Chapter 4

Simplified Inclusion of Relational Information using Vertex-Enrichment*

A key limitation of DRMs is that they require "flattening" any relational information–both in the data and in domain-knowledge–by propositionalisation. As we saw in the previous chapter, the performance of DRMs can depend quite crucially on the relational features used as propositions. If the relational feature does not contain variable co-references, for example, then there is no way of introducing this once the feature has been propositionalised. In this chapter, we move to a form of DNN that is capable of directly manipulating relational data; and introduce a first attempt at the inclusion of relational domain-knowledge into these DNNs.

Recent advances in the area of deep learning have seen a surge in research on learning from graph-structured data, including techniques for learning deep graph embeddings, generalisations of convolutions in CNNs to graphs, and adopting message-passing mechanisms for graph representation learning. These advances have led to new results in several scientific problems, including the problems dealt with in this dissertation. In general, in this chapter, we will be concerned with deep neural networks for graph-structured data, which are known as Graph-based Neural Networks or, simply, Graph Neural Networks (GNNs).

Although GNNs have been popularised recently (2017 and onwards), GNN-like models are not new. These kinds of models were first proposed in [SS97, BPZ97], and later, in [GMS05, SGT+08] the term "Graph Neural Network" or "GNN" was proposed, which referred to the presently prevailing approach of recursive aggregation of information in a graph. The major boost to the field of GNNs followed the introduction of graph

*The content of this chapter is based on the following:
T. Dash, A. Srinivasan, L. Vig, "Incorporating symbolic domain knowledge into graph neural networks", *Machine Learning*, 2021; https://doi.org/10.1007/s10994-021-05966-z.

convolution [KW17] and the notion of a graph embedding [CWPZ18, ZYZZ18]. The last few years have witnessed numerous advancements in the field of GNNs, primarily in the area of their implementations and applications. Two methodical and comprehensive surveys on GNNs are available in: [ZCH+20] and [WPC+20].

Many of the developments in GNN methodologies are powerful and successful in different real-world applications. However, there has been little or no progress in incorporating domain-knowledge into GNNs, and the focus is mainly on learning GNNs from relational data, in particular, graphs alone. Instead, it is assumed that the iterative propagation of messages (information) from one part of a graph to another via the GNNs' aggregation-and-combine mechanism would result in finding out these domain-concepts automatically, albeit inexplicable to humans [BHB+18]. We believe that incorporating domain-concepts into GNNs in some principled manner would allow constructing powerful predictive models. Further, to the best of our knowledge, GNN applications to date have been restricted to simple node-and-edge features, and have not attempted to encode any significant domain-knowledge. In this chapter, we investigate a simplified method of incorporating symbolic domain-knowledge while learning GNNs from relational data. In particular, our proposal in this chapter is to enrich graphs with domain-knowledge via a technique we call "vertex enrichment". We assess the use of domain-knowledge in this manner using the relational datasets and background knowledge studied in the previous chapter. Overall, the principal contributions of the chapter are as follows: (1) To the field of graph neural networks, this chapter presents a large-scale empirical study using real-world datasets on the inclusion of domain-knowledge. To the best of our knowledge, the number of graphs used and the number of relations encoding domain-knowledge are the most extensive to date; (2) To the field of neuro-symbolic modelling, the technique of vertex-enrichment described in this chapter provides a simple but an effective way of incorporating symbolic relations into graph-based neural networks.

4.1 Graph Neural Networks (GNNs)

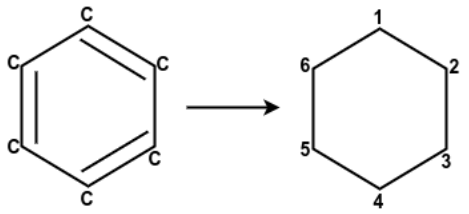
GNNs are primarily developed for learning from data represented as graphs. In this dissertation, our focus will be on GNNs concerned with graph classification. For completeness and clarity of presentation, we provide below the basics of graphs, such as directed and undirected graphs, the role of a neighbourhood function in a graph, and labelled graphs. Further, our examples in this chapter will be about molecular graphs; however, this is just for the convenience of explanation, and our proposed technique is applicable to many other scientific areas, in general.

Definition 4.1 (Graphs)**.** *A graph G is a pair (V, E) where V is a set of vertices, E is a set of edges and a subset of $V \times V$. A graph is said to be undirected if for every*

$(v_i, v_j) \in E$, *we have* $(v_j, v_i) \in E$.

In this chapter, we will be concerned with undirected graphs. We note that for such graphs, E can be represented more compactly as a set consisting of 1- or 2-element subsets of V. We will return to this later, as we extend the consideration to hypergraphs. For molecular graphs, of the kind considered here, self-loops do not occur.

Example 4.1 (Molecules as graphs)**.** *A benzene ring (shown below) can be represented as a graph, in which vertices correspond to atoms and edges correspond to bonds [MW+97].*



The graph-representation of the molecule on the left is (V, E) *where*

$$V = \{1, 2, 3, 4, 5, 6\},$$
$$E = \{(1, 2), (2, 1), (2, 3), (3, 2), (3, 4), (4, 3), (4, 5), (5, 4), (5, 6), (6, 5), (6, 1), (1, 6)\}.$$

We will need the concept of the *neighbourhood* of a vertex in an undirected graph. In this chapter, by "graph" we will mean an undirected graph.

Definition 4.2 (Neighbourhood)**.** *Given a graph* $G = (V, E)$*, a neighbourhood function* σ *is a is a map from* V *to* 2^V *defined as* $\sigma(v) = \{v_i \in V : (v, v_i) \in E\}$.

Example 4.2. *For the graph in Example 4.1, we get* $\sigma(1) = \{2, 6\}, \sigma(2) = \{1, 3\}, \ldots,$ $\sigma(6) = \{1, 5\}$.

GNNs are concerned with labelled undirected graphs, that is, each vertex (and each edge) of a graph is associated with a labelling, which refers to some properties associated with that that vertex or that edge. In GNN terminology, a vertex-label or an edge-label is called a "message".

