algorithm surely guaranteeing the absence of misclassification of any kind. Additionally, we show that verification in the node classification case becomes decidable as soon as we restrict the degree of the considered graphs. Furthermore, we discuss possible changes to these results depending on the considered GNN model and specifications.

1 Introduction

The Graph Neural Network (GNN) framework, i.e. models that compute functions over graphs, has become the goto technique for learning tasks over structured data in recent years. This is not surprising since GNN application possibilities are enormous, ranging from physics [1] and chemistry/biology [2] over recommender systems [3] to general knowledge graph applications which itself includes a broad range of applications [4]. Additionally, the performance of GNN in most applications is without a doubt outstanding. Naturally, the high interest in GNN and their broad range of applications is without a doubt outstanding. Naturally, the high interest in GNN and their broad range of applications including safety-critical ones, for instance in traffic situations, impose two necessities: first, a solid foundational theory of GNN is needed that describes possibilities and limits of GNNs’ functionalities. Second, methods for assessing the safety of GNN are needed, in the best case guaranteeing that unwanted behaviour cannot occur.

Compared to the amount of work on performance improvement for GNN or the development of new model variants, the amount of work studying basic theoretical results about GNN is rather limited. Some general results have been obtained as follows: independently, Xu et al. [5] and Morris et al. [6] showed that GNN belonging to the model of Message Passing Neural Networks [7] are non-universal in the sense that they cannot be trained to distinguish specific graph structures. Furthermore, both relate the expressiveness of GNN to the Weisfeiler-Leman graph isomorphism test. This characterization is thoroughly described and extended by Grohe in [8]. Loukas [9] showed that GNN from the MPNN model can be Turing universal under certain conditions and gave impossibility results of GNN with restricted depth and width for solving certain graph problems.

Similarly, there is a significant lack of foundational work regarding safety of GNN. Research in this direction is almost exclusively concerned with developing adversarial attack and defense methods [10,11,12,13] or assessing and improving the adversarial vulnerability of GNN [14,15]. Assume that a GNN is used for some classification task. The basic idea of adversarial methods is to find (or defend against) minimal pertubations to formerly correctly classified graphs, such that after the
perturbation the graph is misclassified. However, by nature such approaches cannot guarantee for
given GNN that misclassifications or, more general, that unwanted behaviour of some kind does not
occur. Analysing the behaviour of dynamic systems is the home ground of formal verification which
has proved to be a very useful tool for guaranteeing the safety or, more generally, expected behaviour
of programs using general methodologies like model checking, theorem proving, static analysis, etc.
Such techniques have successfully been used to detect errors in particular dynamic systems, like
software, hardware, protocols, etc. Given the ubiquity of GNN in all sorts of applications, including
safety-critical ones, it is imminent to apply formal verification techniques to GNN as well, following
the employment of such techniques for classical neural networks.

In this paper we formulate and study the problem of formal verification for GNN. In Section 3 we
derive two general verification tasks in the context of GNN, resulting from two of their major appli-
cations – (whole) graph classification and node classification. In both cases, the obvious unwanted
behaviour is misclassification in general or that of particularly specified inputs. We are interested in
the possibility to guaranteeing their absence, in other words the safety of a given GNN. Ruling out
such misclassifications boils down to the question after the existence of certain inputs on which the
GNN produces certain outputs. Using this understanding, we formulate both verification tasks as de-
cision problems following the pattern: given a GNN, some specification of inputs and a specification
of outputs, decide whether there is a valid input that leads to a valid output.

Clearly, there are potentially infinitely many different inputs to a GNN. This does not automatically
render the verification problems undecidable; for instance, there also are infinitely many different
inputs to an ordinary neural network (NN) in the form of vectors over \( \mathbb{R} \). Yet, the correspon-
ding verification problem for NN is decidable (in fact NP-complete \[17 \] \[18 \]), i.e. there are methods
which can analyse this (countably) infinite search space by finite means, for instance algorithms
for the Linear Programming problem, cf. \[23 \]. So searching for particular inputs in spaces of real
numbers is not necessarily problematic, and one may be inclined to expect the verification problem
for graph neural networks to be decidable, too, as their inputs are finite graphs labeled with vectors
over \( \mathbb{R} \). Intuitively, discrete structures like finite graphs may seem easier than continuous spaces but
computationally, the former are often harder. So perhaps surprisingly or not, the verification problem
for graph-classifier GNN from the common message passing neural network model (MPNN) \[21 \]
turns out to be undecidable. We show that the discrete part of their inputs can be enforced to encode
structures which model the solutions of known undecidable problems. In other words, the step
from neural networks to graph neural networks is in fact a big jump when it comes to the problem
of formal verification, and there is no implementable procedure whatsoever that takes a GNN and
decides whether it misclassifies some input according to some (rather simple) specification. This is
the purpose of Section 4.

For node-classifier GNN from the MPNN model the situation is different: if the degree of the con-
sidered class of graphs is restricted then the verification problem is decidable. We argue for this in
Section 5. It should be clear that both of these and possible future results on this strongly depend on
the expressiveness of the considered GNN and specifications. We discuss these dependencies and
their influences in Section 6.

2 Preliminaries

We denote vectors using bold symbols like \( \mathbf{x} \) or \( \mathbf{y} \) and sets using capital letters like \( I \) or \( M \). Let \( I \) be
a finite set. We denote a vector of \( |I| \) variables with \( x_I \) and usually refer to the \( i \)th dimension with
\( x_i \) for \( i \in I \). A multiset is defined like a regular set, but it can contain duplicates.

Undirected, labeled graphs and trees A graph \( G \) is a triple \((V,E,L)\) where \( V \) is a finite set
of nodes, \( E \subseteq V^2 \) a symmetric set of edges and \( L : V \to \mathbb{R}^n \) is a labeling function, assigning
a real-valued vector to each node. We define the neighbourhood \( N_v \) of a node \( v \) as the set \{ \( v' \mid
(v,v') \in E \} \). We call the minimal \( d \in \mathbb{N} \) for which holds for all \( v \in V \) that \( |N_v| \leq d \) the degree
of \( G \) and respectively call \( G \) a \( d \)-graph. We say that a node \( v \in V \) is has color \( i \) if \( F(v)_i = 1 \).

\[1\] This notion of a verification problem is the same as one of the major problems in classical software analysis,
the so-called reachability problem. Additionally, it is also without a doubt the most considered problem in
context of formal verification of classical neural networks. A thorough survey of the latter is given by Huang
et al. \[16 \].
We call these small NN gadgets. The GNN input size of \((\text{NN})\). We use relatively small NN as building blocks to describe the structure of more complex ones. A specification \(|\phi\rangle\) maps a graph \((V,E,L)\) to a single vector from \(\mathbb{R}^m\) as activation across all layers and simply refer to these as neural networks (NN). We only consider classical feed-forward neural networks using ReLU given by \(R(x) = \max(0, x)\) as activation across all layers and simply refer to these as neural networks (NN). We use relatively small NN as building blocks to describe the structure of more complex ones. We call these small NN gadgets and typically define a gadget by specifying its computed function. Obviously, this way of defining a gadget is unambiguous as there are infinitely many NN computing the same function. We assume that it is clear in the respective situation which NN is the obvious candidate. Let \(N\) be a NN. We call \(N\) positive if for all inputs \(x\) holds that \(N(x) > 0\). We call \(N\) upwards bounded if there is \(\hat{N}\) with \(\hat{N} \in \mathbb{R}\) such that \(N(x) \leq \hat{N}\) for all inputs \(x\).

**Neural networks** We only consider classical feed-forward neural networks using ReLU given by \(R(x) = \max(0, x)\) as activation across all layers and simply refer to these as neural networks (NN). We use relatively small NN as building blocks to describe the structure of more complex ones. We call these small NN gadgets and typically define a gadget by specifying its computed function. Obviously, this way of defining a gadget is unambiguous as there are infinitely many NN computing the same function. We assume that it is clear in the respective situation which NN is the obvious candidate. Let \(N\) be a NN. We call \(N\) positive if for all inputs \(x\) holds that \(N(x) > 0\). We call \(N\) upwards bounded if there is \(\hat{N}\) with \(\hat{N} \in \mathbb{R}\) such that \(N(x) \leq \hat{N}\) for all inputs \(x\).

**Graph neural networks** We only consider Graph Neural Networks (GNN) corresponding to the model of Message Passing Neural Networks [2] or Aggregation-Combine Graph Neural Networks [20]. A Graph Neural Network (GNN) \(N\) consists of a sequence of \(k\) GNN layers \(l_1, l_2, \ldots, l_k\) followed by a readout layer \(l_r\). A GNN layer \(l_i\) specifies an aggregation function \(agg_i\) and a combination function \(comb_i\). The aggregation function \(agg_i\) maps a finite multiset of vectors from \(\mathbb{R}^m\) to a single vector from \(\mathbb{R}^m\). The combination function \(comb_i\) computes a vector \(\mathbb{R}^{m+1} \rightarrow \mathbb{R}^n\). Given a single vector \(x \in \mathbb{R}^m\) and a finite multiset \(M\) of vectors from \(\mathbb{R}^m\), layer \(l_i\) computes \(l_i(x,M) = comb_i(x,agg_i(M))\). We call \(m\) the input size of \(l_i\) and \(n\) its output size. A readout function reads maps a finite multiset of vectors from \(\mathbb{R}^m\) to a vector from \(\mathbb{R}^n\). We call \(m\) the input size of read and \(n\) its output size. All layers \(l_i\) with \(1 \leq i \leq k\) of \(N\) are build such that the input size of \(l_i\) equals the output size of \(l_{i-1}\) as well as the input size of \(l_e\) equals the output size of \(l_k\). The input size of \(N\) is given by the input size \(m\) of \(l_1\) and its output size by the output size \(n\) of \(l_r\). The GNN \(N\) maps a graph \(G = (V,E,L)\) where \(L: V \rightarrow \mathbb{R}^m\) to a real-valued vector from \(\mathbb{R}^n\) in the following way: for each node \(v \in V\) it recursively computes a vector \(x^k_v \in \mathbb{R}^m\) with \(x^0_v = L(v)\) and \(x^i_v = l_i(x^{i-1}_v, M^{i-1}_v)\) where \(M^{i-1}_v\) is the multiset of vectors \(x^{i-1}_{v'}\) of all neighbours \(v'\) of \(v\). After this it applies the read function to the multiset of vectors \(x^k_v\) for all \(v \in V\) which computes the output of the overall network. We distinguish between two categories of GNN based on different kinds of used read functions. The first one are node-classifier GNN. The read function of a node-classifier GNN \(N\) projects its input multiset to a single vector \(x^k_v\) for some node \(v\) and then computes its output based on \(x^k_v\). We denote the application of \(N\) to some graph \(G\) with projection to the vector of a node \(v\) with \(N(G,v)\). The second one are graph-classifier GNN. The read function of a graph-classifier GNN \(N\) aggregates its input multiset into a single vector \(x\) and computes its output based on \(x\). We denote the application of \(N\) to some graph \(G\) with \(N(G)\). We only consider GNN where the aggregation, combination and readout layers are given as follows: \(agg_i(M) = \sum_{x \in M} x\), \(comb_i(x, M) = N_i(x, agg_i(M))\) where \(N_i\) is a NN and \(read(M) = N_r(\sum_{x \in M} x)\) where, again, \(N_r\) is a NN.

