Graph-Based Wave Function Collapse for Data Augmentation in Graph Classification

An Experimental Evaluation Using a Graph Convolutional Network

Bachelor Thesis
Faculty of Science, University of Bern

submitted by

Marco Brubacher

from Balerna, Switzerland

Supervision:

PD Dr. Kaspar Riesen
Institute of Computer Science (INF)
University of Bern, Switzerland

Abstract

This thesis proposes an adaptation of the Wave Function Collapse (WFC) algorithm to generate synthetic vertex-labelled, undirected graphs and evaluates its application as a data augmentation method for graph classification. The introduced algorithm extracts from a given input graph a multiset of rooted radius-r neighbourhood patterns and derives pattern-compatibility constraints via pairwise comparisons. These constraints guide the generation of a synthetic graph that preserves the input graph's local neighbourhood structure while allowing global variation. The method is incorporated into a graph classification pipeline for augmenting training sets. A configurable number of training graphs are duplicated. One synthetic graph is generated for each duplicated instance, and the resulting synthetic graphs are added to the original training set before model training. The resulting augmented training sets are evaluated on three TUDataset benchmarks (MSRC_21, PROTEINS and DD) using a fixed graph convolutional network implemented in PyTorch Geometric. For each dataset, multiple configurations are evaluated, defined by the pattern radius r, the target size range of the synthesised graphs, and the number of synthetic graphs added to the training set. Performance on the held-out test set is reported as test accuracy, macro-F1, and balanced accuracy, relative to a baseline trained on the original (non-augmented) training set. The best dataset-specific configuration improves test accuracy by +2.59 percentage points (pp) on MSRC_21 and by +3.54 pp on DD. On PROTEINS, the best configuration yields a smaller gain of +1.73 pp, while several configurations reduce performance. Overall, the results indicate that the proposed WFC-based training set augmentation can be practical, but its benefits differ between datasets and configurations.

Acknowledgements

I would like to express my sincere gratitude to PD Dr. Kaspar Riesen for his guidance throughout the entire project. I am also grateful to the University of Bern and, in particular, the Pattern Recognition Group for providing the opportunity to conduct this work. Finally, I would like to thank my family and friends for their encouragement over the past months.

Contents

Chapter 1

Introduction

1.1 Motivation and Problem

A recurring practical challenge in supervised graph classification is limited dataset size and class imbalance in graph classification benchmarks [1, 2]. For instance, the D&D (DD) protein dataset in the TUDataset collection consists of 1178 proteins represented as graphs, with 691 enzymes and 487 non-enzymes [3, 4]. Such imbalanced class distributions can pose significant challenges for learning algorithms and bias trained classifiers toward the majority class [2]. When class-specific substructures are rare, models may fail to learn them reliably, particularly under imbalance [5]. Classifiers that aggregate information from local neighbourhoods, such as Graph Convolutional Networks (GCNs) [6], can be especially sensitive in this setting, since under class imbalance, they may overfit majority-class patterns while underrepresenting minority-class signals [5].

To mitigate these limitations, graph data augmentation is applied to increase data diversity and improve robustness in graph neural network training [7]. A typical augmentation technique perturbs the original graph, for example, by randomly removing or masking vertices or edges [7]. Such local perturbations can improve robustness but provide limited control over the resulting structures and may alter potentially informative k-hop neighbourhoods [7]. This thesis examines a complementary approach: graph-level augmentation that synthesises new graphs by adapting the Wave Function Collapse (WFC) algorithm from grid-based images to graph-structured data, to preserve local structural patterns and graph statistics from the training data.

1.2 Wave Function Collapse

Wave Function Collapse (WFC), proposed by Maxim Gumin, is a constraint-based generation algorithm that synthesises new images from local patterns learned from an example input [8]. Intuitively, WFC scans the example to collect small local patterns, estimates how frequently they occur, and extracts adjacency constraints from them. New images are then generated by redrawing the content on a predefined output grid while respecting these learned adjacency constraints, so that every small local region in the output resem-

bles some region observed in the input [9]. In this way, WFC reproduces local patterns (motifs) and their frequency distribution while still allowing global variability.

Since its introduction, several variants of WFC have been explored for different forms of content, including tile-based game levels [9], 3D voxel-based models [10], and implementations on hexagonal grids [11]. In this thesis, the overlapping WFC model is taken as the starting point and reformulated to operate on graph-structured data rather than regular grids. This reformulated variant is referred to throughout this work as the *Graph WFC* algorithm for simplicity.

1.3 Graph WFC

The Graph WFC algorithm adapts the core idea of WFC from regular grids to graph-structured data. As in WFC, it extracts a set of local patterns and their compatibility relations from the training data and then assembles a synthetic graph by enforcing these constraints [8, 9]. The synthetic graph is expanded incrementally by adding vertices and edges while aiming to approximately preserve empirical statistics of the training data, such as the vertex-degree distribution. This yields graphs that are locally consistent with the observed patterns, but may differ in overall size, global structure, and connectivity [9].

In line with the motivation in Section 1.1, Graph WFC is designed to improve the training of GCN-based classifiers on small or imbalanced datasets. Effective augmentation should introduce synthesised graphs that are locally plausible while still exposing the classifier to new and unseen data. With its explicit, constraint-based construction and absence of learned parameters, Graph WFC provides an alternative augmentation mechanism that enforces compatibility constraints derived from observed local patterns, rather than applying stochastic perturbations such as vertex and edge dropping or feature masking [7, 9].

Since the method does not require fitting additional learned parameters, it avoids a separate training stage. However, synthesis time can vary substantially and may require backtracking or restarts as constraint density increases [9]. In this thesis, Graph WFC is used solely as an augmentation technique for graph classification. A key strength of the method is the preservation of local patterns that may be discriminative, whereas a corresponding weakness is the lack of guarantees for properties that depend on long-range structure.

1.4 Approach

This section summarises the experimental protocol used to evaluate Graph WFC as a data augmentation method for graph classification with a GCN [6]. Each dataset is divided into training, validation, and test subsets using stratified sampling to approximately preserve class proportions in each subset. The validation and test subsets are excluded from augmentation and from parameter fitting, with the validation subset used for model selection and the test subset reserved for final evaluation. Graph WFC is applied exclusively to

the training subset: training graphs are exported to the Graph WFC pipeline, vertex attributes are encoded into discrete codes, synthetic graphs are generated, and the resulting graphs are imported back into the learning pipeline. The GCN is then trained either on the original training subset (baseline) or on an augmented training set that combines the original graphs with those generated by Graph WFC.

To assess when augmentation is beneficial, multiple training runs are conducted per dataset with different random seeds. The main experimental factor is the number of training graphs exported to Graph WFC and augmented with synthetic counterparts, relative to a baseline that uses only real training graphs. Two further factors are varied: the target graph size used during synthesis and the pattern size, which determines the size of the pattern used to derive compatibility constraints in Graph WFC. Experiments explore feasible combinations of these settings for each dataset to identify parameter settings in which Graph WFC augmentation consistently improves performance without changing the classifier. Details of the Graph WFC implementation are presented in Chapter 3, and the datasets, models, augmentation pipeline, and evaluation protocol are described in Chapter 4.

1.5 Research Question and Goals

The following central research question guides this thesis:

> **RQ:** Does Graph-WFC-based data augmentation improve the test-set performance of GCN-based graph classifiers on small or imbalanced graph classification benchmarks, as measured by accuracy, macro-F1, and balanced accuracy, compared with training on the original graphs only?

To address this question, the work pursues two main goals:

- **G1** — Design and implement Graph WFC for undirected, vertex-labelled graphs and integrate it as a configurable graph-level augmentation component in a graph-classification pipeline, with controllable parameters for augmentation amount, target graph size, and pattern size.
- **G2** — Quantitatively evaluate the effect of Graph WFC augmentation on GCN performance across datasets and augmentation settings, and identify parameter regimes in which augmentation is beneficial or detrimental relative to a baseline trained only on real graphs.

Chapter 2

Background

2.1 Graphs

In Graph WFC, the key combinatorial objects are rooted neighbourhoods of bounded radius in a given training graph, viewed as induced subgraphs of that graph. These special subgraphs are considered up to isomorphism. The aim of this section is to provide the necessary theory, step by step, to formalise these rooted neighbourhoods.

Graphs are treated throughout this thesis in a purely combinatorial sense: vertices form a finite set, edges define undirected adjacency between vertices, and labels distinguish vertex types. The notation and terminology largely follow the standard conventions of Diestel and West [12, 13], adapted to finite, simple, undirected graphs with vertex labels.

Thus, the section begins with the underlying combinatorial structure:

Definition 2.1.1 (Graph)**.** A graph is a pair $G = (V, E)$ where V is a finite set and

$$E \subseteq \binom{V}{2} := \big\{\{x, y\} \mid x, y \in V,\ x \neq y\big\},$$

whose elements are called the *edges* of G. The elements of V are the *vertices* of G. By construction, loops and multiple edges are excluded, so G is a finite simple undirected graph.

Definition 2.1.2 (Subgraphs and induced subgraphs)**.** Let $G = (V, E)$ be a graph. If $V' \subseteq V$ and $E' \subseteq E$ are such that every edge $e \in E'$ satisfies $e \subseteq V'$, then $G' = (V', E')$ is a *subgraph* of G. If $V' \subseteq V$ and

$$E' := \{\, e \in E \mid e \subseteq V' \,\},$$

then

$$G[V'] := (V', E')$$

is the *induced subgraph* of G on V'.

A subgraph consists of a subset of the vertices and edges of a graph. An induced subgraph $G[V']$ on a vertex set V' includes all edges of G with both endpoints in V',

whereas a general subgraph may omit some of these edges. Graphs frequently contain vertices of different types, which in this context are represented by vertex labels, while edges remain unlabelled.

Definition 2.1.3 (Vertex-labelling)**.** Let $G = (V, E)$ be a graph. A *vertex-labelling* of G is a function $\ell : V \to \Sigma$, where Σ is a finite alphabet of labels. The labels are categorical symbols (only equality matters), and multiple vertices may share the same label. A graph equipped with a vertex-labelling is written (V, E, ℓ) and is called a *vertex-labelled graph*. When the context is clear, (V, E, ℓ) is abbreviated by G.

In this setting, the vertex set V represents the objects of interest, the edge set E records which pairs of vertices are adjacent, and the vertex-labelling ℓ assigns a categorical type to each vertex. Local structure in a graph is described in terms of the vertices adjacent to a given vertex and the number of such neighbours.

Definition 2.1.4 (Neighbourhood and degree)**.** Let $G = (V, E)$ be a graph and $x \in V$. The *neighbourhood* of x is

$$N_G(x) := \{\, y \in V \mid \{x, y\} \in E \,\}.$$

The *degree* of x is

$$d_G(x) := |N_G(x)|.$$

When the graph G is clear from context, the shorter notations $N(x)$ and $d(x)$ are used.

Thus $N_G(x)$ collects all vertices adjacent to x, and $d_G(x)$ records how many such neighbours x has. These quantities describe the immediate local environment of a vertex and will form the basis for the radius-r neighbourhoods introduced below.

To move beyond the immediate neighbourhood of a vertex, a notion of how far apart two vertices lie in a graph is needed, which will be provided by paths and the associated distance function.