Definition 4.3 (Graph Labellings)**.** *Let* $\mathcal{V}$ *be a set of vertex labels and* $\mathcal{E}$ *be a set of edge labels. Then a vertex-labelling of a graph* $G = (V, E)$ *is a function* $\psi : V \to 2^{\mathcal{V}}$ *and an edge-labelling is a function* $\epsilon : E \to 2^{\mathcal{E}}$*. We will denote a labelled graph by the tuple* $(V, E, \sigma, \psi, \epsilon)$.

In the definition above, we do not commit to any specific data structure that should be used to implement the label set. This could be, for example, a Boolean-valued array of size $|\mathcal{V}|$.

Example 4.3. *The vertex labels of the graph given in Example 4.1 can be the atom-types (Carbon, C), and edge labels can be the bond-types (single bond: 1, double bond: 2). The label for the vertex 1 is* $\psi(1) = \cdots = \psi(6) = \{C\}$. *The labelling for the edges are* $\epsilon((1,2)) = \epsilon((2,1)) = \{2\}$, $\epsilon((2,3)) = \epsilon((3,2)) = \{1\}$ *and so on.*

Although not evident in this example, vertex- and edge-labels can have more than one element (hence the mapping to $2^{\mathcal{V}}$ and $2^{\mathcal{E}}$). This will be necessary later. We will use the term *graph* interchangeably to denote the tuple (V, E) or the tuple $(V, E, \sigma, \psi, \epsilon)$.

The defining property of a GNN is that it uses some form of neural message-passing in which messages (vertex- or edge-labels) are exchanged between vertices of a graph and updated using a neural network [GSR+17]. The message-passing process is conceptually implemented by using a function called *Relabel*. This involves an iterative update of the vertex- (and edge-) labels. The output of the function is a relabelled graph, where the vertex- (and edge-) labels are updated.

Definition 4.4 (Relabel)**.** *Given a graph* $(V, E, \sigma, \psi, \epsilon)$. *Relabel is a function that returns a graph* $(V, E, \sigma, \psi', \epsilon')$, *where the functions* ψ' *and* ϵ' *may be different to* ψ *and* ϵ.

Since we are interested in classifying graphs, that is, given a set of class labels $\mathcal{Y}$, we want to construct a function that maps a graph of the form $(V, E, \sigma, \psi, \epsilon)$ to a class-label in $\mathcal{Y}$. This requires a graph-level representation (also called a graph-embedding [Ham20]), meaning, every labelled graph is encoded as a d-dimensional real-valued feature vector. This is conceptually implemented using a function called Vec that vectorises a relabelled graph. This feature vector is then input to a (multi-layered) neural network, denoted by a function NN that maps the feature vector to a set of class-labels. The whole pipeline of graph classification is shown in Figure 4.1. Many GNN implementations, including the ones used in this dissertation, assume the graph to be undirected and ignore the edge-labelling in ϵ.

Definition 4.5 (Vec)**.** *Let* $\mathcal{G}$ *denote the set of graph-tuples of the form* $(V, E, \sigma, \psi, \epsilon)$. *For* $d \geq 1$ *and a function* $Vec : \mathcal{G} \to \mathbb{R}^d$, *a vectorisation of the graph is* $Vec((V, E, \sigma, \psi, \epsilon))$.

Definition 4.6 (GNN)**.** *Let* $NN : \mathbb{R}^d \to \mathcal{Y}$ *denote a neural network that maps a real-valued vector to a set of class-labels. Given a* $G = (V, E, \sigma, \psi, \epsilon)$, $GNN(G) = NN(Vec(Relabel(G)))$.

4.1.1 General working principle of GNNs

Let $G = (V, E, \sigma, \psi, \epsilon)$ be a labelled graph as described above. Let X_v denote a vector that represents the initial labelling (ψ) of a vertex $v \in V$. That is, X_v is the feature-vector associated with the vertex v. The relabelling function $Relabel : (V, E, \sigma, \psi, \epsilon) \mapsto (V, E, \sigma, \psi', \epsilon)$

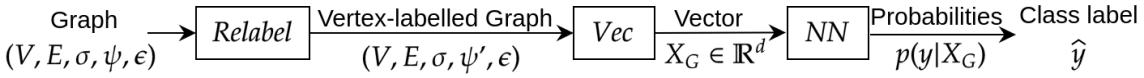


Figure 4.1: A diagrammatic representation of graph classification using a GNN. Graphs are of tuples of the form $(V, E, \sigma, \psi, \epsilon)$, where V is a set of vertices; E is a set of edges; σ is some neighbourhood function; ψ is a vertex-labelling; and ϵ is an edge-labelling. Often σ is left out, and derived from the edges in E.

(iteratively) updates the labelling of the vertices in G. This process involves two procedures: (a) AGGREGATE: for every vertex, this procedure aggregates the information from neighboring vertices; and (b) COMBINE: this procedure updates the label of a vertex by combining its present label with its neighbors', as obtained by AGGREGATE procedure. Mathematically, at some iteration k, the labelling of a vertex v (denoted by h_v) is updated as follows:

$$a_v^{(k)} = \mathsf{AGGREGATE}^{(k)}\left(\left\{h_u^{(k-1)} : u \in \mathcal{N}(v)\right\}\right),$$
$$h_v^{(k)} = \mathsf{COMBINE}^{(k)}\left(h_v^{(k-1)}, a_v^{(k)}\right),$$

where $\mathcal{N}(v)$ denotes the set of vertices adjacent to v. Initially (at $k = 0$), $h_v^{(0)} = X_v$.

The graph vectorisation function $Vec : (V, E, \sigma, \psi', \epsilon) \mapsto \mathbb{R}^d$ constructs a vector representation of the entire graph (also called a graph embedding). This step is carried out after the representations of all the vertices are relabelled by some iterations over AGGREGATE and COMBINE. The vectorised representation of an entire graph, denoted by G here, can be obtained using a READOUT procedure that aggregates vertex features from the final iteration ($k = K$):

$$h_G = \mathsf{READOUT}\left(\left\{h_v^{(K)} \mid v \in G\right\}\right)$$

In practice, AGGREGATE and COMBINE procedures are implemented using graph convolution and pooling operations. The READOUT procedure is usually implemented using a global or hierarchical pooling operation. Variants of GNNs result from modifications to these 3 procedures: AGGREGATE, COMBINE and READOUT. The functional forms of AGGREGATE, COMBINE and READOUT are provided in an appendix section. Below we provide some brief notes on some GNN variants that are implemented in our experiments.