**Vector specifications** A (real-valued) vector specification \(\phi\) for a given set of variables \(X\) is defined by the grammar \(\phi ::= \varphi \land \varphi \mid t \leq b, t ::= c \cdot x \mid t + t\) where \(b,c \in \mathbb{Q}\) and \(x \in X\) is a variable. A specification \(\varphi\) with occurring variables \(x_0, \ldots, x_{n-1}\) is satisfied by \(x = (r_0, \ldots, r_{n-1}) \in \mathbb{R}^n\) if each inequality in \(\varphi\) is satisfied in real arithmetic with each \(x_i\) set to \(r_i\).

**Post’s Correspondence Problem (PCP)** A standard and well-known undecidable [21] decision problem is: given a \(P = \{(\alpha_1, \beta_1), (\alpha_2, \beta_2), \ldots, (\alpha_k, \beta_k)\} \subseteq \Sigma^* \times \Sigma^*\) for some alphabet \(\Sigma\), decide whether there is a non-empty sequence of indices \(i_1, i_2, \ldots, i_l\) from \(\{1, \ldots, k\}\) such that \(\alpha_{i_1}\alpha_{i_2}\cdots \alpha_{i_l} = \beta_{i_1}\beta_{i_2}\cdots \beta_{i_l}\). The \(\alpha_i, \beta_i\) are also called tiles. PCP is known to be undecidable when \(|\Sigma| \geq 2\), i.e. we can always assume \(\Sigma = \{a, b\}\).

Consider the solvable instance \(P_0 = \{(aab, aa), (b, bb), (ba, bb)\}\). It is not hard to see that \(I = 1, 3, 1, 2\) is a solution for \(P_0\). Furthermore, the corresponding sequence of tiles can be visualised as shown in Figure [1] The upper word is produced by the \(\alpha_i\) parts of the tiles and the lower one by the \(\beta_i\). The end of one and beginning of the next tile are visualised by the vertical part of the step lines.
3 Canonical graph neural network verification problems

Despite its obvious connection to the general problem of safe AI, formal verification of GNN has received little attention in the literature. We therefore start by describing the problem of GNN verification in general which naturally leads to the definition of well-defined decision problems, necessary to establish (un-)decidability results that help form a picture of the limits of automatic verification in machine learning.

Independent of the specific application, GNN are usually used for one of two tasks: (whole) graph classification or node classification. Let \( G \) be a graph with set of nodes \( V \) and \( K \) be a finite set of classes. In a graph classification task, a trained graph-classifier GNN \( N \) maps \( G \) to one of these classes from \( K \). In other words, \( N \) classifies the graph \( G \) according to \( K \). In a node classification task, a trained node-classifier GNN \( N' \) is used to map a specific node \( v \in V \) to one of the classes from \( K \). Again, in other words, \( N' \) classifies \( v \) of \( G \) according to \( K \). All useful applications that involve such classification tasks have one thing in common, namely that misclassification of inputs is generally undesirable. Additionally, depending on the application, misclassification of one kind can be less or more dangerous than others. For example, in safety-critical contexts missing a critical situation is oftentimes worse than giving a false alarm. Therefore, a reasonable and useful understanding of GNN verification is one of methods which give guarantees about the absence of misclassifications for classes of input graphs specified in a particular way.

This notion of GNN verification can be further generalised and abstracted. Usually, a GNN (and NN in general) used for a \( k \)-class classification is designed such that it outputs a \(|K|\)-dimension vector \( y \) which is the input for a function, like softmax, giving a probability for the membership of the current input to each class \( k \). However, the latter step merely transforms vector values \( y \) into single real values and is irrelevant for decidability concerns. So it suffices to consider problems which ask whether such a vector \( y \) satisfies certain conditions.

We therefore propose the following (pattern of a) decision problem as capturing the core idea of GNN verification as argued above: given a GNN \( N \), some specification \( \varphi \) of valid inputs (a graph with or without a specified node), and a specification \( \psi \) of valid outputs, decide whether there is an input that satisfies \( \varphi \) such that the output of \( N \) under this input satisfies \( \psi \). Note that this indeed constitutes a pattern as one may consider this for specific kinds of GNN, specific graph/node specifications, output specifications, or even graphs of specific structure only, either for the purpose of meeting the demands of a particular application scenario or, not least, because decidability of GNN verification may well depend on the expressiveness of the GNN model, the graph classes and the specification languages under consideration. Hence, these should of course be considered as parameters to the GNN verification problem.

Here, we consider graphs, GNN and vector specifications as defined in Section 2. We only consider the trivial input specification \( \top \) that is satisfied by any input. Clearly, undecidability – see Section 4 – then also transfers to richer input specifications. However, decidability – see Section 5 – is clearly not automatically retained for richer input specifications. The exact formal decision problems studied in the following sections are thus defined as follows.

**Definition 1.** The graph-classifier verification problem \( \text{GVP}^\top \) is: given a graph-classifier GNN \( N \) and vector specification \( \varphi \), decide whether there is a graph \( G \) such that \( N(G) \) satisfies \( \varphi \). The node classifier verification problem \( \text{NVP}^\top \) is: given a node-classifier GNN \( N' \) and vector specification \( \varphi \), decide whether there is a graph \( G \) with node \( v \) such that \( N'(G, v) \) satisfies \( \varphi \). The problem \( \text{NVP}^d \) for \( d \in \mathbb{N} \) is defined as \( \text{NVP}^\top \) but restricted to graphs of degree \( d \).

See Section 6 for a brief discussion on the robustness of the results presented in the next sections under changes of the underlying GNN and specifications formalisms.
4 Verifying graph classifiers is undecidable

The ultimate goal of this section is to show undecidability of $\text{GVP}^\top$. To do so, we define a satisfiability problem for a logic of graphs labeled with real-valued vectors, which we call Graph Linear Programming (GLP) as it could be seen as an extension of ordinary linear programming on graph structures. We prove that its undecidability by a reduction from PCP. This handles much of the intricacies of encoding discrete structures – here: words witnessing a solution of a PCP instance – by means of vectors of real numbers and the operations that can be carried out on them inside a GNN.

We then see that the graph linear programs in the image of the reduction are of a particular shape which can be used to define a – therefore also undecidable – fragment, called Discrete Graph Linear Programming (DGLP). We then show how this fragment can be reduced to $\text{GVP}^\top$, thus establishing its undecidability in a way that separates the structural from the arithmetical parts in a reduction from PCP to $\text{GVP}^\top$. As a side-effect, with GLP we obtain a relatively natural undecidable problem on graphs and linear real arithmetic which may possibly serve to show further undecidability results on similar graph neural network verification problems.

4.1 From PCP to GLP

We begin by defining the Graph Linear Programming problem GLP. Let $X = \{x_1, \ldots, x_n\}$ be a set of variables. A node condition $\varphi$ is a formula given by the syntax

$$\varphi := \sum_{i=1}^{n} a_ix_i + b_i(\circ x_i) \leq c \mid \varphi \land \varphi \mid \varphi \lor \varphi$$

where $a_i, b_i, c \in \mathbb{Q}$. Intuitively, the $x_i$ are variables for a vector of of $n$ real values, constituting a graph’s node label, and $\circ$ is used to describe access to the node’s neighbourhood, resp. their labels.

We write $\text{sub}(\varphi)$ for the set of subformulas of $\varphi$ and $\text{Var}(\varphi)$ for the set of variables occurring inside $\varphi$. We use the abbreviation $t = c$ for $t \leq c$ and $t \leq -c$.

Let $G = (V, E, L)$ be a graph with $L : V \to \mathbb{R}^n$. A node condition $\varphi$ induces a set of nodes of $G$, written $[\varphi]^G$, and is defined inductively as follows.

$$v \in [\sum_{i=1}^{n} a_i x_i + b_i(\circ x_i) \leq c]^G \iff \sum_{i=1}^{n} a_i L(v)_i + b_i(\sum_{v' \in N_v} L(v')_i) \leq c$$
$$v \in [\varphi_1 \land \varphi_2]^G \iff v \in [\varphi_1]^G \cap [\varphi_2]^G$$
$$v \in [\varphi_1 \lor \varphi_2]^G \iff v \in [\varphi_1]^G \cup [\varphi_2]^G$$

If $v \in [\varphi]$ we say that $v$ satisfies $\varphi$.

A graph condition $\psi$ is a formula given by the syntax $\psi := \sum_{i=1}^{n} a_i x_i \leq c \mid \psi \land \psi$, where $a_i, c \in \mathbb{Q}$. The semantics of $\psi$, written $[\psi]$, is the subclass of graphs $G = (V, E, L)$ with $L : V \to \mathbb{R}^n$ such that

$$G \in [\sum_{i=1}^{n} a_i x_i \leq c] \iff \sum_{i=1}^{n} a_i (\sum_{v \in V} L(v)_i) \leq c,$$
$$G \in [\psi_1 \land \psi_2] \iff G \in [\psi_1] \cap [\psi_2].$$

Again, if $G \in [\psi]$ we say that $G$ satisfies $\psi$.