Definition 2.1.5 (Simple paths and distance)**.** Let $G = (V, E)$ be a graph. A *simple path* in G is a finite sequence of pairwise distinct vertices

$$(v_0, \ldots, v_k)$$

such that $\{v_{i-1}, v_i\} \in E$ for all $i = 1, \ldots, k$. Its *length* is k, the number of edges. For $x, y \in V$, an x–y path is a simple path in G with $v_0 = x$ and $v_k = y$. The *distance* $\mathrm{dist}_G(x, y)$ is the length of a shortest x–y path in G. If no such path exists, then

$$\mathrm{dist}_G(x, y) := \infty.$$

For $x, y \in V$, the distance $\mathrm{dist}_G(x, y)$ measures the minimal number of steps needed to move from x to y along edges of the graph. By construction, $\mathrm{dist}_G(x, x) = 0$ and the distance is symmetric in its arguments. When x and y lie in different connected

components of G, the convention $\mathrm{dist}_G(x, y) = \infty$ records that there is no connecting path. In this sense, the distance function captures how close or far vertices are in the connectivity structure of G: small values indicate that two vertices are linked by short paths, whereas large values or ∞ indicate weak or absent connectivity.

The distance function induces vertex-centred neighbourhoods that collect all vertices within a prescribed radius of a given centre. These neighbourhoods are formalised as balls, see Lyons and Peres [14].

Definition 2.1.6 (Balls)**.** Let $G = (V, E)$ be a graph and $v \in V$. For $r \in \mathbb{N}_0 := \{0, 1, 2, \dots\}$, the (closed) ball of radius r around v is

$$B_G(v, r) := \{\, u \in V \mid \mathrm{dist}_G(u, v) \leq r \,\}.$$

Thus $B_G(v, 0) = \{v\}$ contains only the centre, while $B_G(v, 1) = \{v\} \cup N_G(v)$ consists of v together with all its neighbours. As r increases, the balls form a nested family

$$B_G(v, 0) \subseteq B_G(v, 1) \subseteq B_G(v, 2) \subseteq \cdots$$

that captures progressively larger portions of the graph around v. For finite graphs, each ball is a finite vertex set, and restricting G to $B_G(v, r)$ yields a finite induced subgraph that will serve as the radius-r neighbourhood of v in what follows.

To describe local structure not only in terms of adjacency but also in terms of vertex labels, it is convenient to consider the induced subgraph on a ball around a vertex, together with the distinguished centre and the restricted labelling.

Definition 2.1.7 (Rooted radius-r neighbourhoods)**.** Let $G = (V, E, \ell)$ be a vertex-labelled graph, $v \in V$, and $r \in \mathbb{N}_0$. The *rooted radius-r neighbourhood* of v in G is the triple

$$\mathcal{H}_G(v, r) := \big(G[B_G(v, r)],\, v,\, \ell|_{B_G(v,r)}\big),$$

that is, the induced subgraph on the ball $B_G(v, r)$ together with root v and the restriction of the labelling ℓ to $B_G(v, r)$. For $r = 1$, this is the induced subgraph on $\{v\} \cup N_G(v)$ with distinguished centre v and labels inherited from G.

The rooted neighbourhood $\mathcal{H}_G(v, r)$ is finite and vertex-labelled: the vertex v plays the role of the root, the ball $B_G(v, r)$ specifies all vertices within distance at most r of v, and edges and labels are inherited from G. In particular, $\mathcal{H}_G(v, r)$ collects

- the vertices in $B_G(v, r)$,
- all edges of G between those vertices, and
- the values of the labelling function $\ell : V \to \Sigma$ restricted to $B_G(v, r)$, that is, the function $\ell|_{B_G(v,r)} : B_G(v, r) \to \Sigma$.

Equivalently, the restricted labelling satisfies $\ell|_{B_G(v,r)}(u) = \ell(u)$ for all $u \in B_G(v, r)$. It is simply the original labelling $\ell : V \to \Sigma$ viewed as a function on the vertex set

$B_G(v, r) \subseteq V$. This generalises the notion of an r-neighbourhood as used by Winkler [15] to vertex-labelled graphs.

A concrete example is shown in Figure 2.1, where the rooted radius-1 neighbourhood $\mathcal{H}_G(v, 1)$ around a vertex v in a small vertex-labelled graph G is highlighted.

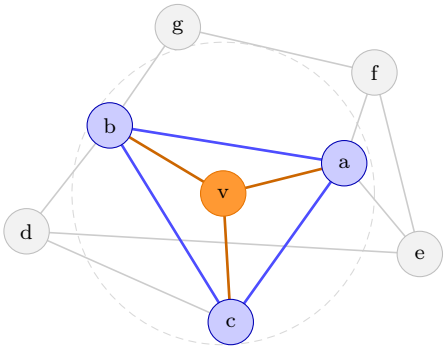

Figure 2.1: Vertex-labelled graph $G = (V, E, \ell)$ drawn in grey, with the rooted radius-1 neighbourhood of v highlighted in colour. The centre v (orange), its neighbours $N_G(v) = \{a, b, c\}$ (blue), and all edges of the induced subgraph $G[\{v\} \cup N_G(v)]$ (highlighted in orange and blue) form the rooted neighbourhood. The dashed circle visualises the ball $B_G(v, 1)$. By definitions 2.1.6 and 2.1.7, the rooted neighbourhood $\mathcal{H}_G(v, 1)$ is the rooted, vertex-labelled graph obtained by restricting G to this ball and distinguishing v as the root.

Now that the theory for extracting finite rooted, vertex-labelled neighbourhoods from a larger graph has been established, the next step is to compare such structures: two rooted neighbourhoods should be regarded as identical if there exists a graph isomorphism between their underlying induced subgraphs. In detail, this means that there exists a bijection between the vertex sets that preserves adjacency and vertex labels and maps the root of one neighbourhood to the root of the other. As a first step, the standard notion of graph isomorphism is recalled.

Definition 2.1.8 (Graph isomorphism)**.** Let $G = (V, E)$ and $G' = (V', E')$ be graphs. An *isomorphism* $\varphi : V \to V'$ is a bijection such that

$$\{x, y\} \in E \iff \{\varphi(x), \varphi(y)\} \in E' \quad \text{for all } x, y \in V.$$

The notation $G \cong G'$ indicates the existence of such an isomorphism. For vertex-labelled graphs $G = (V, E, \ell)$ and $G' = (V', E', \ell')$, an isomorphism must also preserve labels, that is, $\ell(x) = \ell'(\varphi(x))$ for all $x \in V$.

With rooted radius-r neighbourhoods defined in definition 2.1.7 and graph isomorphism defined, the next step is to adapt isomorphism to the rooted setting. In a rooted radius-r neighbourhood, the centre vertex plays a distinguished role and must be preserved by any admissible isomorphism. This leads to the following notion of isomorphism for rooted neighbourhoods.

Definition 2.1.9 (Isomorphism of rooted radius-r neighbourhoods)**.** Let $G = (V, E, \ell)$ and $G' = (V', E', \ell')$ be vertex-labelled graphs, let $v \in V$, $v' \in V'$, and $r \in \mathbb{N}_0$. Their rooted radius-r neighbourhoods are

$$\mathcal{H}_G(v, r) = \big(G[B_G(v, r)],\, v,\, \ell|_{B_G(v,r)}\big),$$
$$\mathcal{H}_{G'}(v', r) = \big(G'[B_{G'}(v', r)],\, v',\, \ell'|_{B_{G'}(v',r)}\big).$$

A *root-preserving labelled isomorphism* between $\mathcal{H}_G(v, r)$ and $\mathcal{H}_{G'}(v', r)$ is a bijection

$$\varphi : B_G(v, r) \longrightarrow B_{G'}(v', r)$$

such that

$$\begin{aligned} \{x, y\} \in E \iff \{\varphi(x), \varphi(y)\} \in E' \quad & \text{for all } x, y \in B_G(v, r), \\ \ell(x) = \ell'\big(\varphi(x)\big) \quad & \text{for all } x \in B_G(v, r), \\ \varphi(v) = v'. & \end{aligned}$$

If such a bijection exists, the two rooted neighbourhoods are said to be *root-preserving labelled isomorphic*, and the notation $\mathcal{H}_G(v, r) \cong_{r\ell} \mathcal{H}_{G'}(v', r)$ is used.

In particular, this notion applies both to rooted neighbourhoods in different graphs and to rooted neighbourhoods within a single graph. The relation of root-preserving labelled isomorphism partitions rooted neighbourhoods into isomorphism classes. Two rooted radius-r neighbourhoods $\mathcal{H}_G(v, r)$ and $\mathcal{H}_{G'}(v', r)$ encode the same local structure precisely when they are root-preserving labelled isomorphic in the sense of Definition 2.1.9. This viewpoint coincides with the standard treatment of rooted neighbourhoods in the theory of local graph limits. See, for example, Lovász [16, Sections 18.3 and 19.1].

In the context of Graph WFC, rooted radius-r neighbourhoods serve as the *patterns*, in direct analogy to the overlapping variant of the WFC algorithm that inspired Graph WFC (see Section 1.2 for the motivation, and Section 2.3 for an explicit formulation of the overlapping variant). For a fixed radius r and a given training graph G, all rooted neighbourhoods $\mathcal{H}_G(v, r)$ with $v \in V(G)$ are extracted, grouped into isomorphism classes under root-preserving labelled isomorphism, and their empirical frequencies are computed. These isomorphism classes form the pattern vocabulary, and the associated frequencies provide local compatibility statistics used in the subsequent Graph WFC constructions. Thus, the constructions in this section provide the theoretical basis for handling rooted radius-r neighbourhoods in the implementation. Their algorithmic application as patterns is developed in Sections 3.1 and 3.2.

2.2 Graph Convolutional Network

To address the research question and goals stated in Section 1.5, a standard graph-classification model is used as a simple baseline. More precisely, a graph convolutional network (GCN) based on the propagation rule of Kipf and Welling [6] is employed as the baseline classifier, using the `GCNConv` and `global_mean_pool` operators provided by PyTorch Geometric [17]. This section provides a compact description of the employed architecture and notation to document the baseline used in all experiments.

Graph neural networks (GNNs) are neural models that operate on graph-structured data by iteratively propagating and transforming vertex features along edges [18]. GCNs constitute a widely used architecture within the GNN family and combine neighbourhood aggregation with shared linear transformations [6]. In the experiments, a standard GCN without any architectural or algorithmic modifications is employed as the classifier.