4.1.2 Note on GNN variants

We focus mainly on the following variants of the AGGREGATE-COMBINE procedures:

1. Spectral graph convolution, GCN [KW17]: This is a spectral method for graph convolution that uses convolutional aggregator. This is a simple and well-behaved

layer-wise propagation rule for neural network models which operate directly on graphs.

2. Multistage graph convolution, k-GNN [MRF+19]: This convolution method can perform convolution operations using multiple-sized neighbourhoods (the authors call this "higher order" graph convolution).

3. Graph convolution with attention, GAT [VCC+18]: This is a spatial method of graph convolution that uses an "attention" mechanism, that estimates the importance of vertices in the neighbourhood of a vertex.

4. Sample-and-aggregate graph convolution, GraphSAGE [HYL17]: Here the convolution procedure samples from a distribution that is constructed from feature-vectors of vertices in the neighbourhood of a vertex.

5. Graph convolution with auto-regressive moving average, ARMA [BGLA21]: This is a convolution method that employs a polynomial function of the feature-vectors in the neighbourhood of a vertex.

In our implementations that we will describe later, in addition to the graph convolution methods mentioned above, we use a graph-pooling step that applies down-sampling to graphs. This operation allows to obtain refined graph representations at each layer. Like in convolutional neural networks, a (graph-)pooling operation follows a (graph-)convolution operation. The primary aim of including a graph pooling operation after each graph convolution is that this operation can reduce the graph representation while ideally preserving important structural information. In the research conducted in this dissertation, we use a popular structural-attention based graph pooling method [LLK19].

To implement the READOUT procedure, we use hierarchical graph-pooling method proposed by [CVJ+18]. Therefore, a GNN variant for us will refer to a GNN constructed with one of the above mentioned graph-convolution operators, the graph-pooling operator and the hierarchical operator. A detailed conceptual and mathematical description of these convolution and pooling operators are provided in Appendix A.

4.2 Inclusion of n-ary relations into GNNs by Enriching Vertex-Labels

GNNs, as we have described them so far, deal with node- and edge-labels in an undirected graph, in which edges are sets of vertex-pairs. That is, the edges represent a symmetric binary relation. However, for many real-world problems—including the ones considered in this chapter—we have access to domain-knowledge which relate more than just pairs

of vertices. For example, if a molecule is represented as a graph (with atoms as vertices, and an edge denoting a bond between a pair of vertices), then a benzene-ring is a relation amongst 6 distinct vertices, with some specific constraints on the vertices and edges. Here, we will consider domain-knowledge to be a set of relations, each of which can be expressed as a hypergraph.

Definition 4.7 (Hypergraphs)**.** *A hypergraph H is the pair (V, E'), where V is a set of vertices and E' is a non-empty subset of 2^V. Each element of E' is called a hyperedge.*

Example 4.4. *A hypergraph of the molecular graph given in Example 4.1 can be*

$$H = (\{1,2,3,4,5,6\}, \{\{1,2\}, \{3,4,5,6\}, \{2,4,5\}, \{1,2,3,4,5,6\}\}).$$

We note that since hyperedges are sets, there is no distinction between permutations of vertices in a hyperedge. So, as defined here, we will take hyperedges as being undirected. Hypergraph labellings can be defined similarly as before, using a pair of functions for vertex- and edge-labels. We will reuse the notation ψ and ϵ for these functions, with annotations to clarify what is meant. The neighboorhood relation σ is left unspecified here (one obvious definition is $\sigma(v_i) = \{v_j : h \in E', \{v_i, v_j\} \subseteq h\}$). In this chapter, we are interested in n-ary relations that can be expressed as hypergraphs.

Definition 4.8 (n-ary Relation as a Labelled Hypergraph)**.** *A n-ary relation R defined over vertices of a graph $G = (V, E)$ is a hypergraph $H = (V, E')$, and every hyperedge $h \in E'$ has n elements from V. We will denote this as $R(G) = H$. Let ψ_G denote a vertex-labelling over G and R/n denote the predicate-symbol for R. With some abuse of notation, the vertex-labelling function for $R(G) = H = (V, E')$ is as follows:*

$$\psi_H(v) = \begin{cases} \psi_G(v) \cup \{R/n\} & \text{if } \exists h \in E' s.t.\ v \in h \\ \emptyset & \text{otherwise} \end{cases}$$

and the hyperedge-labelling function is

$$\epsilon_H(h) \ = \ \{R/n\} \quad (h \in E').$$

That is, the vertex-labelling of a vertex v in the hypergraph H is a set containing the existing vertex-label of v in G augmented by the predicate-symbol R/n vertex-label.

Example 4.5. *Consider a relation for a benzene ring:*

$$\begin{aligned} benzene(a_1, a_2, a_3, a_4, a_5, a_6) \leftarrow & \\ & cycle(a_1, a_2, a_3, a_4, a_5, a_6) \wedge \\ & aromatic(a_1, a_2, a_3, a_4, a_5, a_6). \end{aligned}$$

One possible vertex-labelling is

$$\psi_H(1) = \cdots = \psi_H(6) = \{C, benzene/6\}$$

(here, C denotes "carbon"). A hyperedge-labelling may contain:

$$\epsilon_H(\{1, 2, 3, 4, 5, 6\}) = \{benzene/6\}.$$

The extension to multiple relations, not all of the same arity, is straightforward.