The problem GLP is defined as follows: given a graph condition $\psi$ and a node condition $\varphi$ over the same set of variables $X = \{x_1, \ldots, x_n\}$, decide whether there is a graph $G = (V, E, L)$ with $L : V \to \mathbb{R}^n$ such that $G$ satisfies $\psi$ and all nodes in $G$ satisfy $\varphi$. Such an $L = (\psi, \varphi)$ is called a graph linear program, which we also abbreviate as GLP. It will also be clear from the context whether GLP denotes a particular program or the entire decision problem.

Theorem 1. GLP is undecidable.

Proof. We sketch the proof here and give a full version in Appendix A.1. The proof is done via reduction from PCP. Given a PCP instance $P = \{ (\alpha_1, \beta_1), (\alpha_2, \beta_2), \ldots, (\alpha_k, \beta_k) \}$ we construct a GLP $L_P$, such that $L_P$ is only satisfiable by a graph encoding a sequence of tiles of $P'$ in the following way. Reconsider the solution $I = 1, 3, 1, 2$ of the PCP instance $P_3$ over the alphabet $\{a, b\}$ as shown in Figure 1. This visualization implies a way of encoding a sequence of tiles as a graph with at least two label dimensions. We call these dimensions $x_a$ and $x_b$. 

5
We define $\phi$ where $M$ is the set of colours. If $w_i = a$ then $x_a = 1$ and $x_b = 0$ of the corresponding node and vice-versa if $w_i = b$. Analogously, $\beta_1 \beta_2 \beta_3 \beta_2$ is represented by the blue chain. The borders between two tiles are represented as edges between the yellow and blue nodes corresponding to the starting positions of a tile. The encoding as a graph uses additional auxiliary nodes, edges and label dimensions, in order to ensure that the labels along the yellow and blue nodes indeed constitute a valid PCP solution, i.e. the sequences of their letter labels are the same, and they are built from corresponding tiles in the same order. In Figure 2, these auxiliary nodes and edges are indicated by the dashed parts.

Let $L_P = (\psi_P, \varphi_P)$ be the GLP resulting from $P$. Its components are of course still to be defined. First, we need some basic insights into node conditions. A node condition $\bigwedge_{i \in I} \bigvee_{m \in M_i} x_i = m$ where $M_i \subseteq \mathbb{R}$ expresses that a dimension $x_i$ must be discrete. If $M_i = \{0, 1\}$ for some discretised dimension $x_i$ then this dimension can be seen as the representation of a colour $c$. In this case, we call $x_i$ a colour dimension, a colour and use $c$ as abbreviation for $x_i$. Now, a node condition $\psi_P$ expresses that if $v$ is of colour $c$ then some condition $\varphi_P$ must hold. We abbreviate this by writing $c \rightarrow \varphi$. Consider some subset of colours $C'$. The node expression $\bigwedge_{c \in C'} (\varphi \rightarrow \bigwedge_{c' \neq c} \neg \varphi \lor \bigvee_{c' \neq c} \varphi')$ expresses that a node satisfying this expression must be coloured with exactly one colour of $C'$.

We define $\varphi_P := \varphi_{\text{cond}} \land \varphi_{\text{discr}}$ where $\varphi_{\text{discr}}$ discretises dimensions or makes them colours and $\varphi_{\text{cond}} := \bigwedge_{c \in C} \varphi_c \land \lnot c \rightarrow \lnot \varphi_c$ enforces exactly the nodes of colour $c$ to satisfy $\varphi_c$, for some set of colours $C$. The existence of the two chains can then be enforced by $L_P$ using two colours, for example yellow and blue, demanding that each node has one or two neighbours of its own colour and that for each colour there are exactly two nodes with one neighbour in $G$. The last property is expressed using $\psi_P$ by first colouring each such end nodes with additional colours (yellow, end) and (blue, end) and then using $x_{\text{yellow, end}} = 2 \land x_{\text{blue, end}} = 2$.

It is a little bit more tricky to let $L_P$ enforce that (i) both chains encode the same word, and that (ii) the words encoded by both chains correspond to a correct sequence of tiles from $P$. Property (i) is equivalent to the fact that each node is labelled with $a$ or $b$ if its counterpart is labelled with $a$ respectively $b$. For this, $L_P$ enforces a grid-like structure between the chains, as indicated by the dashed vertical lines in Figure 2 using extra colours (yellow, $a$), (yellow, $b$), (blue, $a$) and (blue, $b$) and checking whether they are equal between the yellow and blue node pairs using the formula $x_{\text{yellow, end}} = \exists x_{\text{blue, end}} = 2 \land x_{\text{blue, end}} = 2$ and so on.

Property (ii) above is equivalent to the fact that (ii.A) all tiles’ starting points are connected without overlapping each other, and that (ii.B) a node is coloured with $a$ or $b$ if the corresponding symbol in the tile sequence is $a$ resp. $b$. To ensure (ii.A), $L_P$ enforces the existence of a third chain, indicated by the dashed and light-green nodes in Figure 2 in between the connections of the other two chains and demands that the order of connections $\text{yellow} \rightarrow \text{green} \rightarrow \text{blue}$ follows the order of the green nodes from left to right. For (ii.B) it suffices to check for each node starting a tile $\alpha_i$ or $\beta_i$ that colours $a$ and $b$ of nodes in distance $|\alpha_i|$ respectively $|\beta_i|$ to the right are correct. Obviously, node conditions can only express properties about the direct neighbourhood of a node. But note that $|P| < \infty$, i.e. the maximum size of a tile part $\alpha_i$ or $\beta_i$ is bounded by some $m$. In the case of $P_0$ we have $m = 3$ for example. Therefore, $L_P$ enforces a colouring $1, 2, \ldots, m, 1, 2, \ldots$ of the blue and yellow chain and demands for each colour $i$ that each node of this colour has stored all colour information of its $i + 1$ coloured neighbours in sufficiently many extra dimensions. Due to the chainlike structure, this ensures that a node has stored the $a$ and $b$ colour information of its two neighbours to the right.

Figure 2: Encoded solution $I$ of PCP instance $P_0$. 

The encoding is depicted in form of solid lines and nodes in Figure 2. The word $w_a = \alpha_1 \alpha_2 \alpha_3 \alpha_2$ is represented by the chain of yellow nodes from left to right in such way that there is a node for each symbol $w_i$ of $w_a$. If $w_i = a$ then $x_a = 1$ and $x_b = 0$ of the corresponding node and vice-versa if $w_i = b$. Analogously, $\beta_1 \beta_2 \beta_3 \beta_2$ is represented by the blue chain. The borders between two tiles are represented as edges between the yellow and blue nodes corresponding to the starting positions of a tile. The encoding as a graph uses additional auxiliary nodes, edges and label dimensions, in order to ensure that the labels along the yellow and blue nodes indeed constitute a valid PCP solution, i.e. the sequences of their letter labels are the same, and they are built from corresponding tiles in the same order. In Figure 2, these auxiliary nodes and edges are indicated by the dashed parts.
In summary, all the necessary requirements ensuring that a graph encodes a solution for a given pcp instance \( P \) can be expressed using node and graph conditions in a GLP \( L_P \) which is satisfiable iff \( P \) has a solution. Moreover, \( L_P \) is effectively constructible from \( P \). Hence, undecidability of pcp transfers to GVP\(^\top\).

GLP seems to be too expressive in all generality for a reduction to GVP\(^\top\), at least it does not seem (easily) possible to mimic arbitrary disjunctions in a GNN. However, the node conditions \( \varphi \) resulting from the reduction from PCP to GLP are always of a very specific form \( \varphi = \varphi' \land \varphi_{\text{discr}} \) where \( \varphi_{\text{discr}} \) is build like above with \( M_i \subseteq \mathbb{N} \) and \( \varphi' \) has the following property. Let \( X \) be the set of non-discrete dimensions as given by \( \varphi_{\text{discr}} \). For each \( \varphi_1 \lor \varphi_2 \in \varphi(\varphi') \) it is the case that \( \text{Var}(\varphi_1) \cap X = \emptyset \) or \( \text{Var}(\varphi_2) \cap X = \emptyset \). In other words, in each disjunction in \( \varphi' \) at most one disjunct contains non-discretised dimensions. We call this fragment of GLP Discrete Graph Linear Programming (DGLP) and, likewise, also use DGLP for the corresponding decision problem. The observation that the reduction function from PCP constructs graph linear programs which fall into DGLP (see Appendix A.1) immediately gives us the following result.

**Corollary 1.** DGLP is undecidable.

### 4.2 From DGLP to GVP\(^\top\)

**Theorem 2.** GVP\(^\top\) is undecidable.

**Proof.** By a reduction from DGLP. Given a DGLP \( L = (\varphi, \psi) \) we construct a GNN \( N_L \) that gives a specific output, namely 0, if and only if its input graph \( G \) satisfies \( L \) and therefore is a witness for \( L \in \text{GLP} \). Let \( m, n \in \mathbb{R} \) with \( m \leq n \) and \( M = \{i_1, i_2, \ldots, i_k\} \subseteq \mathbb{N} \) such that \( i_j \leq i_{j+1} \) for all \( j \in \{1, \ldots, k-1\} \). We use the auxiliary gadget \( \langle x \in [m; n]\rangle := R(R(x - n) - R(x - (n + 1)) + R(m - x) + R(m - 1 - x)) \) to define the gadgets

\[
\begin{align*}
(x \leq m) & := R(R(x - m) - R(x - (m + 1))) \\
(x \in M) & := R(\langle x \in [i_1; i_k]\rangle + \sum_{j=1}^{k-1} R(\frac{(i_{j+1} - i_j)}{2}) - (R(x - \frac{i_j + i_{j+1}}{2}) + R(\frac{i_j + i_{j+1}}{2} - x))).
\end{align*}
\]

Each of the gadgets above fulfills specific properties which can be inferred from their functional forms without much effort: let \( r \in \mathbb{R} \). Then, \( \langle r \leq m \rangle = 0 \) if and only if \( r \leq m \), and \( \langle r \in M \rangle = 0 \) if and only if \( r \in M \). Furthermore, both gadgets are positive and \( \langle x \leq m \rangle \) is upwards bounded for all \( m \) by 1 with the property that \( |r - m| \geq 1 \) implies \( \langle r \leq m \rangle = 1 \). We formally prove these properties in Appendix A.2. We use \( \langle x = m \rangle \) as an abbreviation for \( \langle x \leq m \rangle + \langle x \leq m \rangle \).