Using the notation introduced in Section 2.1, let $G = (V, E)$ be a graph with $|V| = N$ vertices. Fix an arbitrary ordering of V and identify $V = \{1, \dots, N\}$, so that G admits an adjacency matrix $A \in \{0,1\}^{N \times N}$. The vertex features are collected in a matrix $X \in \mathbb{R}^{N \times F}$, where the i-th row $x_i^\top$ represents the feature vector of vertex i. Following the formulation of Kipf and Welling [6], self-loops are added by defining $\hat{A} = A + I_N$, and the corresponding degree matrix $\hat{D}$ is diagonal with entries $\hat{D}_{ii} = \sum_{j=1}^{N} \hat{A}_{ij}$. The symmetrically normalised adjacency matrix used in the GCN layer is then given by

$$\tilde{A} \;=\; \hat{D}^{-1/2} \hat{A} \hat{D}^{-1/2}.$$

The initial vertex-feature matrix is defined by $H^{(0)} := X$. A single GCN layer maps input vertex features $H^{(\ell)} \in \mathbb{R}^{N \times F_\ell}$ to output features $H^{(\ell+1)} \in \mathbb{R}^{N \times F_{\ell+1}}$ via

$$H^{(\ell+1)} \;=\; \sigma\big(\tilde{A}\, H^{(\ell)} W^{(\ell)}\big), \tag{2.1}$$

where $W^{(\ell)} \in \mathbb{R}^{F_\ell \times F_{\ell+1}}$ is a trainable weight matrix and $\sigma(\cdot)$ denotes an element-wise nonlinearity. Throughout the experiments, the rectified linear unit (ReLU), defined by $\sigma(t) = \max\{0, t\}$, is used. Bias terms are omitted for readability. This propagation rule corresponds to the `GCNConv` operator from PyTorch Geometric [17], which serves as the basic building block of the baseline classifier.

A graph-level representation is obtained by stacking several layers of the form (2.1) and then applying a pooling operation to the final vertex embeddings. The pooling is chosen to be permutation-invariant, so that the resulting representation does not depend on the arbitrary ordering of vertices. Let $H^{(L)} \in \mathbb{R}^{N \times F_L}$ denote the vertex-feature matrix after L GCN layers. Its i-th row $h_i^{(L)}$ can be interpreted as an embedding of vertex i that summarises information from its L-hop neighbourhood. Because $\tilde{A}$ only mixes features of adjacent vertices, repeated application of (2.1) implies that $h_i^{(L)}$ depends only on the L-hop neighbourhood of i. A global pooling operator aggregates all vertex embeddings

into a single vector

$$h_G \;=\; \mathrm{READOUT}\big(\{h_i^{(L)} : i \in V\}\big) \;\in\; \mathbb{R}^{F_L}. \tag{2.2}$$

In this thesis, READOUT is implemented as global mean pooling, which computes h_G as the average of the final vertex embeddings over all vertices of the graph. The vector h_G is then passed through a linear layer

$$z_G = W_{\text{out}} h_G + b_{\text{out}},$$

which produces C logits $z_G \in \mathbb{R}^C$, that is, one real-valued score for each class before normalisation. These logits are converted into class probabilities by the standard softmax function. The parameters θ, denoting the collection of all trainable weights of the network, are learned by minimising the usual cross-entropy loss on graph labels. With this formulation, both binary and multi-class graph-classification problems are covered through the choice of the number of classes C.

In summary, the classifier employed in the experiments is a conventional GCN implemented with PyTorch Geometric `GCNConv` layers, followed by global mean pooling and a linear output layer, and trained using the usual cross-entropy loss on graph labels. Thus, the contributions of this thesis lie entirely in the Graph WFC implementation and the resulting synthetic training graphs, while the classifier architecture itself is deliberately kept standard.

2.3 Overlapping WFC Model

This section reviews the overlapping Wave Function Collapse (WFC) model as implemented by Gumin [8], with emphasis on pattern extraction, overlap-based adjacency constraints, and grid-based generation. Unless stated otherwise, the algorithmic details in this section follow Gumin's implementation [8]. For an accessible, informal explanation and illustrative example of overlapping WFC are given by Sherratt [19], which also inspired the schematic figures in this section.

2.3.1 Overlapping WFC Algorithm

Pattern extraction and adjacency constraints The algorithm operates on a regular grid and derives its pattern set directly from an input image. Let the size of the input grid be $H \times W$, and let $k \in \mathbb{N}$ denote a chosen pattern size (typically $k \in \{2, 3, 4\}$). A window of size $k \times k$ is anchored with its upper-left corner at grid position $(1, 1)$ of the image. For each anchor position (h, w) with $1 \le h \le H$, $1 \le w \le W$, the algorithm records the $k \times k$ values under the window as one pattern. Conceptually, the window starts at $(1, 1)$ and is moved along the first row through $(1, 2), \ldots, (1, W)$, then continues with the second row $(2, 1), \ldots, (2, W)$, and so on until all rows $h = 1, \ldots, H$ have been visited. Whenever part of the window would extend beyond the right or bottom edge of the image, the missing

values are taken from the opposite side in the corresponding direction, so that the image behaves as if it wraps around horizontally and vertically. In this way, a $k \times k$ pattern is associated with every grid position (h, w) in the input image. An illustration of this pattern extraction procedure is shown in Figure 2.2.

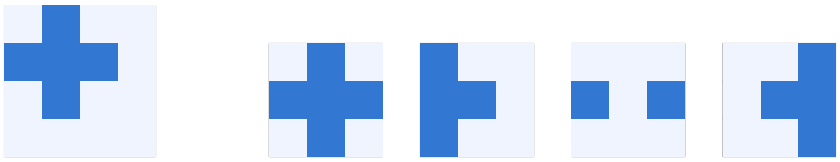

Figure 2.2: Pattern extraction in the overlapping WFC model for pattern size $k = 3$ on a 4×4 input image. The input image is scanned by a 3×3 window from left to right along the top row, producing the first four patterns shown on the right. The same sliding and wrapping procedure is applied at every grid position (h, w) with $1 \leq h \leq H$, $1 \leq w \leq W$, yielding here 16 patterns in total. The layout of this illustration is inspired by the informal example of overlapping WFC in Sherratt [19].

Each distinct $k \times k$ configuration is assigned a pattern index in $\{0, \ldots, |P| - 1\}$, where P denotes the resulting finite pattern set. During pattern extraction, every newly observed pattern is optionally rotated and reflected, and all resulting variants are compared against the existing pattern set P. If none of these variants is present in P, each is inserted as a new pattern with its own index. If a match is found, the corresponding pattern's empirical frequency counter is increased instead of storing a duplicate.

Adjacency constraints between patterns are obtained by testing every ordered pair of patterns for overlap compatibility. For each direction $\hat{d} \in \{\pm\hat{e}_x, \pm\hat{e}_y\}$ and each pair $p_i, p_j \in P$, the algorithm shifts p_j by one cell in direction $\hat{d}$ and compares the overlapping region of size $(k-1) \times k$ or $k \times (k-1)$ in the two patterns. If all values in this overlap are identical, p_i and p_j are marked as compatible in direction $\hat{d}$. Otherwise, the pair is marked as incompatible. Formally, this yields for each direction $\hat{d}$ a binary adjacency matrix $A^{(\hat{d})} \in \{0,1\}^{|P| \times |P|}$ with entries $A_{ij}^{(\hat{d})} = 1$ if and only if p_i and p_j are compatible when p_j is placed one step in direction $\hat{d}$ relative to p_i. An illustration of the pattern overlap compatibility is shown in Figure 2.3. The pattern set P together with the adjacency matrices $\{A^{(\hat{d})}\}$ define the constraints that WFC must enforce during generation.

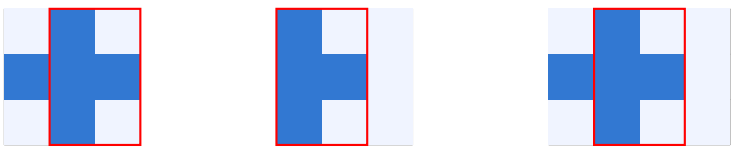

Figure 2.3: Horizontal overlap compatibility between two patterns in the overlapping WFC model. Pattern 1 and Pattern 2 correspond to the first two extracted patterns in Figure 2.2. Pattern 2 is shifted one cell to the right over Pattern 1, and the red rectangles indicate the $k \times (k-1)$ overlap region (here 3×2) that must be identical in both patterns for them to be compatible in this direction. The schematic design of this overlap illustration is likewise inspired by the example in Sherratt [19].

Generation on the output grid Generation is performed on a target grid with a chosen output size. At the start of the algorithm, each grid cell (h, w) is associated with a domain $D_{h,w} \subseteq P$ of admissible patterns. All domains are initialised as $D_{h,w} = P$ for all (h, w), so that every cell initially contains the full pattern set. The algorithm then iteratively selects a cell and collapses its domain by choosing a single pattern from its current domain $D_{h,w}$. To choose which cell to collapse next, WFC uses a Shannon-entropy heuristic based on the empirical pattern frequencies. Let $w_p > 0$ denote the empirical frequency (weight) of pattern $p \in P$ obtained during pattern extraction. For any domain $D \subseteq P$, these weights are restricted to D and renormalised to obtain probabilities

$$q_p \;=\; \frac{w_p}{\sum_{r \in D} w_r} \quad \text{for } p \in D,$$

and the Shannon entropy of D is defined as

$$H(D) \;=\; -\sum_{p \in D} q_p \, \log q_p.$$

At each step, the algorithm considers all cells with $|D_{h,w}| > 1$, computes $H(D_{h,w})$ for each, and picks one whose domain has the lowest entropy, breaking ties at random. At the very beginning, all domains are identical, so the first cell is effectively selected uniformly at random. Once a cell (h, w) has been chosen, one pattern $p^* \in D_{h,w}$ is drawn at random with probabilities $\{q_p\}_{p \in D_{h,w}}$, and the domain is replaced by the singleton $\{p^*\}$. In this way, more frequent patterns are more likely to be chosen, but the choice remains stochastic.

Collapsing a cell with chosen pattern p^* imposes additional constraints on its neighbours. Let i^* denote the index of p^* in P. For each of the four directions $\hat{d} \in \{\pm\hat{e}_x, \pm\hat{e}_y\}$, consider the adjacent cell in direction $\hat{d}$ (if it lies inside the output grid) and denote its current domain by $D_{\text{nbr}} \subseteq P$. This domain is intersected with the set of patterns that are compatible with p^* according to the adjacency matrices $A^{(\hat{d})}$ defined above:

$$D_{\text{nbr}} \;\leftarrow\; \{\, p_j \in D_{\text{nbr}} \mid A^{(\hat{d})}_{i^* j} = 1 \,\}.$$

In other words, every neighbour of the collapsed cell loses all patterns that would violate the overlap constraints in the corresponding direction. Whenever a neighbour's domain changes, that neighbour is treated as a new source of constraints, and the same update is applied to its own neighbours. This propagation continues until no domain changes any more. If, during this propagation, any domain $D_{h,w}$ becomes empty, implying a contradiction, the run is aborted and the generation restarts. If no errors occur and propagation terminates successfully, the algorithm recomputes the entropies $H(D_{h,w})$ and again selects a cell of lowest entropy to collapse. This collapse and propagation cycle is repeated until all domains have collapsed to singletons, at which point each output cell (h, w) contains exactly one pattern index, yielding a complete pattern assignment on the target grid.

To render the final image, a fixed pixel position within each $k \times k$ pattern is chosen as

an anchor (commonly the top-left pixel). The algorithm outputs a grid of pattern indices, and each output cell is rendered by drawing only the anchor pixel colour of its assigned pattern [19]. Because the adjacency constraints have been enforced consistently across the grid, selecting the same anchor pixel for every pattern yields a coherent image whose local statistics reflect those of the exemplar, while the underlying $k \times k$ patterns remain implicit.