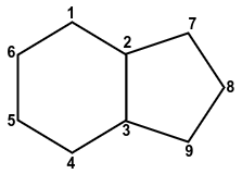
Definition 4.9 (Multiple Relations as a Labelled Hypergraph)**.** *Let $R_1, \ldots, R_k$ be relations defined on vertices of a graph $G = (V, E)$, s.t. $R_i(G) = (V, E_i')$. Then $\bigcup_{i=1}^{k} R_i(G)$ is the hypergraph $H = (V, E')$ where $E' = \bigcup_{i=1}^{k} E_i'$. The corresponding labelling functions are*

$$\psi_H(v) = \bigcup_{i=1}^{k} \psi_{H_i}(v)$$

and

$$\epsilon_H(v) = \bigcup_{i=1}^{k} \epsilon_{H_i}(v).$$

Example 4.6. *Consider the following molecular graph with two relations: benzene/6 and pyrrole/5.*



One possible vertex-labelling for this graph is

$$\begin{aligned} \psi_H(1) = \psi_H(4) = \psi_H(5) = \psi_H(6) &= \{C, benzene/6\} \\ \psi_H(2) = \psi_H(3) &= \{C, benzene/6, pyrrole/5\} \\ \psi_H(7) &= \{N, pyrrole/5\} \\ \psi_H(8) = \psi_H(9) &= \{C, pyrrole/5\} \end{aligned}$$

and a hyperedge-labelling is

$$\epsilon_H(\{1,2,3,4,5,6\}) = \{benzene/6\}$$
$$\epsilon_H(\{2,7,8,9,3\}) = \{pyrrole/5\}.$$

In principle, provided we are able to define a neighbourhood function σ for hypergraphs, the definition of GNNs in Defn. 4.6 does not change. We would however like to use one of the standard GNN implementations described in the previous section, which restricts graphs with 2-vertex edges, and edge-labels to singleton sets. With some loss of information, we extract a suitable graph from a hypergraph.

Definition 4.10 (Vertex-Enriched Graphs)**.** *Let $G = (V, E)$ be a graph, with neighbourhood function σ, vertex-labelling function ψ, and edge-labelling function ϵ. Here, E is a subset of $V \times V$. Let $\mathcal{R} = \{R_1, \ldots, R_k\}$ be a set of relations defined on G, and $\bigcup R_i(G)$ be the hypergraph $H = (V, E')$ with vertex-labelling function ψ' as in Defn. 4.9. Then $G' = (V, E, \sigma, \psi', \epsilon)$ is called a vertex-enriched form of $G = (V, E, \sigma, \psi, \epsilon)$. We denote this by $VE(G, \mathcal{R}) = G'$.*

Example 4.7. *The molecular graph G for Example 4.6 is*

$$G = (\{1,2,3,4,5,6,7,8,9\}, \{(1,2),(2,1),\ldots,(1,6),(6,1),(2,7),(7,2),\ldots,(9,3)(3,9)\}).$$

A vertex-labelling of G is

$$\psi(1) = \cdots = \psi(6) = \psi(8) = \psi(9) = \{C\}$$
$$\psi(7) = \{N\}$$

The vertex-labelling of the vertex-enriched graph G', after the inclusion of the relations in Example 4.6 is:

$$\psi'(1) = \psi'(4) = \psi'(5) = \psi'(6) = \{C, benzene/6\}$$
$$\psi'(2) = \psi'(3) = \{C, benzene/6, pyrrole/5\}$$
$$\psi'(7) = \{N, pyrrole/5\}$$
$$\psi'(8) = \psi'(9) = \{C, pyrrole/5\}.$$

The edge-labelling and neighborhood functions do not change after relation enrichment.

The vertex-enriched graph thus extends the vertex-labelling of a graph G, with the vertex-labels from the hypergraph H obtained from relations $R_1, \ldots, R_k$ defined on G. Procedure 3 provides a set of formal steps to enrich a graph G given a set of relations $\mathcal{R}$.

Procedure 3 requires identification of subgraphs of the original graph. That is, for every relation $R_i \in \mathcal{R}$, the corresponding hyperedge H_i is a subset of vertices $\{v_1, \ldots, v_n\} \in V$, such that $(v_1, \ldots, v_n) \in R_i$. This step requires the identification of all subsets of vertices of the graph constituting hyperedge as above. For a graph (V, E), this can, in the worst case require an examination of $\binom{|V|}{n}$ combinations. Therefore, for arbitrary sized graphs and subgraphs, this is computationally hard. In practice, we will be forced to impose bounds on the size of V_s and on the size of the subgraph.

We note that the process of vertex-enrichment is a simplification of the full relational information available. For example, in the example above, if an atom (represented by a vertex in the molecular graph) is part of more than 1 benzene ring, then its vertex-enrichment will only contain a single entry for *benzene*/6, indicating that it is part of 1 or more benzene rings. We discuss these limitations in detail later.

Procedure 3 Vertex-Enrichment of a graph G, given a set of relations $\mathcal{R}$. The new label of a vertex includes all the relations of which the vertex is part. The procedure takes as input a graph G, a set of relations $\mathcal{R}$, and returns a vertex-enriched graph G'.

1: **procedure** EnrichGraph($G = (V, E, \sigma, \psi, \epsilon)$, $\mathcal{R} = \{R_1, \ldots, R_k\}$)
2: Let $\psi' := \psi$
3: **for all** $R_i \in \mathcal{R}$ **do**
4: Let $R_i \subseteq V^n$
5: $\mathcal{H}_i = \{\{v_1, \ldots, v_n\} : (v_1, \ldots, v_n) \in R_i\}$
6: Let $V_s = \bigcup_{H_j \in \mathcal{H}_i} H_j$
7: **for all** $v_j \in V_s$ **do**
8: $\psi'(v_j) := \psi'(v_j) \cup \{R_i/n\}$
9: **return** $G' = (V, E, \sigma, \psi', \epsilon)$

The vertex-enriched graph obtained here are immediately suitable for the GNN implementations we consider in this chapter: The vertex-labels of a graph are sets and a GNN implementation cannot handle sets. We provide a simple technique to transform these graphs into a form suitable for GNN implementations.

4.2.1 Vertex-Enriched GNNs

For use by a GNN, a vertex-label in a labelled graph needs to be transformed to a fixed-length feature-vector. In our implementations of GNN variants, each vertex-label is transformed to a fixed-length multi-hot Boolean-valued vector associated with the vertex. We define a function $Vectorise$ to do this transformation.