The input size of \( N_L \) equals the amount of variables occurring in \( \varphi \) and \( \psi \). \( N_L \) has one layer with two output dimensions \( y_{\text{discr}} \) and \( y_{\text{cond}} \) and the readout layer has a single output dimension \( y' \). The subformula \( \varphi_{\text{discr}} \) is \( \sum_{i \in I} \langle x_i \in M_i \rangle \) and then checked using \( \langle y_{\text{discr}}^1 = 0 \rangle \) in the readout layer.

The remaining part of \( \varphi \) is represented in output dimension \( y_{\text{cond}} \) in the following way. Obviously, an atomic \( \leq \) formula is represented using \( a(x \leq m) \) gadget. A conjunction \( \varphi_1 \land \varphi_2 \) is represented by a sum of two gadgets \( f_1 + f_2 \) where \( f_1 \) represents \( \varphi_1 \). For this to work, we need the properties that all used gadgets are positive and that their output is 0 when satisfied.

To represent a disjunction \( \varphi_1 \lor \varphi_2 \) where \( f_1 \) and \( f_2 \) are the gadgets representing \( \varphi_1 \) resp. \( \varphi_2 \), we need the fact that \( L \) is a DGLP. W.l.o.g. suppose that \( \varphi_1 \) only contains discrete variables and that \( \varphi_{\text{discr}} \) is satisfied. Then we get: if \( \varphi_1 \) is not satisfied then the output of \( f_1 \) must be greater or equal to 1. The reason for this is the following. If the property of some \( \langle x \leq m \rangle \) gadget is not satisfied its output must be 1, still under the assumption that its input includes discrete variables only. Furthermore, as \( \langle x \leq m \rangle \) is positive and upwards bounded, the value of \( f_2 \) must be bounded by some value \( k \in \mathbb{R}^{>0} \). Therefore, we can represent the disjunction using \( R(f_2 - k R(1 - f_1)) \). Note that this advanced gadget is also positive and upwards bounded. Again, the value of \( y_{\text{cond}} \) is checked in the readout layer using \( \langle y_{\text{cond}}^1 = 0 \rangle \). The graph condition \( \psi \) is represented using a sum of \( \langle x \leq m \rangle \) gadgets.

Thus, we can effectively translate a DGLP \( L \) into a pair \((N_L, y')\) of a GNN and an output specification such that there is a graph \( G \) with \( N_L(G) = 0 \) if and only if \( G \) satisfies \( L \), i.e. \( L \in \text{DGLP} \). This transfers the undecidability from DGLP to GVP\(^\top\).

**Corollary 2.** GVP\(^\top\) is already undecidable for GNNs with a single layer.
Let \( \phi \) be a node classifier with \( k \) layers and output dimension \( m \) and consider some graph \( G \) with specified node \( v \) such that \( N(G, v) = y \) for some \( y \in \mathbb{R}^m \). The crucial insight, leading to the decidability of \( \text{NVP}^T_d \) is that there is a tree \( B \) of finite depth \( k \) and with root \( v_0 \) such that \( N(B, v_0) = y \). The intuitive reason for this is that \( N \) can update the label of node \( v \) using information from neighbours of \( v \) of distance at most \( k \). For example, assume that \( k = 2 \) and \( G, v \) are given as shown on the left side of Figure 3, where the information of a node, given by its label, is depicted using different colours: \( y \) (yellow), \( b \) (blue), \( r \) (red), \( g \) (green) and \( p \) (pink). As \( N \) only includes two layers, information from the unfilled (white) nodes are not relevant for the computation of \( N(G, v) \) as their distance to \( v \) is greater than 2. Take the tree \( B \) on the right side of Figure 3. We get that \( N(G, v) = N(B, v_0) \) since, clearly, all the information that is accumulated in \( v \) in \( G \) by a GNN with 2 layers is also accumulated in \( v_0 \) in \( B \) in the same way.

**Theorem 3.** \( \text{NVP}^T_d \) is decidable in \( \text{NEXPTIME} \) for any \( d \in \mathbb{N} \).

**Proof.** First we observe the tree-model property for GNN over graphs of bounded degree: let \( N \) be a node classifier with \( k \) layers, \( \varphi \) a vector specification and \( d \in \mathbb{N} \). We have \((N, \varphi) \in \text{NVP}^T_d\) if and only if there is a \( d \)-tree \( B \) of depth \( k \) with root \( v_0 \) such that \( N(B, v_0) \) satisfies \( \varphi \). A full proof of this is given in Appendix B. Consequently, in order to decide the existence of a graph of bounded degree \( d \) that is node-classified by a GNN with \( k \) layers according to some specification \( \varphi \), it suffices to enumerate all possible \( d \)-trees of depth \( k \) and check whether one of them is a witness for \((N, \varphi) \in \text{NVP}^T_d \), provided that the latter can be checked automatically.

A slightly better complexity bound is achieved using the following nondeterministic algorithm.

Let \( d \in \mathbb{N} \) be fixed, and assume that a GNN \( N \) with \( k \) layers and an output specification \( \varphi \) is given. First guess an unlabeled tree \( B = (V, E, v_0) \) of degree at most \( d \) and depth exactly \( k \). This can be done in time \( O\left(\frac{d^{2k-1}}{2^{k-1}}\right) \) as this is the maximal number of nodes in a \( d \)-tree of depth \( k \), when the root-only tree is considered to have depth 1. Let \( V = \{v_0, \ldots, v_l\} \) and assume the combination \( \text{comb} \) as well as the readout function \( \text{read} \) of \( N \) are given by the NN \( N_1, \ldots, N_k, N \), where \( N_1 \) has input dimension \( 2 \times m \) and output dimension \( n \). By definition, the GNN \( N \) applied to \( B \) computes \( N_1(x, v, \sum_{v' \in N_0} x(v')) \) as the new label for each \( v \in V \) after layer \( l_1 \). However, as \( B \) is known at this point we know the neighbourhood for each node \( v \). Therefore, we can represent the whole computation of layer \( l_1 \) by a single NN \( N_1 \) with input dimension \( l \cdot m \) and output dimension \( l \cdot n \) given by \( (N_1(\text{id}(x_{v_0}), \text{id}(\sum_{v \in N_{v_0}} x(v))), \ldots, N_1(\text{id}(x_{v_l}), \text{id}(\sum_{v \in N_{v_l}} x(v)))) \) where \( \text{id}(x) := R(R(x) - R(-x)) \) is a simple gadget computing the identity. In the same way we can transform the computation of layer \( l_i \), \( i \geq 2 \), into a network \( N_i \) using the output of \( N_{i-1} \) as inputs. Then, we can use the output dimensions of \( N_i \) corresponding to node \( v_0 \) as input for \( N_i \) and, thus, get an NN \( N_r \) representing the computation of \( N \) over graphs of structure \( B \) for arbitrary labeling functions \( L \).

This reduces the question of whether there is labeling \( L \) for \( B \) such that \( N(B_L, v_0) \) to the following: given a NN and vector specification \( \varphi \), is there an input \( x \) such that \( N(x) \) satisfies \( \varphi \)? This problem...
is equivalent to the well-known reachability problem for NN with trivial input specifications $\top$, which is known to be decidable and NP-complete as first stated in [17] and investigated further in [18]. Note that the constructed NN is of exponential size, namely in the order of $|B| \cdot |N|$. Whilst the second factor is clearly linear in the original input, the former is exponential as argued above. Moreover, we can now internalize the technique used to prove membership in NP of the reachability problem for NN and furthermore guess, for each ReLU node $v$ in $N_v$, whether its input $x$ is non-negative and its output is $x$ as well, or it is negative and the output is 0. After performing these guesses, we can write down a linear program that has a solution iff there is an input to $N_v$ such that the values in hidden layers induced by this input are in accordance with the guessed states of the ReLU nodes, and the output satisfies $\varphi$. It is well-known that linear programs can be solved in polynomial time [23], but note that the linear program constructed here is of exponential size in $k$ which is part of the input (represented in unary as the number of layers of $N$). This therefore constitutes a nondeterministic procedure running in exponential time deciding the existence of a $d$-tree with corresponding properties and, thus also the existence of a graph of bounded degree $d$. 

The argument for this decidability result also implies a verification algorithm for instances $(N, \varphi)$ of $NVP_d^\top$ for some $d$: transforming the given GNN for each unlabeled $d$-tree of depth $k$ into an NN as described above and then using an algorithm for the reachability problem of NN. If one such iteration gives a positive result then $(N, \varphi) \in NVP_d^\top$, otherwise $(N, \varphi) \notin NVP_d^\top$.

6 Summary and applicability of results

This work presents two major results: we showed that verification of graph-classifier GNN according to the MPNN model and layers represented using NN with ReLU activation is undecidable, even for the trivial input specification $\top$. We also showed that verification of node-classifier GNN from the same setting is decidable if the degree of valid input graphs is bounded. These results can serve as a basis for further research on formal verification of GNN but their extendability depends on several parameters.

Dependency on the GNN model We restricted our investigations to GNN from the MPNN model, which is a blueprint for spatial-based GNN [24]. However, the MPNN model does not directly specify how the aggregation, combination and readout functions of GNN are represented. Motivated by common choices, we restricted our considerations to GNN where the aggregation functions compute a simple sum of their inputs and the combination and readout functions are represented by NN with ReLU activation only. Theorem 2 only extends to verification problems for GNN models that are at least as expressive as the ones considered here. For some minor changes to our GNN setting, like considering NN with other piecewise-linear activation functions, it is easily argued that undecidability still holds. However, as soon as we leave the MPNN or spatial-based model the question of decidability of GVP opens anew. Bridging results about the expressiveness of GNN from different models, for example spatial- vs. spectral-based, is ongoing research like done by Balcilar et al. [25], and it remains to be seen which future findings on expressiveness can be used to directly transfer the undecidability result obtained here.