2.3.2 WFC for Graph Generation

Viewed abstractly, WFC greedily solves a constraint satisfaction problem on a grid: each cell chooses a pattern so that all local adjacency constraints, learned from an exemplar, are satisfied [9]. At the same time, sampling patterns according to their empirical frequencies makes the generated output locally resemble the exemplar in terms of pattern statistics, while still allowing globally different configurations [8, 9]. This combination of strict local compatibility and flexible global structure is appealing for graph generation, because message-passing GNNs such as GCNs can be viewed as updating vertex representations by aggregating information from their local neighbourhoods over successive layers [18]. Moreover, in supervised graph learning, data augmentation is attractive when it preserves label-relevant local structures while still generating globally diverse graphs [5, 7]. Generative models such as NetGAN empirically demonstrate that it is possible to sample synthetic graphs that approximate the structural statistics of an observed graph while still producing new global graph instances [20].

Taken together, these observations suggest that the overlapping WFC model may be a suitable candidate for graph generation, provided it can be reformulated to operate on graphs rather than grids. The next chapter presents Graph WFC, one approach to applying the overlapping WFC model to graphs.

Chapter 3

Implementation

3.1 Graph Patterns and Data I/O

This section specifies the data input and output (data I/O) formats and introduces the Graph WFC model, which represents patterns as rooted radius-r neighbourhoods, as defined in definition 2.1.7. A later subsection describes how compatibility constraints between patterns are extracted. These components are inspired by the overlapping WFC model (Section 2.3) and together form the first stage of the graph generation pipeline.

3.1.1 Data I/O

The Graph WFC generator takes as input a finite family of undirected, vertex-labelled training graphs as in definition 2.1.1 and produces synthetic graphs over the same vertex-label alphabet. Formally, the input is

$$\mathcal{G}_{\text{train}} = \big\{\, G_k = (V_k, E_k, \ell_k) \mid k = 1, \dots, M \,\big\}, \tag{3.1}$$

where each G_k is a vertex-labelled graph in the sense of definition 2.1.3, with labelling function $\ell_k : V_k \to \mathcal{L}$ for a fixed finite label alphabet $\mathcal{L}$. The corresponding outputs are

$$\mathcal{G}_{\text{syn}} = \big\{\, \tilde{G}_k = (\tilde{V}_k, \tilde{E}_k, \tilde{\ell}_k) \mid k = 1, \dots, M \,\big\}, \tag{3.2}$$

which are vertex-labelled graphs over $\mathcal{L}$. Thus, the generator attempts to produce one synthetic graph $\tilde{G}_k$ for each input graph G_k. Unsuccessful synthesis attempts may yield no output graph, so the number of outputs may be smaller than the number of inputs.

For each input $G_k = (V_k, E_k, \ell_k)$ with $k = 1, \dots, M$, the generator extracts a pattern vocabulary and associated compatibility tables from G_k, and then synthesises the corresponding synthetic graph $\tilde{G}_k$ using only quantities learned from this training graph. After the synthesis step, all pattern data and summary statistics from iteration k are discarded, and the internal state of the generator is reinitialised before the next training graph is processed. For the remainder of this section, the mathematical and algorithmic procedure is described for a fixed but arbitrary iteration k, since each iteration is self-contained.

3.1.2 Pattern Extraction

Given a training graph G_k from Equation (3.1) and a fixed radius $r \in \mathbb{N}$ with $r \geq 1$, the first stage of the algorithm is *pattern extraction*. The algorithm iterates once over all vertices $v \in V_k$ and, for each vertex, forms the rooted radius-r neighbourhood $\mathcal{H}_{G_k}(v, r)$ as defined in definition 2.1.7. Thus, for a training graph with vertex set V_k, the pattern-extraction step yields $|V_k|$ rooted neighbourhoods. For notational convenience, the following shorthand is introduced for these neighbourhoods:

$$H_k^{(r)}(v) \;:=\; \mathcal{H}_{G_k}(v, r), \tag{3.3}$$

so that $H_k^{(r)}(v)$ denotes the rooted radius-r neighbourhood around v in G_k. To identify duplicates among radius-r neighbourhoods, these rooted graphs are compared up to root-preserving vertex-labelled isomorphism, as defined in definition 2.1.9. Two vertices $v, v' \in V_k$ therefore give rise to the same local structure at radius r precisely when

$$H_k^{(r)}(v) \cong_{r\ell} H_k^{(r)}(v').$$

The corresponding set of patterns at radius r is the set of $\cong_{r\ell}$-equivalence classes of rooted neighbourhoods in G_k and is defined as

$$\mathcal{P}_k^{(r)} \;:=\; \big\{\, [H_k^{(r)}(v)]_{\cong_{r\ell}} : v \in V_k \,\big\}, \tag{3.4}$$

where $[H_k^{(r)}(v)]_{\cong_{r\ell}}$ denotes the $\cong_{r\ell}$-equivalence class of $H_k^{(r)}(v)$. Each class $\pi \in \mathcal{P}_k^{(r)}$ is called a *pattern* at radius r.

To quantify how often each pattern occurs at radius r, the multiplicity $m_k^{(r)}(\pi)$ of a pattern $\pi \in \mathcal{P}_k^{(r)}$ is defined as the number of vertices whose rooted neighbourhood lies in that class:

$$m_k^{(r)}(\pi) \;:=\; \big|\{\, v \in V_k : H_k^{(r)}(v) \in \pi \,\}\big|. \tag{3.5}$$

Since every vertex contributes its neighbourhood to exactly one pattern class, the multiplicities satisfy

$$\sum_{\pi \in \mathcal{P}_k^{(r)}} m_k^{(r)}(\pi) = |V_k|. \tag{3.6}$$

The associated empirical distribution of radius-r patterns can then be defined by

$$\widehat{p}_k^{(r)}(\pi) \;:=\; \frac{m_k^{(r)}(\pi)}{|V_k|}, \qquad \pi \in \mathcal{P}_k^{(r)}. \tag{3.7}$$

By construction, this is a probability mass function on $\mathcal{P}_k^{(r)}$, and in particular

$$\sum_{\pi \in \mathcal{P}_k^{(r)}} \widehat{p}_k^{(r)}(\pi) = 1. \tag{3.8}$$

Equivalently, $\widehat{p}_k^{(r)}(\pi)$ is the probability that a uniformly chosen vertex $v \in V_k$ has rooted neighbourhood $H_k^{(r)}(v)$ in the pattern class π. This choice of normalisation (uniform root and radius-r neighbourhoods) is consistent with the Benjamini–Schramm framework for local limits of finite graphs [21, §1.2].

So far, patterns have been defined for a single radius r. In practice, however, the implementation considers a finite set of radii. Let

$$\mathcal{R} \;:=\; \{1, \dots, r_{\max}\} \subseteq \mathbb{N} \tag{3.9}$$

denote the set of radii of interest, where $r_{\max} \geq 1$ is a chosen maximal radius. For each radius $r \in \{1, \dots, r_{\max}\}$, the corresponding pattern set $\mathcal{P}_k^{(r)}$ is constructed as in (3.4).

For the generation stage, only patterns at the maximal radius $r_{\max}$ are used as domain values. As in the overlapping WFC algorithm (Section 2.3.1), each cell in the Graph WFC maintains a domain of admissible patterns of a single size (see Section 3.2.1). In the Graph WFC this common size is fixed to radius-$r_{\max}$ neighbourhoods. Hence the pattern vocabulary presented in the cell domains during synthesis is

$$\Pi_k \;:=\; \mathcal{P}_k^{(r_{\max})}, \tag{3.10}$$

and each cell ultimately selects one element of Π_k as its pattern when it collapses to a singleton. The general definition (3.7) yields, for $r = r_{\max}$, an empirical distribution $\widehat{p}_k^{(r_{\max})}$ on $\mathcal{P}_k^{(r_{\max})}$. Restricting this to the vocabulary $\Pi_k \subseteq \mathcal{P}_k^{(r_{\max})}$ gives

$$\widehat{p}_k(\pi) \;:=\; \widehat{p}_k^{(r_{\max})}(\pi) \;=\; \frac{m_k^{(r_{\max})}(\pi)}{|V_k|}, \qquad \pi \in \Pi_k, \tag{3.11}$$

so that $\widehat{p}_k$ is simply the radius-$r_{\max}$ empirical distribution from (3.7), restricted to those patterns that are actually used as domain values.

For later use in generation, pattern weights are introduced by

$$w_k(\pi) \;:=\; m_k^{(r_{\max})}(\pi), \qquad \pi \in \Pi_k, \tag{3.12}$$

so that $w_k(\pi)$ coincides with the multiplicity $m_k^{(r_{\max})}(\pi)$ from (3.5). Only the relative magnitudes of these weights are used during synthesis, for example to bias pattern selection in the entropy-based collapse step (Section 3.2.2).

3.1.3 Pattern Compatibility

Let G_k be a training graph from Equation (3.1) and $r \in \mathcal{R}$. Denote $\mathcal{P}_k^{(r)}$ as the corresponding set of radius-r patterns from Section 3.1.2. This subsection defines a compatibility relation on $\mathcal{P}_k^{(r)}$ that encodes which patterns may occur consistently adjacent for all radii r. Informally, two patterns are declared compatible if they share at least one vertex-label sequence along a simple root-to-radius-r path in one pattern and the reversed sequence along a radius-r-to-root path in the other. The construction therefore first specifies how

these label sequences are extracted from a single pattern, and then defines compatibility by comparing the resulting sets.

Root-to-radius paths of a pattern. Let $\pi \in \mathcal{P}_k^{(r)}$ be a pattern at radius r. By the definition of $\mathcal{P}_k^{(r)}$ in (3.4), there exists a vertex $v \in V_k$ such that the rooted radius-r neighbourhood $H_k^{(r)}(v)$ belongs to the pattern class π (see definitions 2.1.7 and 2.1.9). In the following, such a rooted neighbourhood $H_k^{(r)}(v)$ is called a representative of π. A *root-to-radius-r path* in a representative $H_k^{(r)}(v)$ is a path $(v_0, \ldots, v_r)$ in the sense of definition 2.1.5 that satisfies

$$v_0 = v \quad \text{and} \quad \operatorname{dist}_{G_k}(v_i, v) = i \quad \text{for all } i = 0, \ldots, r.$$

Such a path starts at the root v and moves strictly outwards in graph distance, increasing the distance to the root by exactly one at each step until distance r is reached. The associated label sequence is defined by

$$\lambda(v_0, \ldots, v_r) := \big(\ell_k(v_0), \ldots, \ell_k(v_r)\big) \in \mathcal{L}^{r+1},$$

where $\ell_k : V_k \to \mathcal{L}$ is the vertex-labelling from Equation (3.1). Root-preserving labelled isomorphisms preserve distances from the root and vertex labels (see definition 2.1.9). Consequently, the collection of label sequences $\lambda(v_0, \ldots, v_r)$ arising from all root-to-radius-r paths in a representative $H_k^{(r)}(v)$ depends only on the pattern π, not on the chosen representative. Hence the set of label sequences extracted from a pattern is well defined, which allows the following construction.

Definition 3.1.1 (Outward label-path set of a pattern)**.** The *outward label-path set* of a pattern $\pi \in \mathcal{P}_k^{(r)}$ is

$$\mathcal{L}_r(\pi) := \big\{ \lambda(v_0, \ldots, v_r) \,\big|\, (v_0, \ldots, v_r) \text{ is a root-to-radius-}r \text{ path of } \pi \big\} \subseteq \mathcal{L}^{r+1}.$$

Thus $\mathcal{L}_r(\pi)$ collects all label sequences of the pattern π that occur along simple paths starting at the root of π and moving strictly outwards layer by layer up to radius r. For a given pattern π and radius r, it may happen that π contains no root-to-radius-r path (for example, when no vertex lies at distance exactly r from the root), therefore $\mathcal{L}_r(\pi) = \varnothing$.