Definition 4.11 (Vectorisation)**.** *Let $\mathcal{G}$ be a set of vertex-enriched graphs, where each graph G' is of the form $(V, E, \sigma, \psi', \epsilon)$. Assume a set of domain relations, denoted by $\mathcal{R}$, available in background knowledge B. Let $\mathcal{R}$ consist of k relations: $R_1, \ldots, R_k$. We define*

a label-vectorisation function $\psi'' : V \times \{1, \ldots, k\} \to \{0, 1\}$ *as*

$$\psi''(v, i) = \begin{cases} 1 & \text{if } R_i \in \psi'(v) \\ 0 & \text{otherwise} \end{cases}$$

Thus, the vectorisation of the label of a vertex $v \in V$ *is* $(\psi''(v, 1), \ldots, \psi''(v, k))$*. Then, the graph* $G'' = (V, E, \sigma, \psi'', \epsilon)$ *is the vectorised form of* G' *and is denoted as* $G'' = Vectorise(G', \mathcal{R})$*.*

Definition 4.12 (Vertex-Enriched GNN)**.** *Let* $G = (V, E, \sigma, \psi, \epsilon)$*, and* $Vectorise$*,* $Relabel$*,* Vec *and* NN *be as before. Then, a Vertex-enriched GNN is*

$$VEGNN(G) = NN(Vec(Relabel(Vectorise(VE(G, \mathcal{R}), \mathcal{R})))).$$

Example 4.8. *Let's consider the example of a molecular graph* G *given given in Example 4.6 and its corresponding vertex-enriched graph* G' *in Example 4.7. For this example, let's assume* $\mathcal{R} = \{C, N, O, benzene/6, furan/5, pyrrole/5\}$*. The vectorised form of* G' *will have vertices associated with the following vectors of length* 6*:*

v	$\psi''(v)^\top$
1	$[1, 0, 0, 1, 0, 0]$
2	$[1, 0, 0, 1, 0, 1]$
3	$[1, 0, 0, 1, 0, 1]$
4	$[1, 0, 0, 1, 0, 0]$
5	$[1, 0, 0, 1, 0, 0]$
6	$[1, 0, 0, 1, 0, 0]$
7	$[0, 1, 0, 0, 0, 1]$
8	$[1, 0, 0, 0, 0, 1]$
9	$[1, 0, 0, 0, 0, 1]$

Notice that the atom types such as carbon, nitrogen, oxygen, denoted by C*,* N*,* O*, respectively, are treated as relations with arity* 0*.*

Procedure 4 and Procedure 5 are two simple procedures to construct and evaluate VEGNNs. These procedures assumes two sub-procedures: TrainGNN, that trains a standard GNN using a set of labelled graph-instances; and Evaluate, that computes the predictive performance of a trained model.

Procedure 4 Procedure to construct a VEGNN model. The procedure takes as inputs: a set of of labelled data-instances $D_{Tr} = \{(g_1, y_1), \ldots, (g_N, y_N)\}$, where g_i is the graph-representation of a relational data-instance and y_i is a class-label associated with g_i; a set of domain-relations $\mathcal{R}$; a set of hyperparameters P; and returns a VEGNN model.

1: **procedure** TRAINVEGNN($D_{Tr}, \mathcal{R}$)
2: $D'_{Tr} = \{(g'_i, y_i) : (g_i, y_i) \in D_{Tr}$ and $g'_i = VE(g_i, \mathcal{R}\}$
3: $D''_{Tr} = \{(g''_i, y_i) : (g'_i, y_i) \in D'_{Tr}$ and $g''_i = Vectorise(g'_i, \mathcal{R})\}$
4: Let $VEGNN =$ TRAINGNN(D''_{Tr}, P)
5: **return** $VEGNN$

Procedure 5 Procedure to test a trained VEGNN model. The procedure takes as inputs: a trained model $VEGNN$; a set of of labelled data-instances $D_{Te} = \{(g_1, y_1), \ldots, (g_M, y_M)\}$; a set of domain-relations $\mathcal{R}$; and returns the predictive performance of $VEGNN$.

1: **procedure** TESTVEGNN($D_{Te}, \mathcal{R}$, P)
2: $D'_{Te} = \{(g'_i, y_i) : (g_i, y_i) \in D_{Te}$ and $g'_i = VE(g_i, \mathcal{R}\}$
3: $D''_{Te} = \{(g''_i, y_i) : (g'_i, y_i) \in D'_{Te}$ and $g''_i = Vectorise(g'_i, \mathcal{R})\}$
4: Let $\hat{\mathbf{y}} = \{\hat{y}_i = VEGNN(g''_i) : (g''_i, \cdot) \in D''_{Te}\}$
5: Let $perf =$ EVALUATE($\mathbf{y}, \hat{\mathbf{y}}$) ▷ where $\mathbf{y} = \{y_i : (g''_i, y_i) \in D''_{te}\}$
6: **retun** $perf$

4.3 Empirical Evaluation

4.3.1 Aims

The aim of this empirical study is to evaluate the performance of VEGNNs that includes domain-knowledge using the vertex-enrichment technique. That is,

- We investigate whether the performance of a VEGNN that includes domain-knowledge using vertex-enrichment is better than the performance of a GNN that does not include domain-knowledge.

4.3.2 Materials

The datasets and background knowledge used here are the same 73 classification problems used in the previous chapter (see section 3.7.2 and section 3.7.2). Here we only describe changes in Materials specific to the experiments in this chapter.

Algorithms and Machines

The data and background knowledge are written in Prolog. A Prolog program is used to extract the set of vertices for which a domain-relation is true. We use YAP compiler for execution of this logic program. All the deep neural network experiments are conducted in a Python environment. The GNN models are implemented by using the PyTorch

Geometric library [FL19], which is a popular geometric deep learning extension for PyTorch [PGM+19] and it provides graph pre-processing routines and makes the definition of graph convolution easier to implement.

For all the experiments, we use a machine with Ubuntu (16.04 LTS) operating system, and hardware configuration such as: 64GB of main memory, 16-core Intel Xeon processor, a NVIDIA P4000 graphics processor with 8GB of video memory.