Analogously, Theorem 2 only extends (possibly with improvements to the exact complexity bounds) to GNN are at most as expressive as the ones considered here. It is not possible, for example, to directly infer the decidability of $NVP_d^\top$ for models like DropGNN [26], which are shown to be more expressive as MPNN.

Dependency on the specifications While the restriction to trivial input specifications taken here makes the undecidability result strong, it makes the decidability result weaker to the same extent. An interesting observation is that NVP is undecidable as soon as we allow for input specifications that can express properties like $\exists v \forall v' E(v, v')$, stating that a valid graph must contain a node that is connected to all other nodes. If the existence of such a “master” node is guaranteed for some input of a GNN, we can use this node to check the properties specified in graph conditions. The same reduction idea as seen in Section 4 can then be used to show the undecidability of NVP with sufficiently expressive specifications. We refer to future work for finding interesting classes of input specifications, for which NVP$^\top$ or NVP$^d$ are decidable.
References

[1] Thomas N. Kipf, Ethan Fetaya, Kuan-Chieh Wang, Max Welling, and Richard S. Zemel. Neural relational inference for interacting systems. In Jennifer G. Dy and Andreas Krause, editors, Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018, volume 80 of Proceedings of Machine Learning Research, pages 2693–2702. PMLR, 2018.

[2] Alex Fout, Jonathon Byrd, Basir Shariat, and Asa Ben-Hur. Protein interface prediction using graph convolutional networks. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA, pages 6530–6539, 2017.

[3] Wenqi Fan, Yao Ma, Qing Li, Yuan He, Yihong Eric Zhao, Jiliang Tang, and Dawei Yin. Graph neural networks for social recommendation. In Ling Liu, Ryan W. White, Amin Mantrach, Fabrizio Silvestri, Julian J. McAuley, Ricardo Baeza-Yates, and Leila Zia, editors, The World Wide Web Conference, WWW 2019, San Francisco, CA, USA, May 13-17, 2019, pages 417–426. ACM, 2019.

[4] Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. AI Open, 1:57–81, 2020.

[5] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In 7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019. OpenReview.net, 2019.

[6] 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 The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019, pages 4602–4609. AAAI Press, 2019.

[7] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In Doina Precup and Yee Whye Teh, editors, Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017, volume 70 of Proceedings of Machine Learning Research, pages 1263–1272. PMLR, 2017.

[8] Martin Grohe. The logic of graph neural networks. In 36th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021, Rome, Italy, June 29 - July 2, 2021, pages 1–17. IEEE, 2021.

[9] Andreas Loukas. What graph neural networks cannot learn: depth vs width. In 8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020. OpenReview.net, 2020.

[10] Xiang Zhang and Marinka Zitnik. Gnnguard: Defending graph neural networks against adversarial attacks. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, Advances in Neural Information Processing Systems, volume 33, pages 9263–9275. Curran Associates, Inc., 2020.

[11] Daniel Zügner and Stephan Günnemann. Adversarial attacks on graph neural networks via meta learning. In International Conference on Learning Representations, 2019.

[12] Uday Shankar Shanthamallu, Jayaraman J. Thiagarajan, and Andreas Spanias. Uncertainty-matching graph neural networks to defend against poisoning attacks. In Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021, pages 9524–9532. AAAI Press, 2021.

[13] Jiaqi Ma, Shuangrui Ding, and Qiaozhu Mei. Towards more practical adversarial attacks on graph neural networks. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, Advances in Neural Information Processing Systems, volume 33, pages 4756–4766. Curran Associates, Inc., 2020.
[14] Jan Schuchardt, Aleksandar Bojchevski, Johannes Gasteiger, and Stephan Günnemann. Collective robustness certificates: Exploiting interdependence in graph neural networks. In International Conference on Learning Representations, 2021.

[15] Simon Geisler, Daniel Zügner, and Stephan Günnemann. Reliable graph neural networks via robust aggregation. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, Advances in Neural Information Processing Systems, volume 33, pages 13272–13284. Curran Associates, Inc., 2020.

[16] Xiaowei Huang, Daniel Kroening, Wenjie Ruan, James Sharp, Youcheng Sun, Emese Thamo, Min Wu, and Xinping Yi. A survey of safety and trustworthiness of deep neural networks: Verification, testing, adversarial attack and defence, and interpretability. Comput. Sci. Rev., 37:100270, 2020.

[17] Guy Katz, Clark W. Barrett, David L. Dill, Kyle Julian, and Mykel J. Kochenderfer. Reluplex: An efficient SMT solver for verifying deep neural networks. In Rupak Majumdar and Viktor Kuncak, editors, Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I, volume 10426 of Lecture Notes in Computer Science, pages 97–117. Springer, 2017.

[18] Marco Sälzer and Martin Lange. Reachability is np-complete even for the simplest neural networks. In Paul C. Bell, Patrick Totzke, and Igor Potapov, editors, Reachability Problems - 15th International Conference, RP 2021, Liverpool, UK, October 25-27, 2021, Proceedings, volume 13035 of Lecture Notes in Computer Science, pages 149–164. Springer, 2021.

[19] David G. Luenberger. Introduction to Linear and Nonlinear Programming. Addison-Wesley, 1973.

[20] Pablo Barceló, Egor V. Kostylev, Mikaël Monet, Jorge Pérez, Juan L. Reutter, and Juan Pablo Silva. The logical expressiveness of graph neural networks. In 8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020. OpenReview.net, 2020.

[21] Emil L. Post. A variant of a recursively unsolvable problem. Bulletin of the American Mathematical Society, 52(4):264–268, 1946.

[22] Moshe Y. Vardi. Why is modal logic so robustly decidable? In Proc. DIMACS Workshop on Descriptive Complexity and Finite Models, volume 31 of DIMACS Series in Discr. Math. and Theor. Comp. Sci., pages 149–183. DIMACS/AMS, 1996.

[23] L. G. Khachiyan. A polynomial algorithm in linear programming. Soviet Mathematics Doklady, 20:191–194, 1979.

[24] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks. IEEE Trans. Neural Networks Learn. Syst., 32(1):4–24, 2021.

[25] Muhammet Balcilar, Guillaume Renton, Pierre Héroux, Benoit Gaüzère, Sébastien Adam, and Paul Honeine. Analyzing the expressive power of graph neural networks in a spectral perspective. In International Conference on Learning Representations, 2021.

[26] Pál András Papp, Karolis Martinkus, Lukas Faber, and Roger Wattenhofer. Dropgnn: Random dropouts increase the expressiveness of graph neural networks. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, Advances in Neural Information Processing Systems, volume 34, pages 21997–22009. Curran Associates, Inc., 2021.
A Verifying graph classifiers is undecidable: proof details

A.1 Proving that GLP and DGLP are undecidable

We use the following abbreviations for GLP. For a set \( C \) of colours we define \( \text{colour}(C) = \bigwedge_C (x_c = 0) \lor (x_c = 1) \) and exactly\_one(C) = \( \bigwedge_C (c \rightarrow (\bigwedge_{c' \neq c} \neg c') \land (\neg c \rightarrow \bigvee_{c' \neq c} c')) \) where \( c := (x_c = 1), \neg c := (x_c = 0), c \rightarrow \varphi := (x_c = 0) \lor \varphi \) and \( \neg c \rightarrow \varphi := (x_c = 1) \lor \varphi \). To keep the notation clear and if its unambiguous, we write \( i \) for some variable \( x_i \) in node and graph conditions.

Let \( G = (V, E, L) \) be a graph. For some nodest set \( V' \subseteq V \) we define \( N_v(V') = \{ v' \mid v' \in N_v, v' \in V' \} \). We call a subset of nodes \( V' = \{ v_1, \ldots, v_k \} \subseteq V, k \geq 2 \) a chain if it holds that \( N_{v_i}(V') = \{ v_2 \}, N_{v_i}(V') = \{ v_1, v_{i+1} \} \) for \( 2 \leq i \leq k-1 \) and \( N_{v_k}(V') = \{ v_{k-1} \} \). We call \( v_i \) start, \( v_i \) a middle node and \( v_k \) end of \( V' \) and assume throughout the following arguments that index 1 denotes the start and the maximal index \( k \) denotes the end of a chain. Let \( V_1 = \{ v_1, \ldots, v_k \} \) and \( V_2 = \{ u_1, \ldots, u_k \} \) be subsets of \( V \) and chains. We say that \( V_1 \cup V_2 \) is a ladder if for all \( v_i, u_i \) holds that \( N_{v_i}(V_1 \cup V_2) = N_{v_i}(V_1) \cup \{ u_i \} \) and \( N_{u_i}(V_1 \cup V_2) = N_{u_i}(V_2) \cup \{ v_i \} \).

First, we show that DGLP can recognize if a graph \( G \) consists of exactly one ladder and one additional chain. If this is the case, we call a graph \( G \) a chain-ladder. Let \( C_3 = \{ c_1, c_2, c_3 \} \) and \( T = \{ (c, s), (c, m), (c, e) \mid c \in C_3 \} \) be sets of colours. Let \( \varphi_{\text{CL}}, \psi_{\text{CL}} \) be the following DGLP over variables \( V, x, x' \) and \( x, c, e, s, \)\( T \): \( C_3 \) of colours:

\[
\begin{align*}
\varphi_{\text{CL}} &:= \varphi_{\text{cond}} \land \bigwedge_{(c_1, T)} \text{exactly\_one}(M) \land \text{colour}(C_3 \cup T) \\
\varphi_{\text{cond}} &:= \bigwedge_{C_3} (\neg c_1 \rightarrow \bigwedge_T (c_1, s) \land (c_1, m) = 2 \land (c_1, e) = 0) \\
& \land (\bigwedge (c_1, m) \rightarrow \bigwedge (c_1, c_1) = 2 \land (c_1, e) = 0) \\
& \land (\bigwedge (c_1, e) \rightarrow \bigwedge (c_1, c_1) = 2 \land (c_1, e) = 0) \\
\psi_{\text{CL}} &:= \bigwedge_{C_3} (c_1, s) = 1 \land (c_1, e) = 1 \land c = (c_1, e)
\end{align*}
\]

**Lemma 1.** If \( G = (V, E, L) \) satisfies \( \varphi_{\text{CL}}, \psi_{\text{CL}} \) then \( G \) is a chain-ladder and if \( G' = (V', E') \) is an unlabeled, chain-ladder then there is \( L' \) such that \( G = (V', E', L') \) satisfies \( \varphi_{\text{CL}}, \psi_{\text{CL}} \).