Reversed path sets and compatibility. To express that two patterns can be aligned along at least one path as mentioned at the beginning of this subsection, a reversal operation on label sequences is introduced. For a sequence

$$a = (a_0, \ldots, a_r) \in \mathcal{L}^{r+1},$$

its reversal is defined as

$$\operatorname{rev}(a) := (a_r, \ldots, a_0).$$

For a subset $S \subseteq \mathcal{L}^{r+1}$, the pointwise reversal is written

$$\mathrm{rev}(S) := \{\, \mathrm{rev}(a) \mid a \in S \,\}.$$

Definition 3.1.2 (Reversed label-path set of a pattern)**.** The *reversed label-path set* of a pattern $\pi \in \mathcal{P}_k^{(r)}$ is

$$\mathcal{L}_r^{\mathrm{rev}}(\pi) := \mathrm{rev}\big(\mathcal{L}_r(\pi)\big) = \{\, \mathrm{rev}(a) \mid a \in \mathcal{L}_r(\pi) \,\} \subseteq \mathcal{L}^{r+1}.$$

Thus the compatibility of two patterns can now be expressed in terms of intersections of outward and reversed path sets:

Definition 3.1.3 (Pattern compatibility)**.** Let $\pi, \pi' \in \mathcal{P}_k^{(r)}$. The patterns π and π' are *compatible at radius* r if

$$\mathcal{L}_r(\pi) \cap \mathcal{L}_r^{\mathrm{rev}}(\pi') \neq \varnothing.$$

If $\mathcal{L}_r(\pi) = \varnothing$, then the pattern π contains no root-to-radius-r label sequence and therefore provides no compatibility evidence at radius r. Under the stated compatibility criterion, π is incompatible with every radius-r pattern.

Compatibility matrix. For a fixed training graph G_k and a fixed radius $r \in \mathcal{R}$, the compatibility relation from definition 3.1.3 induces a symmetric $\{0,1\}$-valued matrix over the pattern set $\mathcal{P}_k^{(r)}$. Let $n := |\mathcal{P}_k^{(r)}|$ denote the number of radius-r patterns in G_k, and fix an enumeration

$$\mathcal{P}_k^{(r)} = \{\pi_1^{(r)}, \ldots, \pi_n^{(r)}\}.$$

The *compatibility matrix* at radius r for the training graph G_k is defined as

$$C_k^{(r)} \in \{0,1\}^{n \times n},$$

with entries

$$C_k^{(r)}(i,j) := \begin{cases} 1, & \pi_i^{(r)} \text{ and } \pi_j^{(r)} \text{ are compatible at radius } r, \\ 0, & \text{otherwise,} \end{cases} \qquad i,j \in \{1,\ldots,n\},$$

where compatibility is understood in the sense of definition 3.1.3. Since the compatibility relation is symmetric, the matrix $C_k^{(r)}$ is symmetric. For the given radius r, $C_k^{(r)}$ therefore provides a complete description of which patterns may occur adjacent to one another along root-to-radius-r paths in the Graph WFC generation process.

For later use in Section 3.2.4 it is convenient to read $C_k^{(r)}$ row-wise as compatibility neighbourhoods on $\mathcal{P}_k^{(r)}$. For $\rho \in \mathcal{P}_k^{(r)}$, let $i \in \{1,\ldots,n\}$ be such that $\rho = \pi_i^{(r)}$ and define

$$\mathcal{N}_k^{(r)}(\rho) \;:=\; \big\{\, \pi_j^{(r)} \in \mathcal{P}_k^{(r)} \;\big|\; C_k^{(r)}(i,j) = 1 \,\big\}. \tag{3.13}$$

Thus $\mathcal{N}_k^{(r)}(\rho)$ is the set of all radius-r patterns that are compatible with ρ.

Recall that in Section 3.1.2, patterns were extracted separately for each radius $r \in \mathcal{R}$, see (3.9). Applying the construction above to every such radius yields, for each training graph G_k, the family of compatibility matrices

$$\mathcal{C}_k \;:=\; \{\, C_k^{(r)} \mid r \in \mathcal{R} \,\}. \tag{3.14}$$

Equivalently, one may collect the associated neighbourhood maps as

$$\mathcal{N}_k \;:=\; \{\, \mathcal{N}_k^{(r)} \mid r \in \mathcal{R} \,\}. \tag{3.15}$$

The families $\mathcal{C}_k$ and $\mathcal{N}_k$ describe the same multi-radius compatibility structure: $\mathcal{C}_k$ stores it as $\{0,1\}$-valued matrices, whereas $\mathcal{N}_k$ stores it as set-valued maps $\rho \mapsto \mathcal{N}_k^{(r)}(\rho)$ on the pattern sets $\mathcal{P}_k^{(r)}$.

3.2 Graph Generation

The constructions in Sections 3.1.2 and 3.1.3 associate to each training graph G_k a family of compatibility matrices $\mathcal{C}_k$ (see (3.14)) and the corresponding compatibility neighbourhood maps $\mathcal{N}_k$ (see (3.15)) on the pattern sets $\mathcal{P}_k^{(r)}$ (see (3.13)). These structures encode, for all radii $r \in \mathcal{R}$, which patterns may occur adjacent along root-to-radius-r paths in G_k. This section explains how a new graph $\tilde{G}_k$ is synthesised from these compatibility constraints. The core design follows the overlapping WFC model introduced in Section 2.3.

At a high level, the generator maintains an evolving *cell graph*: a finite simple graph whose vertices are abstract *cells*, not the actual vertices of the synthetic graph. Each cell is a placeholder for a future vertex of $\tilde{G}_k$ and is equipped with a domain of admissible patterns from the global vocabulary $\Pi_k = \mathcal{P}_k^{(r_{\max})}$ at the reference radius $r_{\max}$ (see Equation (3.10)). During generation, this cell graph is grown and rewired, while the domains of its cells are progressively reduced until each cell has collapsed to a single pattern. The algorithm proceeds roughly as follows.

Generation is initialised with a single seed cell whose domain equals the full pattern vocabulary Π_k. The following steps from a single iteration, which is repeated until the cell graph reaches the prescribed target size:

- a cell is selected via a minimum-entropy heuristic and its domain is *collapsed* to a single pattern;
- the cell graph is *expanded* by adding new neighbouring cells with initial domain Π_k and inserting edges;
- constraints are *propagated* by pruning the domains of adjacent cells using the multi-radius compatibility neighbourhood maps $\mathcal{N}_k$;
- collapsed cells are *reconnected* by inserting additional edges, subject to the same compatibility structure encoded by $\mathcal{N}_k$.

Once the cell graph reaches the prescribed target size, a final clean-up phase is applied. At this point, each cell has a singleton domain. The readout step then replaces each cell by the centre vertex (root) of its assigned pattern together with the corresponding label, yielding the synthetic graph $\tilde{G}_k$. This centre vertex is directly analogous to the anchor pixel in the overlapping WFC model (Section 2.3). This completes the informal outline of the algorithm. The next subsection formalises the initialisation of the Graph WFC state. Subsequent subsections cover the expansion, propagation, reconnection and clean-up phases, followed by the readout of the synthetic graph $\tilde{G}_k$ from the fully collapsed cell configuration.

3.2.1 Initialisation

The initialisation phase configures the global hyperparameters of the Graph WFC model and sets up the initial state from which the iterative process starts. For a fixed training graph $G_k = (V_k, E_k, \ell_k)$, the pattern vocabularies $\big(\mathcal{P}_k^{(r)}\big)_{r\in\mathcal{R}}$ and the compatibility structure $(\mathcal{C}_k, \mathcal{N}_k)$ introduced in Sections 3.1.2 and 3.1.3 are treated as precomputed inputs. A maximal pattern radius $r_{\max} \in \mathcal{R}$ is regarded as a global hyperparameter and induces the reference-radius pattern set $\Pi_k = \mathcal{P}_k^{(r_{\max})}$ via Equations (3.9) and (3.10). This subsection first introduces the global parameters that control target size and expansion behaviour, and then formalises the cell-level state (cells, domains, and adjacency) together with its initial configuration.

Global parameters. For the training graph G_k, let $n_k := |V_k|$ denote its number of vertices. The target size of the synthetic graph $\tilde{G}_k$ is controlled by a user-specified size factor $c_{\text{size}} > 0$, yielding the nominal target vertex count

$$N_k^{\text{tar}} := \big\lfloor c_{\text{size}}\, n_k \big\rfloor, \tag{3.16}$$

which serves as the reference scale for generation. As outlined at the start of this section, synthesis consists of a generation phase (the iterative collapse, expand, propagate, and connect loop) followed by a clean-up phase that completes the process. To define the size-based transition between these phases, a fixed tolerance constant $\delta_{\text{size}} := 0.1$ is used. The size-based transition from the generation phase to the clean-up phase is triggered once the following lower threshold is reached:

$$N_k^{\text{low}} \;:=\; \big\lfloor (c_{\text{size}} - \delta_{\text{size}})\, n_k \big\rfloor. \tag{3.17}$$

Afterwards, the clean-up phase begins and expansion is constrained by the hard upper cap

$$N_k^{\max} \;:=\; \big\lceil (c_{\text{size}} + \delta_{\text{size}})\, n_k \big\rceil. \tag{3.18}$$

A second global parameter, the *expansion cap*,

$$B_k^{\text{exp}} \in \mathbb{N}, \tag{3.19}$$

which bounds the number of new cells that may be introduced in a single expansion step. This cap helps to avoid tree-like blow-up, especially when several cells are expanded in one step (see Section 3.2.3 for the allocation rule). In the implementation, $B_k^{\exp}$ is chosen from the degree distribution of G_k as a fixed high-percentile value. Let $d_{G_k}(v)$ denote the degree of a vertex $v \in V_k$ in G_k (cf. definition 2.1.4). Let $d_{(1)}^{(k)} \le \cdots \le d_{(n_k)}^{(k)}$ be the non-decreasing ordering of the multiset $\{\, d_{G_k}(v) : v \in V_k \,\}$, and set

$$i_k := \max\{\, 1,\ \lceil 0.9\, n_k \rceil \,\}, \qquad B_k^{\exp} := d_{(i_k)}^{(k)}. \tag{3.20}$$

$B_k^{\exp}$ is the empirical 90th percentile of the vertex degrees in G_k. This deliberately simple, hand-crafted heuristic keeps local branching under control by ensuring that no expansion step introduces more new cells than the degree of a typical high-degree vertex.

In summary, the adjustable hyperparameters in the initialisation step are the maximal pattern radius $r_{\max}$ (which determines the vocabulary Π_k via Equations (3.9) and (3.10)) and the size factor c_{size} controlling the nominal target count N_k^{tar}. A fixed design constant $\delta_{\text{size}} = 0.1$ defines the size band used to derive the lower threshold N_k^{low} and the hard upper cap $N_k^{\max}$. The choice of using the empirical 90th percentile for the expansion cap $B_k^{\exp}$ is treated as part of the fixed model design in this work.