4.3.3 Method

In all experiments, we refer to GNN variants as $GNN_{1,\ldots,5}$ and the corresponding vertex-enriched versions are $VEGNN_{1,\ldots,5}$. For the methodology outlined below, let D be a set of labelled data-instances represented as graphs: $\{(g_1, y_1), \ldots, (g_N, y_N)\}$ where y_i is a class-label associated with the graph g_i. For constructing the VEGNNs, we assume that we have access to: (a) a set of domain relations $\mathcal{R}$, (b) implementations of the GNN variants, and (c) the procedures TrainVEGNN and TestVEGNN as described in Procedure 4 and Procedure 5, respectively. Our method for investigating the performance of $VEGNN$s is straightforward:

(1) Randomly split D into D_{Tr} (train-set) and D_{Te} (test-set);

(2) Construct a GNN model using D_{Tr} (GNN without the domain-relations in $\mathcal{R}$). Let GNN_i be the resulting GNN model, where $i = 1, \ldots, 5$ refers the GNN variant;

(3) Construct a VEGNN model using D_{Tr} (GNN with the domain-relations in $\mathcal{R}$). Let $VEGNN_i$ be the resulting VEGNN model;

(4) Obtain the predictive performance of the GNN model constructed in Step (2) on D_{Te};

(5) Obtain the predictive performance of the VEGNN model constructed in Step (3) on D_{Te};

(6) Compare the predictive performance of $VEGNN_i$ against that of GNN_i ($i = 1, \ldots, 5$).

The following additional details are relevant:

- We have used a 70:30 train-test split for each of the datasets. 10% of the train-set is used as a validation set for hyperparameter tuning.
- The relations in $\mathcal{R}$ are those described in section 3.7.2.

- The general workflow involved in GNNs is described in section 4.1. A diagram of the components involved in implementing that workflow for constructing a VEGNN is shown in Figure 4.2. As shown in the figure, a GNN in our implementations consists of three graph convolution blocks and three graph pooling blocks. The convolution and pooling blocks interleave each other (that is, C-P-C-P-C-P).

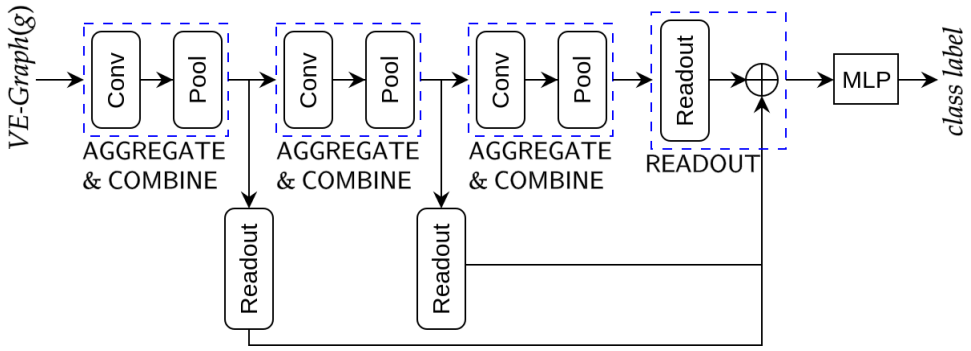


Figure 4.2: Components involved in implementing the workflow in section 4.1 for VEGNN models. The input is the vectorised representation of a vertex-enriched graph, denoted here as $VE\text{-}Graph(g)$ for an graph data-instance g. The blocks 'Conv' and 'Pool' refer to the graph-convolution and graph-pooling operations, respectively. The 'Readout' operation constructs a graph representation by accumulating information from all the vertex in the graph obtained after the pooling operation. The final graph representation is obtained in the READOUT block by an element-wise sum (shown as $\oplus$) of the individual graph-representations obtained after each AGGREGATE-COMBINE block. MLP stands for Multilayer Perceptron.

- The convolution blocks can be of one of the five variants, discussed earlier in subsection 4.1.2.
- The graph pooling block uses self-attention pooling [LLK19] with a pooling ratio of 0.5. We use the graph-convolution formula proposed in [KW17] for calculating the self-attention scores used in the self-attention pooling.
- Due to the large number of experiments (resulting from multiple datasets and multiple GNN variants), the hyperparameters in the convolution blocks are set to the default values within the PyTorch Geometric library [FL19].
- We use a hierarchical pooling architecture that uses the readout mechanism proposed by Cangea *et al.* [CVJ+18]. The readout block aggregates node features to produce a fixed size intermediate representation for the graph. The final fixed-size representation for the graph is obtained by element-wise addition of the three readout representations.
- The representation length ($2m$) is determined by using a validation-based approach. The parameter grid for m is: $\{8, 128\}$, representing a small and a large embedding, respectively.

- The final representation is then fed as input to a 3-layered MLP. We use a dropout layer with a fixed dropout rate of 0.5 after the first layer of MLP.

- The input layer of the MLP contains $2m$ units, followed by two hidden layers with m units and $\lfloor m/2 \rfloor$ units, respectively. The activation function used in the hidden layers is `relu`. The output layer uses `logsoftmax` activation.

- The loss function used is the negative log-likelihood between the target class-labels and the predictions from the model.

- We denote the $VEGNN$ variants as: $VEGNN_{1,\ldots,5}$ based on the type of graph convolution method used.

- We use the Adam optimiser [KB15] for training the VEGNNs ($VEGNN_{1,\ldots,5}$). The learning rate is 0.0005, weight decay parameter is 0.0001, the momentum factors are set to the default values of $(\beta_1, \beta_2) = (0.9, 0.999)$.

- The maximum number of training epochs is 1000. The batch size is 128.

- We use an early-stopping mechanism [Pre98] to avoid overfitting during training. The resulting model is then saved and can be used for evaluation on the independent test-set (D_{Te}). The patience period for early-stopping is fixed at 50.

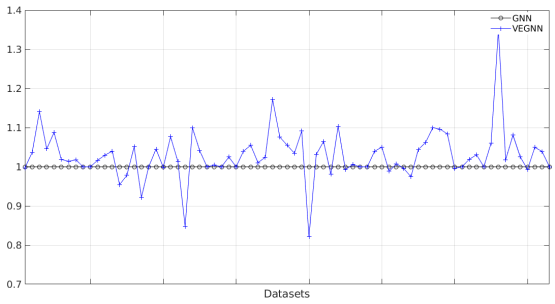
- The predictive performance of a VEGNN model refers to its predictive accuracy on the independent test-set.