**Proof.** Assume that \( G \) satisfies \( \varphi_{\text{CL}}, \psi_{\text{CL}} \). By definition, it follows that all nodes \( v \in V \) satisfy \( \varphi_{\text{CL}} \) and \( G \) satisfies \( \psi_{\text{CL}} \).

Let \( v \in V \) be a node. Due to \( \bigwedge_{(c_1, T)} \text{exactly\_one}(M) \land \text{colour}(C_3 \cup T) \) it holds that \( v \) has exactly one colour \( c_1, c_2 \) or \( c_3 \) and exactly one from \( T \). Furthermore, the subformula \( \bigwedge_{C_3} (\neg c_1 \rightarrow \bigwedge_T (c_1, t) \land \cdots) \) implies that there is \( i \in \{ 1, 2, 3 \} \) such that \( v \) is of colour \( c_i \) and \( (c_i, t) \) for some \( t \).

We divide \( V \) into three sets \( V_1, V_2 \) and \( V_3 \) such that \( v \in V_i \) if and only if \( v \) is of colour \( c_i \) and argue that each \( V_i \) is a chain. Let \( v \in V_i \). Note, that the \( V_i \) are disjoint sets. The subformula \( (c_1, s) \rightarrow \bigwedge (c_1, c_1) = 2 \land (c_1, e) = 0 \) implies that \( v \) has 1 or two neighbours from \( V_i \). From the argument above, we know that \( v \) must be of exactly one colour \( (c_i, s), (c_i, m) \) or \( (c_i, e) \). The implications in \( \varphi_{\text{cond}} \) regarding these three colours imply: if \( v \) is of colour \( (c_i, s) \) or \( (c_i, e) \) it must have exactly one neighbour from \( V_i \) and if \( v \) is of colour \( (c_i, m) \) it must have exactly two neighbours from \( V_i \). The graph condition \( \psi_{\text{CL}} \) implies that there is exactly one node with colour \( (c_i, s) \) and one with colour \( (c_i, e) \). In combination, we have that there is a start \( v_s \) and end \( v_e \) in \( V_i \) both having one neighbour in \( V_i \) and all middle nodes \( v_m \) having two.

Next, consider the \( (c_i, s) \) and \( (c_i, e, id) \) label dimensions. We call \( (c_i, id) \) the id of a node with colour \( c_i \). The subformula \( (\neg c_i \rightarrow \cdots \land (c_i, id) = 0 \land (c_i, e, id) = 0) \land \cdots \) implies that if \( v \) is not of colour \( c_i \) then the corresponding dimensions must be 0 and \( (c_i, s) \rightarrow \cdots \land (c_i, e, id) = 0 \) and \( (c_i, m) \rightarrow \cdots \land (c_i, e, id) = 0 \) imply that if it is not an end node then \( c_i \) is id 0 as well. Next, we use the subformula \( (c_i, s) \rightarrow \cdots \land (c_i, id) = 1 \land (c_i, e, id) = 2 \land \cdots \) that \( v_i \) has id 1 and its neighbour has id 2. This implies, that the only neighbour of \( v_i \) is not itself. The same holds for \( v_e \) due to the subformula \( (c_i, e) \rightarrow \cdots \land (c_i, id) = 1 \land \cdots - 1 \). Furthermore, the subformula \( (c_i, e) \rightarrow \cdots \land (c_i, id) = (c_i, id) \) implies that the id of \( v_e \) is stored in \( (c_i, e, id) \). This is used in the graph condition subformula \( c_i = (c_i, e, id) \) to ensure that the amount of nodes in \( V_i \) is equal to the id of \( v_e \). We make a case distinction: if \( v_e \) is the neighbour of \( v_s \) then the id of \( v_e \) is 2 and,
thus, \( V_1 = \{ v_s, v_e \} \) which obviously is a chain. If \( v_e \) is not the neighbour of \( v_s \) then it must be some \( v_m \). The subformula \((c_2(m, m) \rightarrow \cdots \wedge 2(c_1, id) = \circ(c_1, id) \wedge \cdots) \) implies that \( v_m \) is not its own neighbour and that other neighbour \( v'_m \) must have id 3. Now, if \( v'_m = v_e \) then we can make the same argument as in the other case. If not then we get that \( v'_m \) must have a neighbour \( v''_m \neq v'_m \). The node \( v''_m \) must have id 4 and, thus, it did not occur on the chain before. As \( V_1 \) is finite, this sequence must eventually reach \( v_e \) and we get that \( V_1 \) must be a chain.

So far, we argued that \( V = V_1 \cup V_2 \cup V_3 \) with \( V_i \) disjunct and chains. It is left to argue, that \( V_1 \cup V_2 \) forms a ladder. From our previous arguments we know that the nodes of \( V_i \) have incrementing ids from \( v_s \) to \( v_e \) starting with 1. Therefore, the ladder property is ensured by the subformulas \((c_1 \rightarrow \circ c_2 = 1 \wedge (c_1, id) = \circ(c_2, id)) \) and \((c_2 \rightarrow \circ c_1 = 1 \wedge (c_2, id) = \circ(c_1, id)) \).

The other statement of the lemma, namely that there is a labeling function \( L' \) for \( G' \) such that \((\varphi_{CL}, \psi_{CL}) \) is satisfied, is a straightforward construction of \( L' \) following the arguments above. \( \square \)

Let \( G \) be a chain-ladder with ladder \( V_1 \cup V_2 = \{ v_1, \ldots, v_k \} \cup \{ u_1, \ldots, u_k \} \) and chain \( V_3 = \{ w_1, \ldots, w_l \} \). We call \( G \) a PCP-structure if for all \( w_i \) holds \( N_{w_i}(V_1 \cup V_2 \cup V_3) = N_{w_i}(V_3) \cup \{ v_{h_i}, u_{j_i} \} \) for some \( h_i, j_i \in \{ 1, \ldots, k \} \) such that for all \( 2 \leq i \leq l-1 \) holds that \( h_{i-1} < h_i < h_{i+1} \) and \( j_i-1 < j_i < j_{i+1} \).

We show that there is a DGLP that recognizes PCP-structures. Let \( L, M, R \) be colours, called directions. We define \( L + 1 := M, M + 1 := R, R + 1 := L \) and \( d-1 \) analogously. Let \( C_b \) be as above and \( D = \{ \{ c, d \mid c \in C_b, d \in \{ L, M, R \} \} \). The DGLP \((\varphi_{PS}, \psi_{PS}) \) over the variables \( \text{Var}_{PS} = \text{Var}_{CL} \cup \{ x_c \mid c \in D \} \cup \{ x_{d,c, id} \mid d \in \{ L, M, R \}, c \in \{ c_1, c_2 \} \} \) is defined as follows:

\[
\varphi_{PS} := \varphi_{cond} \land \text{exactly-one }(D) \land \varphi_{CL} \land \text{colour } (D)
\]

\[
\varphi_{cond} := (\bigwedge_{C_b} \neg c_i \rightarrow \bigwedge_D \neg (c_i, d))
\]

\[
\land \bigwedge_{C_b} ((c_i, s) \rightarrow (c_i, L) \land \circ(c_i, M) = 1)
\]

\[
\land ((c_i, m) \rightarrow \bigvee_D (c_i, d) \land \bigwedge_{d \neq d'} \circ(c_i, d')) = 1
\]

\[
\land (\bigwedge_{\{c_1, c_2\}} c_i \rightarrow \circ c_3 \leq 1) \land (c_3 \rightarrow \circ c_1 = 1 \land \circ c_2 = 1)
\]

\[
\land \bigwedge_D (\neg (c_3, d) \rightarrow \bigwedge_{\{c_1, c_2\}} (d, c_i, id) = 0) \land ((c_3, d) \rightarrow \bigwedge_{\{c_1, c_2\}} \circ(c_i, id) = (d, c_i, id))
\]

\[
\land \bigwedge_D ((c_3, d) \rightarrow \bigwedge_{\{c_1, c_2\}} \circ(d-1, c_i, id) \leq (d, c_i, id)) \land (d, c_i, id) \leq \circ (d+1, c_i, id)
\]

\[
\psi_{PS} := \psi_{CL}
\]

**Lemma 2.** If \( G = (V, E, L) \) satisfies \((\varphi_{PS}, \psi_{PS}) \) then \( G \) is a PCP-structure and if \( G' = (V', E') \) is an unlabelled PCP-structure then there is labeling function \( L' \) such that \((V', E', L') \) satisfies \((\varphi_{PS}, \psi_{PS}) \).

**Proof.** Assume that \( G \) satisfies \((\varphi_{PS}, \psi_{PS}) \). As \( \varphi_{CL} \) occurs as a conjunct in \( \varphi_{PS} \) and \( \psi_{CL} \) in \( \psi_{PS} \) implies that \( G \) is a chain-ladder. Let \( V_1 \cup V_2 \) be the ladder and \( V_3 \) the chain.