Cells and domains. From Equation (3.10), the global pattern vocabulary at the reference radius is given by

$$\Pi_k = \mathcal{P}_k^{(r_{\max})}.$$

During the synthesis of $\tilde{G}_k$, the generator maintains a finite set $\mathcal{X}_k$ of abstract vertex placeholders, called *cells*, and a domain map

$$D : \mathcal{X}_k \to 2^{\Pi_k}, \qquad x \mapsto D(x), \tag{3.21}$$

where each value $D(x)$ is finite and non-empty. At any time, the set $D(x) \subseteq \Pi_k$ contains exactly those patterns that are still considered admissible, given the multi-radius compatibility constraints encoded by the compatibility structure $(\mathcal{C}_k, \mathcal{N}_k)$. A cell $x \in \mathcal{X}_k$ is called *uncollapsed* if $|D(x)| > 1$ and *collapsed* if $|D(x)| = 1$. Accordingly, the cell set $\mathcal{X}_k$ is partitioned into

$$\mathcal{F}_k := \{\, x \in \mathcal{X}_k : |D(x)| > 1 \,\}, \tag{3.22}$$

$$\mathcal{S}_k := \{\, x \in \mathcal{X}_k : |D(x)| = 1 \,\}, \tag{3.23}$$

referred to as the *frontier* (uncollapsed) and *settled* (collapsed) cell sets, respectively. Thus

$$\mathcal{F}_k \cap \mathcal{S}_k = \varnothing, \qquad \mathcal{F}_k \cup \mathcal{S}_k = \mathcal{X}_k.$$

Subsequent subsections specify how $\mathcal{X}_k$, D, and the sets $\mathcal{F}_k$ and $\mathcal{S}_k$ are updated by expansion, propagation, and collapse operations.

Adjacency structure. Each cell $x \in \mathcal{X}_k$ is equipped with an *adjacency list*

$$\mathrm{Adj}_k(x) \subseteq \mathcal{X}_k, \tag{3.24}$$

which stores the neighbours of x in the evolving cell graph. The lists are maintained to be loop-free and symmetric,

$$x \notin \mathrm{Adj}_k(x) \quad \text{and} \quad y \in \mathrm{Adj}_k(x) \iff x \in \mathrm{Adj}_k(y),$$

so that the family $\big(\mathrm{Adj}_k(x)\big)_{x\in\mathcal{X}_k}$ induces a simple undirected graph on vertex set $\mathcal{X}_k$. This graph represents the connectivity of the partially constructed graph throughout the synthesis process. In the readout phase, each cell is replaced by the centre vertex along with its label of its unique pattern, and this cell graph is carried over to $\tilde{G}_k$ as its edge set (see Section 3.2.6).

Initial seed. At the start of generation, a single seed cell x_{seed} is created with domain

$$D(x_{\mathrm{seed}}) = \Pi_k, \tag{3.25}$$

and empty adjacency list $\mathrm{Adj}_k(x_{\mathrm{seed}}) = \varnothing$. Thus initially

$$\mathcal{X}_k = \{x_{\mathrm{seed}}\}, \qquad \mathcal{F}_k = \{x_{\mathrm{seed}}\}, \qquad \mathcal{S}_k = \varnothing.$$

This configuration serves as the initial Graph WFC state for graph synthesis. The subsequent expansion, propagation, collapse, and connection steps operate on this state, as described in the following subsections.

3.2.2 Entropy and Collapse

At a high level, each call of the collapse mechanism proceeds as follows. The frontier cells $x \in \mathcal{F}_k$ (3.22) are inspected and assigned a Shannon-entropy score $H_k(x)$ based on a cell-wise pattern distribution induced by the global pattern weights w_k from Equation (3.12), and a cell of minimal entropy is selected for collapse. When the collapse stage is called directly after initialisation, the frontier consists only of the seed cell, $\mathcal{F}_k = \{x_{\mathrm{seed}}\}$, so the entropy-based selection degenerates to collapsing x_{seed}.

More precisely, for a frontier cell $x \in \mathcal{F}_k$ with domain $D(x) \subseteq \Pi_k$, the pattern weights w_k from Equation (3.12) induce a discrete distribution over the admissible patterns:

$$p_x(\pi) := \frac{w_k(\pi)}{\displaystyle\sum_{\pi' \in D(x)} w_k(\pi')}, \qquad \pi \in D(x). \tag{3.26}$$

Here the denominator sums only over the current domain $D(x)$, so p_x is a probability distribution on $D(x)$: each currently admissible pattern $\pi \in D(x)$ receives probability $p_x(\pi)$, while patterns $\pi \notin D(x)$ are ignored. On this basis, the uncertainty associated

with a cell x is quantified by the Shannon entropy of its distribution (3.26),

$$H_k(x) \;:=\; -\sum_{\pi \in D(x)} p_x(\pi)\, \log p_x(\pi), \tag{3.27}$$

where the logarithm is taken in the natural base. Note that cells are moved from the frontier set $\mathcal{F}_k$ to the settled set $\mathcal{S}_k$ as soon as their domain becomes a singleton. Consequently, the entropy $H_k(x)$ is evaluated only for frontier cells $x \in \mathcal{F}_k$.

In each collapse phase, the entropy $H_k(x)$ is evaluated for all frontier cells $x \in \mathcal{F}_k$, and a cell of minimum entropy is selected:

$$x^\star \in \arg\min\{\, H_k(x) : x \in \mathcal{F}_k \,\}. \tag{3.28}$$

When several cells attain the same minimum entropy, ties are resolved by selecting one of them uniformly at random. If $\mathcal{F}_k = \varnothing$ (that is, if all cells have collapsed), no further collapse is possible and the collapse step signals termination. This minimum-entropy selection rule follows the standard WFC heuristic: it prefers cells that are already tightly constrained (low entropy) but not yet forced, thereby greedily resolving first those positions whose pattern choice is most constrained and therefore certain.

Once a target cell $x^\star$ has been selected via (3.28), the next step is to commit to a specific pattern inside its domain. Rather than choosing uniformly from $D(x^\star)$, the generator uses the empirical frequencies encoded in the weights w_k from Equation (3.12). For the domain $D(x^\star)$, consider the following conceptual construction: each admissible pattern $\pi \in D(x^\star)$ is replicated $w_k(\pi)$ times, and one copy is chosen uniformly at random from the resulting multiset. The probability of selecting a particular pattern $\pi \in D(x^\star)$ is then

$$\mathbb{P}\big[\pi^\star = \pi\big] \;=\; \frac{w_k(\pi)}{\displaystyle\sum_{\pi' \in D(x^\star)} w_k(\pi')} \;=\; p_{x^\star}(\pi),$$

which coincides with the distribution $p_{x^\star}$ defined in (3.26). The collapse step reduces the domain of $x^\star$ to the singleton

$$D(x^\star) \leftarrow \{\pi^\star\}, \tag{3.29}$$

that is, $x^\star$ is assigned the unique remaining admissible pattern $\pi^\star$. After this update, $x^\star$ is moved from the frontier to the settled set,

$$\mathcal{F}_k \leftarrow \mathcal{F}_k \setminus \{x^\star\}, \qquad \mathcal{S}_k \leftarrow \mathcal{S}_k \cup \{x^\star\}.$$

For subsequent subsections, it is convenient to single out, in each iteration, the cells that have just collapsed. Recall that one *iteration* refers to one full update cycle of the generator, consisting of the calls *collapse*, *expand*, *propagate*, and *reconnect*. In a given iteration, collapsing the entropy-selected cell $x^\star$ may trigger further domain reductions via constraint propagation (see Section 3.2.4), so that additional neighbouring frontier cells lose all but one admissible pattern and collapse as well. To capture this batch of newly

collapsed cells, let

$$\mathcal{S}_k^{\text{new}} \subseteq \mathcal{S}_k \tag{3.30}$$

denote the set of cells whose domains became singletons during the current iteration. Thus, $\mathcal{S}_k^{\text{new}}$ always contains the entropy-chosen cell $x^\star$ and may contain additional cells that were forced to collapse by propagation. After the iteration completes, $\mathcal{S}_k^{\text{new}}$ is reset to $\varnothing$. In the following subsections, the set $\mathcal{S}_k^{\text{new}}$ serves both as the parent set for local expansion steps (Section 3.2.3) and as the active source set for constraint propagation (Section 3.2.4).

3.2.3 Graph Expansion

In each iteration, the collapse phase produces a set of newly collapsed cells $\mathcal{S}_k^{\text{new}}$ as defined in Equation (3.30). These cells act as *parents* for the subsequent expansion step. Expansion grows the cell graph by creating new, uncollapsed neighbour cells around the parents and by inserting the corresponding edges. The number of new neighbours assigned to each parent is determined from three ingredients: a per-parent *degree demand* derived from the centre vertex of the chosen pattern, a global *expansion budget* limiting how many new cells may be introduced in the current iteration, and a per-parent *capacity* that bounds how much of the demand is realised through newly created neighbours. After the allocation is fixed, the new cells are added to the frontier set $\mathcal{F}_k$ and connected to their parents in the cell graph.

This subsection first defines the degree demand and the global budget, then specifies the allocation rule $b(x)$ of new neighbours per parent, and finally describes how the new cells and adjacencies are created.

Degree-based demand. Each pattern $\pi \in \Pi_k$ is a rooted radius-$r_{\max}$ neighbourhood with a distinguished centre vertex in the sense of definition 2.1.7. Whenever a cell $x \in \mathcal{S}_k$ is collapsed to a pattern π with $D(x) = \{\pi\}$, it is assigned the centre degree

$$d(x) := d_{G_k}\big(v_{\text{centre}}(\pi)\big), \tag{3.31}$$

where $v_{\text{centre}}(\pi)$ denotes the centre (root) vertex of the pattern π. The value $d(x)$ is used as the *degree demand* of x. The total degree demand of all parents newly collapsed in the current iteration is

$$d_{\text{tot}} := \sum_{x \in \mathcal{S}_k^{\text{new}}} d(x). \tag{3.32}$$

If $\mathcal{S}_k^{\text{new}} = \varnothing$ or $d_{\text{tot}} = 0$, no expansion is performed in the current iteration.

Expansion budget. During the generation phase, expansion is performed only up to the size-based transition threshold N_k^{low} defined in Equation (3.17). Given the current cell

set $\mathcal{X}_k$, the remaining quota with respect to this threshold is

$$q_k \;:=\; \max\{\,0,\; N_k^{\text{low}} - |\mathcal{X}_k|\,\}. \tag{3.33}$$

The expansion budget for the current step is then defined as

$$B \;:=\; \min\{B_k^{\text{exp}},\, q_k\} \;\in\; \mathbb{N}_0. \tag{3.34}$$

Thus at most B_k^{exp} new cells are introduced in a single expansion step, and the expansion step cannot increase the cell count beyond N_k^{low}. If $B = 0$, the expansion step is skipped. In the remainder of this subsection, it is assumed that $\mathcal{S}_k^{\text{new}} \neq \varnothing$, $d_{\text{tot}} > 0$, and $B > 0$.