- Comparison of performance is done using the Wilcoxon signed-rank test, using the standard implementation within MATLAB (R218b).

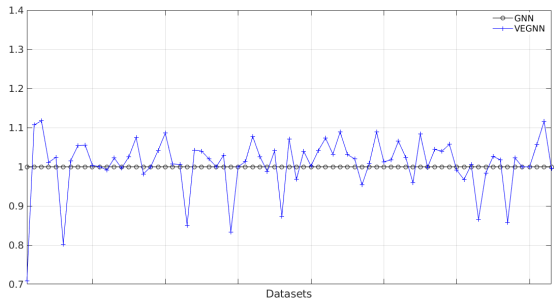
4.3.4 Results

The main results from the experiments are shown qualitatively in Figure 4.3. The principal finding from the tabulations is that inclusion of domain-knowledge into GNNs (that is, the use of vertex-enriched GNNs) results in an improvement in predictive accuracy for all variants of GNN.

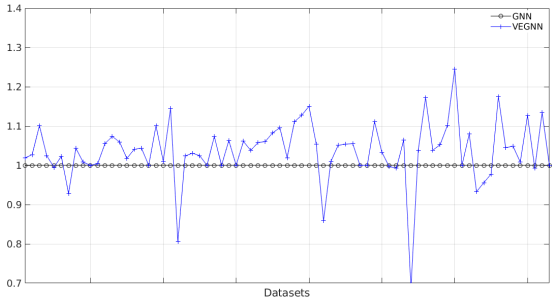
We examine the results in more detail: From Figure 4.3, it is evident that the performance of graph-based networks improves with the inclusion of domain-knowledge. A quantitative tabulation of wins, losses and draws is in Figure 4.4. These results again provide sufficient grounds to answer positively the primary research question addressed in this dissertation, namely: do DNNs (here, represented by GNNs) benefit from the inclusion of domain-knowledge?



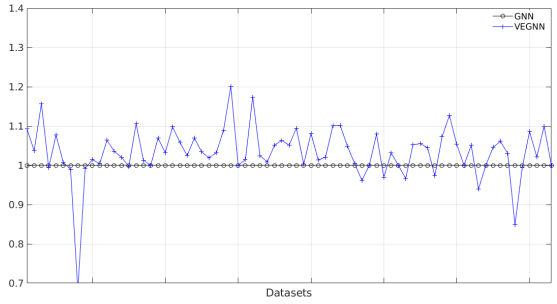
(a) GNN_1 (median gain $\approx 3\%$)



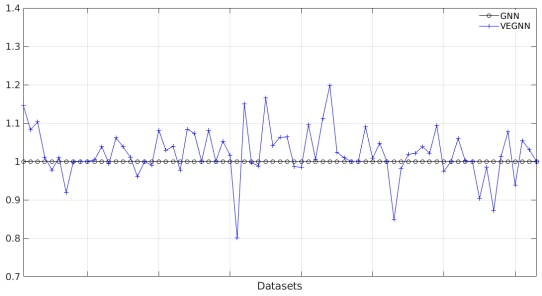
(b) GNN_2 (median gain $\approx 3\%$)



(c) GNN_3 (median gain $\approx 4\%$)



(d) GNN_4 (median gain $\approx 3\%$)



(e) GNN_5 (median gain $\approx 2\%$)

Figure 4.3: Qualitative comparison of predictive performance of VEGNNs against Baseline (that is, GNN variants without access to domain-relations). Performance refers to estimates of predictive accuracy (obtained on a holdout set), and all performances are normalised against that of baseline performance (taken as 1). No significance should be attached to the line joining the data points: this is only for visual clarity.

Some Additional Comparisons

Although not of direct relevance to the primary research conjecture of this dissertation (that is, the inclusion of domain-knowledge can significantly improve the performance of DNNs), it is nevertheless useful to ask how the two approaches we have investigated so far compare against each other. To answer this question, we perform a quantitative comparison between VEGNNs and Deep Relational Machines (DRMs) constructed using propositionalisation of relational features sampled using our proposed hide-and-seek sampling strategy in Chapter 3. We provide a tabulation of this comparison in Figure 4.5. The finding suggests that DRMs can perform at the same or higher level of predictive performance as VEGNNs. At the outset, it may seem easy to conclude that simple machinery like DRMs are better than VEGNNs for the inclusion of domain-knowledge, we

GNN Variant	Accuracy ($VEGNN$ vs. GNN) Higher/Lower/Equal (p-value)
GNN_1	48/14/11 (< 0.001)
GNN_2	48/19/6 (0.005)
GNN_3	53/11/9 (< 0.001)
GNN_4	54/12/7 (< 0.001)
GNN_5	43/19/11 (0.002)

Figure 4.4: Quantitative comparison of predictive performance of $VEGNN$s against GNNs. Here GNN refers to the graph-based neural network without domain-knowledge, and $VEGNN$ refers to the network vertex-enriched with the generic domain-knowledge described in section 3.7.2. The tabulations are the number of datasets on which $VEGNN$ has higher, lower or equal predictive accuracy on a holdout-set. Statistical significance is assessed by the Wilcoxon signed-rank test.

highlight some key issues that are not apparent from these tabulations: (1) DRMs need to be provided with a sufficient number of relational features (here, 250) to match the same level of performance as a VEGNN; (2) DRMs need to be provided with an expressive set of relational features to reach the same level of performance as a VEGNN; (3) For a DRM, there is significant computational effort is required to draw these 250 features using sampling. As discussed in Chapter 3, the sampling procedure incurs a huge computational cost to select a set of 250 features where selecting just one relational feature requires: sampling a lot more than one feature, evaluating them for their utilities, and discarding the features with bad utilities. Whereas VEGNNs do not involve any such sampling step and therefore the computational cost remains relatively minimal.