The subformula \( \text{exactly-one}(D) \) and \( \text{colour}(D) \) in combination with \((\bigwedge_{C_b} \neg c_i \rightarrow \bigwedge_D \neg (c_i, d)) \)

imply that a node is of colour \( c_i \) if and only if it is of exactly one colour \((c_i, d)\). From the arguments of \( \text{Lemma 1} \) we know that each node \( v \) has exactly one colour \( c_i \) and, thus, \( v \) also has a corresponding direction \( d \in \{ L, M, R \} \). The subformulas \((c_i, s) \rightarrow (c_i, L) \land \circ(c_i, M) = 1) \) and \((c_i, m) \rightarrow \bigvee_D (c_i, d) \land \bigwedge_{d \neq d'} \circ(c_i, d') = 1) \) imply that start node of chain \( V_i \) has direction \( L \) and its neighbour \( M \) and that the neighbours of each middle node of direction \( d \), characterized by colour \((c_i, m) \), must have directions \( d-1 \) and \( d+1 \). In combination, this implies that each chain \( V_1, V_2 \) and \( V_3 \) is coloured from start to end with \( L, M, R, L \ldots \)

The subformulas \((\bigwedge_{\{c_1, c_2\}} c_i \rightarrow \circ c_3 \leq 1) \) and \((c_3 \rightarrow \circ c_1 = 1 \land \circ c_2 = 1) \) imply that nodes from ladder \( V_1 \cup V_2 \) have at most one neighbour from \( V_3 \) and each node from chain \( V_3 \) has exactly one neighbour from \( V_1 \) and one from \( V_2 \). Consider the dimensions \((d, c_i, id) \). First, the subformula \( \neg (c_3, d) \rightarrow \bigwedge_{\{c_1, c_2\}} (d, c_i, id) = 0) \) and the conditions of \( \varphi_{CL} \) imply that dimension \((d, c_i, id) \) of node \( v \) are nonzero only if \( v \) is from \( V_3 \) and of direction \( d \). The subformula \((c_3, d) \rightarrow \bigwedge_{\{c_1, c_2\}} \circ(d-1, c_i, id) \leq (d, c_i, id) \) leads to the case that each node \( v \in V_3 \) of direction \( d \) has stored the id of its one neighbour from \( V_1 \) in \((c_1, d, id) \) and the id of its one neighbour from \( V_2 \) in \((c_2, d, id) \). Now, the subformulas \((c_3, d) \rightarrow \bigwedge_{\{c_1, c_2\}} \circ(d-1, c_i, id) \leq (d, c_i, id) \) and
We prove this via reduction from PCP. Let $P = \{(\alpha_1, \beta_1), \ldots, (\alpha_k, \beta_k)\}$ be a PCP instance and let $\bar{m} = \max \{L^k_{i=1} \{i|1|, |\beta_i|\}\}$. Let $B = \{(c_i, d, a, j), (c_i, d, b, j) \mid c_i \in \{c_1, c_2\}, d \in \{L, M, R\}, j \in \{0, \ldots, \bar{m} - 1\}\}$ and $S = \{(c_i, d, p, j) \mid c_i \in \{c_1, c_2\}, d \in \{L, M, R\}, p \in \{1, \ldots, \bar{m}, \perp\}, j \in \{0, \ldots, \bar{m}\}\}$ colours. Additionally, we define $S^0 = S \setminus \{(c_i, d, \perp, j) \mid (c_i, d, \perp, j) \in S\}$, $S^0 = \{(c_i, d, p, 0) \mid (c_i, d, p, 0) \in S\}$ and $B^0 = \{(c_i, d, j, 0) \mid (c_i, d, j, 0) \in B\}$. We define the following DGLP $(\varphi_P, \psi_P)$ over the variables $\text{Var}_{PS} \cup \{x_e \mid e \in B \cup S\}$:

$$
\varphi_P := \varphi_{cond} \land \left(\bigwedge_{(c_1, c_2)} c_i \rightarrow \bigwedge_{(B^0, S^0)} \text{exactly-one}(M) \right) \land \varphi_{PS} \land \text{colour}(B \cup S)
$$

$$
\varphi_{cond} := \bigwedge_D (\neg(e_i, d) \rightarrow \bigwedge_{B \cup S} \neg(e_i, d, p, j))
\land \bigwedge_D (c_i, d) \rightarrow \bigwedge_{S, j < \bar{m}} (c_i, d, p, j + 1) = \bigcirc(c_i, d + 1, p, j))
\land \bigwedge_{(c_1, c_2)} (c_i, e) \rightarrow \bigwedge_{B \cup S, p \neq e} \neg(c_i, d, p, j) \land \bigwedge_D (c_i, d, e, 0)
\land \bigwedge_{(c_1, c_2)} (c_i, e) \rightarrow \bigwedge_D \neg(c_i, d, e, 0)
\land \bigwedge_{S^0, p \neq e} (c_1, d, p, 0) \rightarrow \bigcirc c_1 = 1 \land \bigwedge_{j \neq 0} (c_1, d, \alpha_{p[j]}, j) \land (c_1, d, \perp, j))
\land \bigwedge_{p \neq e} (c_2, d, p, 0) \rightarrow \bigcirc c_2 = 1 \land \bigwedge_{j \neq 0} (c_2, d, \beta_{p[j]}, j) \land (c_2, d, \perp, j))
\land \bigwedge_{(c_1, c_2)} (c_i, s) \rightarrow \bigwedge_{S^0} (c_i, d, p, 0)
\land \bigwedge_{(L, M, R)} \bigwedge (c_i, d) \rightarrow (c_1, d, a, 0) = \bigcirc (c_2, d, a, 0) \land (c_1, d, b, 0) = \bigcirc (c_2, d, b, 0))
\land \bigcirc c_3 \rightarrow \bigwedge_{S^0} \bigwedge_{d \in (L, M, R)} \bigcirc (c_1, d, p, 0) = \bigcirc (c_2, d, p, 0) \land \bigwedge_{S^0} \bigcirc (c_1, d, p, 0) = 1
$$

$$
\psi_P := \psi_{PS}
$$

**Theorem 4.** GLP is undecidable.

**Proof.** We prove this via reduction from PCP. Let $P = \{(\alpha_1, \beta_1), \ldots, (\alpha_k, \beta_k)\}$ and $(\varphi_P, \psi_P)$ be like above. Assume that $(\varphi_P, \psi_P)$ is satisfied by $G$.

From the description above, we can see that $\varphi_{PS}$ and $\psi_{PS}$ are conjuncts of $\varphi_P$ respectively $\psi_P$. Therefore, Lemma 2 implies that $G$ is a PCP-structure. Let $V_1 \cup V_2$ be the ladder and $V_3$ the additional chain in $G$. In addition to the colours resulting from $\varphi_{PS}$, the subformulas $\text{colour}(B \cup S)$ and $(\bigwedge_{(c_1, c_2)} c_i \rightarrow \bigwedge_{(B^0, S^0)} \text{exactly-one}(M))$ ensure that $B$ and $S$ are colours and that each ladder node has exactly one colour from $B^0 \subset B$ and $S^0 \subset S$. The idea of these colours is the following: a colour $(c_i, d, p, j) \in B$ with $p \in \{a, b\}$ and $j \in \{0, \ldots, \bar{m} - 1\}$ represents the symbol $(a$ or $b)$ of a node in distance $j$ of a node coloured with $(c_i, d)$. Similarly, colour $(c_i, d, p, j)$ with $p \in \{1, \ldots, \bar{m}, \perp\}$ represents that a node in distance $j$ of a node coloured with $(c_i, d)$ is the start of the final part $\alpha_0$ if $i = 1$, $p \neq \perp, e$ and $\beta_0$ if $i = 2$, $p \neq \perp, e$. In case of $p = e$ the node in distance $j$ is the end node of chain $V_i$ and $p = \perp$ is a placeholder for nodes which are neither a start or a part of a longer nor a part of a tile part.

We argue how $\varphi_P$ ensures the above mentioned properties of colours $(c_i, d, p, j) \in B \cup S$. The subformula $(\neg(c_i, d) \rightarrow \bigwedge_{B \cup S} \neg(c_i, d, p, j))$ ensures that a node of some colour $(c_i, d, p, j)$ must also be of colour $(c_i, d)$. Especially, this implies that nodes from chain $V_3$ do not have any colour $(c_i, d, p, j)$. The subformulas $(c_i, d) \rightarrow \bigwedge_{B, j \leq \bar{m}} (c_i, d, p, j + 1) = \bigcirc (c_i, d + 1, p, j)$ and $(c_i, d) \rightarrow \bigwedge_{S, j \leq \bar{m}} (c_i, d, p, j + 1) = \bigcirc (c_i, d + 1, p, j)$ ensure that a node with colour $(c_i, d)$ stores the information of its $\bar{m}$ right neighbours.
We can see from the definitions of $\varphi$ that $v_0$ only has colour $(c_i, d, e, 0)$. That $d$ matches its colour $(c_i, d)$ is ensured by $\neg(c_i, d) \implies \cdots$. Therefore, its only and neighbour $v$ must have colour $(c_i, d - 1, e, 1)$ plus its own additional colours with $j = 0$. Now, the left neighbour $v'$ of $v$ must have colours $(c_i, d - 2, e, 2)$, the colours equivalent to $v$ with $j = 1$ and its own colours with $j = 0$ and so on. As the maximum $j$ in case of a colour from $S$ is $m$, tilepart start, end or $\perp$ colours are stored in nodes up to distance $m$ to the left of the original node. The same holds for colours from $B$ with distance $m - 1$.