Allocation of new neighbours. For each parent $x \in \mathcal{S}_k^{\text{new}}$, the expansion step assigns a non-negative integer $b(x) \in \mathbb{N}_0$, specifying how many new uncollapsed cells are created as neighbours of x. To avoid tree-like growth of the synthetic graph, only a limited fraction of the degree demand $d(x)$ is realised by new neighbours in a single expansion step. The expansion capacity of each parent is defined as

$$c(x) \;:=\; \left\lceil \frac{d(x)}{2} \right\rceil, \qquad x \in \mathcal{S}_k^{\text{new}}, \tag{3.35}$$

which is a simple choice that restricts the number of newly created neighbours per expansion step and leaves the remaining demand to be realised via connections between already existing collapsed cells in the connection step (Section 3.2.5). The total capacity of the current parent batch is thus given by

$$c_{\text{tot}} \;:=\; \sum_{x \in \mathcal{S}_k^{\text{new}}} c(x), \qquad x \in \mathcal{S}_k^{\text{new}} \tag{3.36}$$

which is the maximum number of new cells this batch could introduce in the absence of the global budget constraint. The expansion budget B limits how many new cells may be created in this iteration. The assignment $(b(x))_{x \in \mathcal{S}_k^{\text{new}}}$ is defined by two cases:

- *Abundant budget.* If the global budget is at least as large as the total local capacity ($B \geq c_{\text{tot}}$), then the budget does not bind in this iteration. Each parent thus realises its full local capacity of new neighbours and

  $$b(x) = c(x) \quad \text{for all } x \in \mathcal{S}_k^{\text{new}}. \tag{3.37}$$

  In particular, $\sum_{x \in \mathcal{S}_k^{\text{new}}} b(x) = c_{\text{tot}} \leq B$.

- *Scarce budget.* If $B < c_{\text{tot}}$, the global budget is insufficient to realise all local capacities. The distribution of new neighbours is then guided by the degree demands $d(x)$. The total degree demand d_{tot} of the batch is given by (3.32), and each parent

is assigned a degree-based weight

$$p(x) \;:=\; \frac{d(x)}{d_{\mathrm{tot}}}, \qquad x \in \mathcal{S}_k^{\mathrm{new}}, \tag{3.38}$$

forming a probability distribution on $\mathcal{S}_k^{\mathrm{new}}$ that reflects the relative degree demands of each parent. The ideal number of new neighbours for parent x in this expansion step is

$$\tilde{b}(x) \;:=\; B\,p(x), \qquad x \in \mathcal{S}_k^{\mathrm{new}}.$$

Based on these targets, a non-negative assignment is constructed. First, a preliminary number of new neighbours is set to

$$b(x) \;:=\; \min\{\, c(x),\, \lfloor \tilde{b}(x) \rfloor \,\},$$

so that each parent receives no more than its capacity and at most its floored ideal share. Let

$$B_0 \;:=\; \sum_{x \in \mathcal{S}_k^{\mathrm{new}}} b(x)$$

denote the total number of new cells after this preliminary step, and let

$$R \;:=\; B - B_0 \;\in\; \mathbb{N}_0$$

denote the remaining budget. If $R > 0$, the remaining budget is distributed by a largest-remainder-style correction. For each parent define the discrepancy

$$\delta(x) \;:=\; \tilde{b}(x) - b(x), \qquad x \in \mathcal{S}_k^{\mathrm{new}}.$$

The parents are ordered in decreasing order of $\delta(x)$, with ties broken by increasing cell index. While $R > 0$ and there exists a parent with $b(x) < c(x)$, this order is traversed from first to last: for each parent x in the list, if $R > 0$ and $b(x) < c(x)$, then $b(x) \leftarrow b(x) + 1$ and $R \leftarrow R - 1$. The traversal is repeated until $R = 0$ or $b(x) = c(x)$ for all $x \in \mathcal{S}_k^{\mathrm{new}}$. The resulting assignment satisfies $0 \le b(x) \le c(x)$ for all $x \in \mathcal{S}_k^{\mathrm{new}}$ and $\sum_{x \in \mathcal{S}_k^{\mathrm{new}}} b(x) \le B$.

Creation of new cells. Once all $b(x)$ have been determined, the cell graph is expanded by introducing new uncollapsed cells. For each parent $x \in \mathcal{S}_k^{\mathrm{new}}$ with $b(x) > 0$, new child cells

$$y_{x,1}, \ldots, y_{x,b(x)}$$

are created. Each child starts with the full global pattern vocabulary as its domain,

$$D(y_{x,i}) = \Pi_k \quad \text{for } i = 1, \ldots, b(x). \tag{3.39}$$

The new cells are added to the frontier cell set,

$$\mathcal{F}_k \leftarrow \mathcal{F}_k \cup \{y_{x,1}, \ldots, y_{x,b(x)}\}.$$

Since the invariant $\mathcal{X}_k = \mathcal{F}_k \cup \mathcal{S}_k$ is maintained, this update also extends the cell set $\mathcal{X}_k$. The adjacency list Adj_k from Equation (3.24) is then updated symmetrically,

$$y_{x,i} \in \mathrm{Adj}_k(x), \qquad x \in \mathrm{Adj}_k(y_{x,i}), \quad \text{for } i = 1, \ldots, b(x), \tag{3.40}$$

so that each new child cell is connected to its parent by a single undirected edge in the cell graph, and no other adjacencies involving the new cells are introduced at this stage. Since the new cells are now adjacent to existing cells in the cell graph, the constraint propagation procedure from Section 3.2.4 has to be applied next to prune their domains $D(y_{x,i})$ using $\mathcal{N}_k$ from Equation (3.15) in response to the newly induced adjacency constraints.

3.2.4 Constraint Propagation

After each local expansion described in Section 3.2.3, the domains of frontier cells must be pruned so that all remaining patterns are locally compatible with the patterns already fixed in the settled set $\mathcal{S}_k$. Collapsed cells induce compatibility constraints for nearby cells, where the relevant constraints depend on the cell-graph distance in the evolving cell graph defined below. The constraint-propagation step uses the compatibility neighbourhood maps $\mathcal{N}_k$ from Section 3.1.3 and Equation (3.15) to intersect the domains $D(x)$ along this cell graph. Within a single propagation call, only a subset of collapsed cells is treated as sources: the newly collapsed batch $\mathcal{S}_k^{\mathrm{new}}$ from Equation (3.30) in the first pass, and the cells that become forced during propagation in subsequent passes.

Constraint propagation consists of three components. First, the adjacency lists Adj_k induce a cell graph on $\mathcal{X}_k$ together with a graph-distance d_k^{cell} that measures how far cells lie from one another. Second, the multi-radius compatibility maps $\mathcal{N}_k^{(r)}$ on the radius-r pattern sets $\mathcal{P}_k^{(r)}$ are lifted to maps on the radius-$r_{\max}$ pattern vocabulary Π_k via restriction maps, so that all pruning operations can be carried out directly on domains $D(\cdot) \subseteq \Pi_k$. Third, a single-source breadth-first propagation mechanism is defined: given a source cell x with $D(x) = \{\pi_x\}$, the procedure traverses the cell graph up to distance $r_{\max}$ from x and prunes the domains of reached frontier cells according to the lifted compatibility neighbourhoods. A *propagation call* applies this mechanism iteratively, intertwined with graph-expansion steps, starting from the initial source batch $\mathcal{S}_k^{\mathrm{new}}$ and repeatedly expanding and propagating from newly forced cells until no additional cells are forced. This nested procedure forms the propagation phase within a single iteration of the generation loop.

Cell graph and distance. The adjacency lists Adj_k from Equation (3.24) induce a finite simple undirected graph on $\mathcal{X}_k$ in the sense of definition 2.1.1. Let $d_k^{\mathrm{cell}}(x, y)$ denote the graph-distance between $x, y \in \mathcal{X}_k$ in this induced graph, as defined in definition 2.1.5.

Thus $d_k^{\mathrm{cell}}(x,y)$ is the length of a shortest path between x and y, and $d_k^{\mathrm{cell}}(x,y) = \infty$ if no such path exists.

Restriction and lifted compatibility neighbourhoods. Domains of cells are defined over the radius-$r_{\max}$ pattern vocabulary $\Pi_k = \mathcal{P}_k^{(r_{\max})}$ from (3.10), whereas the compatibility neighbourhoods $\mathcal{N}_k^{(r)}$ from (3.13) live on the radius-r pattern sets $\mathcal{P}_k^{(r)}$. If two cells $x, y \in \mathcal{X}_k$ satisfy $1 \le d_k^{\mathrm{cell}}(x,y) \le r_{\max}$, set $r := d_k^{\mathrm{cell}}(x,y)$. Then the radius-$r$ neighbourhood map $\mathcal{N}_k^{(r)}$ selects the compatibility constraints relevant for local consistency.

Each radius-$r_{\max}$ pattern $\pi \in \Pi_k$ implicitly determines a family of concentric subpatterns at smaller radii: these arise by restricting the same rooted neighbourhood around the same centre vertex to radii $1, \dots, r_{\max}$ and then identifying the result up to rooted labelled isomorphism. The smaller-radius patterns encode the same local structure as π, restricted to smaller balls, and allow the corresponding compatibility constraints to be enforced at their respective radii. This association is provided by restriction maps. For every $r \in \mathcal{R}$, a restriction map

$$T_k^{(r)} : \Pi_k \to \mathcal{P}_k^{(r)} \tag{3.41}$$

is defined by restricting a representative rooted neighbourhood of a radius-$r_{\max}$ pattern to radius r. More precisely, if $\pi \in \Pi_k$ is represented by $H_k^{(r_{\max})}(v)$ in the sense of (3.3) and (3.4), then

$$T_k^{(r)}(\pi) \;:=\; \big[H_k^{(r)}(v)\big]_{\cong_{r\ell}} \;\in\; \mathcal{P}_k^{(r)}.$$

Root-preserving labelled isomorphisms (definition 2.1.9) preserve distances from the root and vertex labels, so the radius-r class on the right-hand side does not depend on the chosen representative of π. A radius-$r_{\max}$ pattern $\pi \in \Pi_k$ therefore represents an entire family $\big(T_k^{(r)}(\pi)\big)_{r\in\mathcal{R}}$ of consistent smaller-radius views.

The radius-r compatibility neighbourhoods $\mathcal{N}_k^{(r)}(\rho) \subseteq \mathcal{P}_k^{(r)}$ from (3.13) are lifted to the radius-$r_{\max}$ pattern vocabulary via the restriction maps. For every $\pi \in \Pi_k$ and $r \in \mathcal{R}$, define its *lifted compatibility neighbourhood* at distance r by

$$\widehat{\mathcal{N}}_k^{(r)}(\pi) \;:=\; \big\{\, \pi' \in \Pi_k \;\big|\; T_k^{(r)}(\pi') \in \mathcal{N}_k^{(r)}\big(T_k^{(r)}(\pi)\big) \,\big\} \;\subseteq\; \Pi_k. \tag{3.42}$$

Thus two radius-$r_{\max}$ patterns $\pi, \pi' \in \Pi_k$ are regarded as compatible at cell distance $r \in \mathcal{R}$ if their radius-r restrictions $T_k^{(r)}(\pi)$ and $T_k^{(r)}(\pi')$ are compatible. Here $\mathcal{N}_k^{(r)}$ denotes the original neighbourhood map on $\mathcal{P}_k^{(r)}$, while $\widehat{\mathcal{N}}_k^{(r)}$ is its lifted counterpart on Π_k used during constraint propagation.