We further compare VEGNNs against another propositionalisation based technique for inclusion of domain-knowledge, called BCP [FZG14] that we investigated in Chapter 3. A comparison of VEGNNs against MLPs constructed with BCP features are provided in Figure 4.6. The results here reaffirm that though propositionalisation based techniques are simple, and they require significant computational overhead to perform well: In this case, the number of input BCP features for an MLP ranges from 18000 to 52000 (Refer section 3.7 for more details). This kind of number leads to more complex MLPs that require huge training effort.

4.3.5 Limitations of VEGNNs

So far we have studied how vertex-enrichment allows relations in the background knowledge to be incorporated into a graph neural network. Although this approach is effective in terms of improving the predictive performance of GNNs, it comes with a very basic limitation. Let us examine the following molecule (a graph data-instance) with 2 fused benzene rings and its corresponding vertex-enriched molecular graph in Figure 4.7. In

GNN Variant	Accuracy ($VEGNN$ vs. DRM) Higher/Lower/Equal (p-value)		
	$\lvert\mathcal{R}'\rvert = 50$	$\lvert\mathcal{R}'\rvert = 100$	$\lvert\mathcal{R}'\rvert = 250$
GNN_1	59/13/1 (< 0.001)	50/22/1 (< 0.001)	21/52/0 (< 0.001)
GNN_2	49/23/1 (< 0.01)	39/33/1 (0.81)	19/54/0 (< 0.001)
GNN_3	54/18/1 (< 0.001)	44/28/1 (0.05)	14/59/0 (< 0.001)
GNN_4	59/13/1 (< 0.001)	52/20/1 (< 0.001)	23/50/0 (< 0.001)
GNN_5	53/19/1 (< 0.001)	42/30/1 (0.06)	17/56/0 (< 0.001)

Figure 4.5: Quantitative comparison of predictive performance of VEGNNs against DRMs. Here $VEGNN$ denotes the vertex-enriched GNN with $\mathcal{R}$, and DRM denotes the Deep Relational Machine constructed using propositionalisation of relational features. The relational features for a DRM are sampled using the hide-and-seek sampling strategy proposed in Chapter 3. The set of the hide-and-seek features is denoted by $\mathcal{R}'$. The comparative performance of VEGNNs against DRMs starts worsening after $|\mathcal{R}'| = 500$, which are not shown here. The tabulations are the number of datasets on which $VEGNN$ has higher, lower or equal predictive accuracy on a holdout-set. Statistical significance is assessed by the Wilcoxon signed-rank test.

GNN Variant	Accuracy ($VEGNN$ vs. BCP+MLP) Higher/Lower/Equal (p-value)
GNN_1	51/21/1 (< 0.001)
GNN_2	46/26/1 (0.08)
GNN_3	48/24/1 (0.003)
GNN_4	54/18/1 (< 0.001)
GNN_5	47/25/1 (0.005)

Figure 4.6: Quantitative comparison of predictive performance of $VEGNN$s against that of MLPs constructed using BCP features [FZG14]. The tabulations are the number of datasets on which $VEGNN$ has higher, lower or equal predictive accuracy on a holdout-set. Statistical significance is assessed by the Wilcoxon signed-rank test.

the graph it is clear that vertices v_4 and v_5 are members of two different benzene rings, however the vertex-enrichment is not able to capture this fact automatically. That is, given the vertex-labels alone we may not be able to deduce that these two vertices are members of two different benzene rings.

The limitation of vertex-enrichment stated above is more apparent in the following molecular graph (highlighting only the vertex-labels for two vertices) in Figure 4.8. The vertex v_4 is a member of three different benzene rings and as a result two different fused rings, which has not been captured by the proposed vertex-enrichment technique. We could, therefore, say that vertex-enrichment is a simplification method for the inclusion of domain-knowledge into GNNs. Here we prefer the word "simplification" instead of

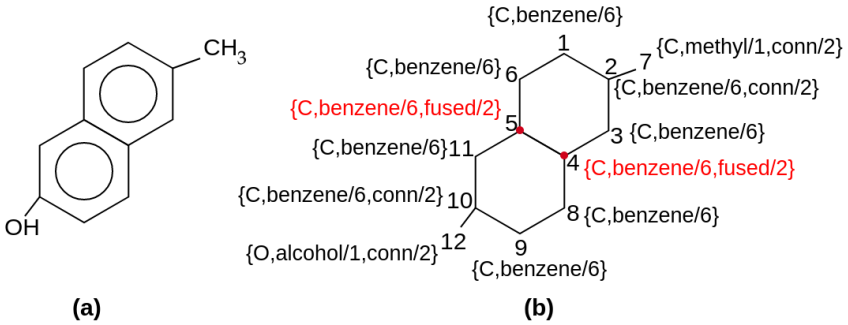


Figure 4.7: Figure showing (a) a molecule with 2 fused benzene rings, (b) its corresponding molecular graph with vertices enriched with domain-relations.

"approximation" for an obvious reason that multiple occurrences of any n-ary domain relation for a vertex in a graph is simplified and shown as a single occurrence.

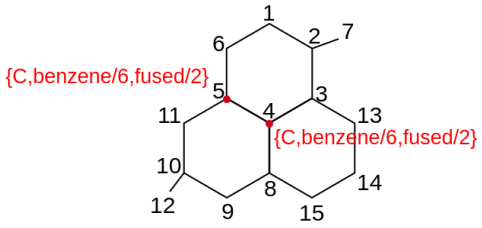


Figure 4.8: Figure highlighting a limitation of the vertex-enrichment technique for a molecular graph.

4.4 Summary

In this chapter, we proposed a simplified technique, called vertex-enrichment, for inclusion of domain-knowledge into graph-based neural networks (or GNNs). The resulting GNNs are called Vertex-Enriched GNNs or VEGNNs. We performed a large-scale evaluation of VEGNNs and compared their predictive performance against GNNs that do not include domain-knowledge. The primary results here clearly show the benefit of having mechanisms to incorporate domain-knowledge into GNNs. To the best of our knowledge, the experiments in this chapter constitute some of the most extensive applications of GNNs to large-scale real-world scientific data arising in the domain of drug-discovery. Further, we studied that vertex-enrichment, being a simplification technique to incorporate all the domain-relations available in the background knowledge, may limit the expressiveness of a VEGNNs.