We are set to argue that $G$ encodes a solution of $I$. The subformula \((\bigwedge_{j=0}^{\alpha_p-1} (c_i, d, \alpha_p[j], j) \land (c_i, d, \perp, j)) \land \bigvee_{p'=1}^{k} (c_i, d, p', [\alpha_p]) \lor (c_i, d, e, [\alpha_p])\) ensures that for each node from $V_1$ that is a tilepart start for some $\alpha_p$ that $\alpha_p$ is written to the right without a next tilepart starting \(((\bigwedge_{j=0}^{\alpha_p-1} (c_i, d, \alpha_p[j], j) \land (c_i, d, \perp, j)))\) and that after $\alpha_p$ is finished that either the next tile part starts or the chain ends \((\bigvee_{p'=1}^{k} (c_i, d, p', [\alpha_p]) \lor (c_i, d, e, [\alpha_p]))\). Analogous conditions are ensured for nodes from $V_2$ by the subformula \((\bigwedge_{j=0}^{\beta_p-1} (c_i, d, \beta_p[j], j) \land (c_i, d, \perp, j)) \land \bigvee_{p'=1}^{k} (c_i, d, p', [\beta_p]) \lor (c_i, d, e, [\beta_p])\). Now, the subformula \((\bigwedge_{i_1,\ldots,i_2} (c_i, s) \land (c_i, s))\) ensures that the start nodes of $V_1$ and $V_2$ correspond to a tilepart start. In combination with the previous conditions, this ensures that chains $V_1$ and $V_2$ are coloured with words $w_\alpha$ and $w_\beta$ corresponding to sequences $\alpha_{i_1}\cdots\alpha_{i_k}$ and $\beta_{j_1}\cdots\beta_{j_l}$.

It is left to argue that the words $w_\alpha$ and $w_\beta$ and $i_1\cdots i_h$ and $j_1\cdots j_l$ are equal. The first equality is ensured by \((\langle c_1, d \rangle \implies (c_1, d, a, 0) = \land (c_2, d, a, 0) \land (c_1, d, b, 0) = \land (c_2, d, b, 0))\) and the fact that $V_1 \cup V_2$ is a ladder. The second equality is ensured by \((c_3 \implies (\bigwedge_{d' \in \{L,M,R\}} \land (c_1, d, p, 0) = \land (c_2, d', p, 0)) \land (c_1, d, p, 0)) \land (c_3 = 1 \land \cdots)\) and \((c_2, d, p, 0) \land (c_3 = 1 \land \cdots)\) and the fact that $G$ is a PCP-structure which means that connections between $V_3$ and $V_1$ respectively $V_2$ are not overlapping. Therefore $j_1\cdots j_h = j_1\cdots j_l$ is a solution for $P$ which implies that $P$ is solvable.

The vice-versa direction, namely that if $P$ is solvable then $(\varphi_p, \psi_p)$ is satisfiable, is argued easily: If $P$ is solvable then there is a solution $I$. We argued in Section 4 how to encode $I$ as a PCP-structure $G$. Note that in contrast to Figure 2, the encoding characterized by $\varphi_{PS}$ demands that the end nodes of chain $V_1$ and $V_2$ are not part of solution $I$. Lemma 3 states that for each unlabeled PCP-structure there is a labeling function $L'$ such that $G$ satisfies $(\varphi_{PS}, \psi_{PS})$. Therefore, if we take a matching PCP-structure $G$ without labels, label it with $L'$ and then extend $L'$ with the colours $(c_i, d, p, j)$ according to $I$ and the arguments above we get that $G$ satisfies $(\varphi_p, \psi_p)$.

We can see from the definitions of $\varphi_{CL}$, $\varphi_{PS}$ and $\varphi_P$ and corresponding graph conditions that they belong to the DGLP fragment of GLP.

**Corollary 3.** DGLP is undecidable.

### A.2 Proving that DGLP is reducible to GVP

In the proof of Theorem 2 we claimed that the following properties are valid for $\langle x \leq m \rangle$ and $\langle x \in M \rangle$ gadgets.

**Lemma 3.** Let $r \in \mathbb{R}$ and $(r_1, \ldots, r_k) \in \mathbb{R}^k$ for some $k$. It holds that $\langle r \leq m \rangle = 0$ if and only if $r \leq m$ and $\langle r \in M \rangle = 0$ if and only if $r \in M$. Furthermore, gadgets $\langle x \leq m \rangle$ and $\langle x \in M \rangle$ are positive and $\langle x \leq m \rangle$ is upwards bounded.

**Proof.** The properties of the $\langle x \leq m \rangle$ are straightforward implications of its functional form.

Next, we prove that $\langle r \in [m; n] \rangle = 0$ if and only if $r \notin [m; n]$. Assume that $r \in [m; n]$. It follows that the output of each inner ReLU node is 0 and therefore $\langle r \in [m; n] \rangle = 0$. Next, assume $r < m$. It follows that $R(m - x) > 0$ and as the value of all other inner ReLU nodes must be greater or equal to 0 it follows that $\langle r \in [m; n] \rangle > 0$. The case $r > n$ is argued analogously as $R(x - n) > 0$.

Consider the $(x \in M)$ gadget and assume that $r \in M$. It clearly holds that $r \in [i_1; i_k]$ and therefore that $R((x \in [i_1; i_k]) = 0$. Furthermore, w.l.o.g. let $r = i_l$ for some $1 \leq l < k$. Then, it follows that $\frac{((i_{l+1} - i_l) - (i_{l+1} + i_{l+1} - i))}{2} = R(r - \frac{i_{l+1} + i_{l+1} - r}{2}) + R(\frac{i_{l+1} + i_{l+1} - r}{2}) R(r - \frac{i_{l+1} + i_{l+1} - r}{2}) + R(\frac{i_{l+1} + i_{l+1} - r}{2})$ for $j \neq l$ and, thus, the inner sum is equal to 0 as well. Now, assume that $r \notin M$. If $r < i_1$ or
Thus this lemma in combination with the proof sketch of Theorem 2 given in Section 4 yields a full proof of Theorem 2.

B A proof for the tree-model property of node-classifier GNN

In the proof (sketch) of Theorem 3 we claimed that GNN have the tree-model property. We formally prove this statement in the following.

Let $G = (V, E, L)$ be a graph and $v \in V$. The set of straight $i$-paths $P^i_v$ of $v$ is defined as $P^0_v = \{v\}$, $P^1_v = \{v' \mid (v, v') \in E\}$ and $P^{i+1}_v = \{v_0 \cdots v_{i-1} v_i v_{i+1} \mid v_0 \cdots v_{i-1} v_i \in P^i_v, (v_i, v_{i+1}) \in E, v_{i-1} \neq v_{i+1}\}$. For some path $p = v_0 \cdots v_l$ we define $p_i$ for $i \leq l$ as the prefix $v_0 \cdots v_i$. Furthermore, let $P^i_v = \{(v_0, p_0)(v_1, p_1)\cdots(v_l, p_l) \mid p = v_0 \cdots v_l \in P^i_v\}$ and let $\tilde{P}^i_v(j) = \{(v_j, p_j) \mid (v_0, p_0)\cdots(v_j, p_j)\cdots(v_l, p_l) \in \tilde{P}^i_v\}$ for some path $p = v_0 \cdots v_l$. Let $N$ be a node-classifier GNN. We denote the value of $v$ after the application of layer $l_i$ with $N^{l_i}(G, v)$ for $i \geq 1$. Furthermore, let $N^0(G, v) = L(v)$ for each node $v$.

**Lemma 4.** Let $N$ be a node classifier GNN with $k$ layers, $\varphi$ a vector specification and $d \in \mathbb{N}$. It holds that $(N, \varphi) \in N\text{VP}_d$ if and only if there is a $d$-tree $B$ of depth $k$ with root $v_0$ such that $N(B, v_0)$ satisfies $\varphi$.

**Proof.** Assume that $N, \varphi$ and $d$ are as stated above. The direction from right to left is straightforward. Therefore, assume that $(N, \varphi) \in N\text{VP}_d$. By definition, there exists a $d$-graph $G = (V, E, L)$ with node $w$ such that $N(G, w)$ satisfies $\psi$. Let $B = \{V_B, E_B, L_B, (w, w)\}$ be the tree with node set $V_B = \bigcup_{j=0}^k \tilde{P}^n_w(i)$. The set of edges $E_B$ is given in the obvious way by $\{((v, p), (v', p')) \mid (v, p), (v', p') \in V_B\}$ and closed under symmetries. Note that from the definition of $\tilde{P}^k_w$ follows that $B$ is a well-defined $d$-tree of depth $k$. The labeling function $L_B$ is defined such that $L_B((v, p)) = L(v)$ for all $(v, p) \in V_B$.

We show that it holds that $N^k(B, (w, w)) = N^k(G, w)$ which directly implies that $N(B, (w, w)) = N(G, w)$. We do this by showing the following stronger statement for all $j = 0, \ldots, k$ via induction: for all $(v, p) \in \tilde{P}^n_w(k - i)$ with $k \geq i \geq j$ holds that $N^j(B, (v, p)) = N^j(G, v)$. The case $j = 0$ is obvious as $L_B$ is defined equivalent to $L$. Therefore assume that the statement holds for $j \leq k - 1$ and all $(v, p) \in \tilde{P}^n_w(k - i)$ with $k \geq i \geq j$. Consider the case $j + 1$ and let $(v, p) \in \tilde{P}^n_w(k - i)$ for some $k \geq i \geq j$. We argue that for each $(v', p') \in N_{v, p}$ it follows that $v' \in N_w$ such that $N^j(B, (v', p')) = N^j(G, v')$ and vice-versa. Let $(v', p') \in N_{v, p}$. By definition of $E_B$, $i \geq 1$ and the fact that all $p \in \tilde{P}^n_w$ are straight follows that $p' = pv'$ or $p = p'v$ but not both. Therefore, either $(v', p') \in \tilde{P}^n_w(k - (i - 1))$ or $(v', p') \in \tilde{P}^n_w(k - (i + 1))$ and, thus, $(v', v) \in E$ or $(v, v') \in E$ which in both cases means $v' = N_w$. The induction hypothesis implies that $N^j(B, (v', p')) = N^j(G, v')$. As these arguments hold for all $(v', p') \in N_{v, p}$ we get that $\sum_{N_{v, p}} N^j(B, (v', p')) \leq \sum_{N_w} N^j(G, v')$. The vice-versa direction is argued analogously which then implies that $\sum_{N_{v, p}} N^j(B, (v', p')) = \sum_{N_w} N^j(G, v')$. From the induction hypothesis we get that $N^j(B, (v, p)) = N^j(G, v)$. Therefore, the definition of a GNN layer implies that $N^{j+1}(B, (v, p)) = N^{j+1}(G, v)$. Therefore, the overall statement holds for all $j \leq k$ and by taking $j = i = k$ we get the desired result.