Compatibility constraints and propagation from sources. In a propagation pass, the active sources are the newly collapsed cells $\mathcal{S}_k^{\mathrm{new}}$. For each source $x \in \mathcal{S}_k^{\mathrm{new}}$ with $D(x) = \{\pi_x\}$, a breadth-first traversal of the induced cell graph on $\mathcal{X}_k$ rooted at x is performed up to depth $r_{\max}$. The update below is applied only to frontier cells. Whenever a frontier cell $y \in \mathcal{F}_k$ is first reached at distance $r := d_k^{\mathrm{cell}}(x,y) \in \{1, \dots, r_{\max}\}$, local

consistency with respect to x requires

$$D(y) \;\subseteq\; \widehat{\mathcal{N}}_k^{(r)}(\pi_x). \tag{3.43}$$

This constraint is enforced by the update

$$D(y) \;\leftarrow\; D(y) \cap \widehat{\mathcal{N}}_k^{(r)}(\pi_x). \tag{3.44}$$

Equivalently, a pattern $\pi' \in D(y)$ is retained only if $T_k^{(r)}(\pi') \in \mathcal{N}_k^{(r)}\big(T_k^{(r)}(\pi_x)\big)$. Since breadth-first search reaches y first at its minimal cell-graph distance from x, this minimal r is used for the update. Thus the pruning removes exactly those radius-$r_{\max}$ pattern candidates whose radius-r restrictions are incompatible with the radius-r restriction of the source pattern. Each frontier cell is visited at most once at its minimal distance from the fixed source x, so its domain is pruned at most once with respect to x. If any update produces $D(y) = \varnothing$, a contradiction is detected and the current generation run is aborted and restarted for the same training graph.

After all sources $x \in \mathcal{S}_k^{\text{new}}$ have been processed, any frontier cell $y \in \mathcal{F}_k$ satisfies

$$D(y) \;\subseteq \bigcap_{\substack{x \in \mathcal{S}_k^{\text{new}}: \\ 1 \le d_k^{\text{cell}}(x,y) \le r_{\max}}} \widehat{\mathcal{N}}_k^{\left(d_k^{\text{cell}}(x,y)\right)}(\pi_x). \tag{3.45}$$

If no source $x \in \mathcal{S}_k^{\text{new}}$ satisfies $1 \le d_k^{\text{cell}}(x,y) \le r_{\max}$, then this pass induces no pruning for y and its domain remains unchanged.

Forced cells and nested propagation–expansion loop. In a single propagation pass from a given source batch, some frontier cells may become fully determined purely due to the compatibility constraints. For that pass, let $\mathcal{F}_k^{\text{start}} \subseteq \mathcal{F}_k$ denote the frontier at the beginning of the pass. By construction of the generation loop, all cells in $\mathcal{F}_k^{\text{start}}$ are uncollapsed at this point and satisfy $|D(y)| > 1$. The set of *forced* cells created during the pass is then

$$F_{\text{forced}} \;:=\; \big\{\, y \in \mathcal{F}_k^{\text{start}} : |D(y)| = 1 \,\big\}. \tag{3.46}$$

These are exactly the cells that were uncollapsed at the start of the pass and have become singleton-domain cells by the end of the pass. After the pass, the frontier and settled sets are updated accordingly,

$$\mathcal{F}_k \;\leftarrow\; \mathcal{F}_k \setminus F_{\text{forced}}, \qquad \mathcal{S}_k \;\leftarrow\; \mathcal{S}_k \cup F_{\text{forced}}, \tag{3.47}$$

which formally records the collapse of all forced cells. By design, a single propagation call (as invoked by the outer generation loop) is organised as a nested loop:

- *Outer initial step.* The call starts from the given batch of sources $\mathcal{S}_k^{\text{new}}$ (containing the entropy-selected cell from Section 3.2.2 and any other cells that have collapsed

during the current iteration). A propagation pass is run once from this source batch using the update rule (3.44), and the resulting set F_{forced} is computed as in (3.46)

- *Inner loop.* As long as $F_{\text{forced}} \neq \varnothing$, the following two steps are repeated:

  1. a graph-expansion step is performed with parent set F_{forced}, as described in Section 3.2.3, thereby adding new uncollapsed neighbours and updating the adjacency lists Adj_k;
  2. a new propagation pass is run with the source batch updated to $\mathcal{S}_k^{\text{new}} \leftarrow F_{\text{forced}}$. Let $\mathcal{F}_k^{\text{start}}$ denote the frontier at the beginning of this pass (that is, after the preceding expansion step). Domains of reached frontier cells are then pruned by repeatedly applying the update rule (3.44) with respect to all sources in $\mathcal{S}_k^{\text{new}}$. After the pass, the set of newly forced cells is recomputed as in (3.46) using this $\mathcal{F}_k^{\text{start}}$, and $\mathcal{F}_k$ and $\mathcal{S}_k$ are updated according to (3.47).

The propagation call terminates as soon as an iteration of the inner loop produces no forced cells, that is, when

$$F_{\text{forced}} \;=\; \varnothing. \tag{3.48}$$

At this point, all compatibility constraints implied by the original source batch $\mathcal{S}_k^{\text{new}}$ and by the edges created in the intermediate expansion steps have been enforced for the current partial assignment. The constraint-propagation phase of the iteration is then complete, and the generation loop proceeds to the connection step in Section 3.2.5.

3.2.5 Graph Connection

After the expansion and propagation steps in a given iteration, the cell set $\mathcal{X}_k$ contains a subset of collapsed cells forming the settled set $\mathcal{S}_k$ from (3.23). Each collapsed cell $x \in \mathcal{S}_k$ carries a unique pattern $\pi_x \in \Pi_k$ and an associated degree demand $d(x)$, defined in (3.31) as the degree of the centre vertex of π_x. The current connectivity between cells is described by the adjacency relation Adj_k. The *graph connection* step tries to add new edges between collapsed cells so that their degrees in the cell graph come as close as possible to the degree demands $d(x)$, while respecting the multi-radius compatibility constraints given by the lifted neighbourhood maps $\widehat{\mathcal{N}}_k^{(r)}$ from (3.42). This step operates purely on the adjacency structure of $\mathcal{S}_k$ and does not modify the domains $D(x)$.

Degree deficits and stubs. For each collapsed cell $x \in \mathcal{S}_k$, define its current cell-graph degree by

$$\deg_k^{\text{cell}}(x) \;:=\; \big|\text{Adj}_k(x)\big|. \tag{3.49}$$

The remaining *stub count* of x is then

$$s(x) \;:=\; \max\big\{\, 0,\; d(x) - \deg_k^{\text{cell}}(x) \,\big\} \;\in\; \mathbb{N}_0, \tag{3.50}$$

which measures how many additional incident edges x needs in order to realise its degree demand $d(x)$ derived from the centre vertex of π_x. Define the set of cells with positive stub count by

$$\mathcal{S}_k^{\text{stub}} \;:=\; \big\{\, x \in \mathcal{S}_k : s(x) > 0 \,\big\} \;\subseteq\; \mathcal{S}_k. \tag{3.51}$$

Locally compatible candidate pairs. A new edge can only be introduced between cells $x, y \in \mathcal{S}_k^{\text{stub}}$ if they are *locally compatible* and not already adjacent. Let $\pi_x, \pi_y \in \Pi_k$ denote the patterns chosen for x and y, respectively. The implementation first restricts attention to unordered pairs

$$\{x, y\} \in \binom{\mathcal{S}_k^{\text{stub}}}{2} \quad \text{such that} \quad y \notin \mathrm{Adj}_k(x),$$

and then enforces radius-1 compatibility at the level of radius-$r_{\max}$ patterns by requiring

$$\pi_y \in \widehat{\mathcal{N}}_k^{(1)}(\pi_x), \tag{3.52}$$

where $\widehat{\mathcal{N}}_k^{(1)}$ is the lifted radius-1 neighbourhood map on Π_k defined in (3.42). Condition (3.52) ensures that the two radius-$r_{\max}$ patterns have been observed as direct neighbours in the training graph and therefore may plausibly be linked in $\tilde{G}_k$. Pairs that fail (3.52), or that already share an edge, are discarded and never considered for wiring. Since the training graphs are undirected, the underlying compatibility relation at radius 1 is symmetric, so the one-sided check in (3.52) suffices.

Multi-radius path validation. Direct radius-1 compatibility is not sufficient, since adding a new edge $\{x, y\}$ can shorten cell-graph distances between y and other already collapsed cells reachable from x. Any such distance decrease can make additional radius-r constraints (with $r \geq 2$) newly relevant. To avoid violating the precomputed compatibility tables at larger radii, each candidate pair $\{x, y\}$ that passes (3.52) is subjected to a multi-radius validation step.

Fix such a pair and consider the direction from x to y. Let π_y be the pattern at the prospective neighbour y. A breadth-first search is run from x in the current cell graph, using only existing edges in Adj_k (that is, the candidate edge $\{x, y\}$ is not yet present) and exploring only collapsed cells. For each depth $t \in \{1, \ldots, r_{\max} - 1\}$, every collapsed cell z discovered at distance t from x carries a fixed pattern π_z, and the algorithm checks that

$$\pi_y \in \widehat{\mathcal{N}}_k^{(t+1)}(\pi_z). \tag{3.53}$$

If the edge $\{x, y\}$ were to be added, then y would be connected to z by a path of length $t+1$ via x. Whenever this new path shortens the existing z to y distance, the newly relevant compatibility level is exactly radius $t + 1$, and (3.53) enforces it. The breadth-first search maintains a visited set rooted at x, so that no cell is revisited at a larger distance after having been seen at a smaller one, and each collapsed cell z is checked at its minimal distance from x.

The same validation is then performed in the opposite direction: a breadth-first search is run from y, and for each collapsed cell z at distance t from y the condition

$$\pi_x \in \widehat{\mathcal{N}}_k^{(t+1)}(\pi_z) \tag{3.54}$$

is required. If the edge $\{x, y\}$ were to be added, then x would be connected to z by a path of length $t+1$ via y. Whenever this new path shortens the existing z to x distance, the newly relevant compatibility level is exactly radius $t+1$. A candidate pair $\{x, y\}$ is retained only if all checks of the form (3.53) and (3.54) succeed. Otherwise, the pair is discarded. Uncollapsed cells are ignored during this validation, since their patterns are not yet fixed and therefore cannot give rise to deterministic violations of the compatibility tables.

Scoring candidate pairs by resource allocation. Among the surviving candidate pairs $\{x, y\}$, the implementation assigns a numerical score that favours cells whose patterns share a rich, overlapping radius-1 neighbourhood in the compatibility tables. For patterns $\pi_x, \pi_y \in \Pi_k$, the *resource-allocation* (RA) score is defined as

$$\mathrm{RA}(\pi_x, \pi_y) := \sum_{\pi_m \in \widehat{\mathcal{N}}_k^{(1)}(\pi_x) \cap \widehat{\mathcal{N}}_k^{(1)}(\pi_y)} \frac{1}{\left|\widehat{\mathcal{N}}_k^{(1)}(\pi_m)\right|}. \tag{3.55}$$

This quantity is a weighted common-neighbour measure on the compatibility graph: each common neighbour π_m contributes more when it has relatively few admissible radius-1 neighbours and less when it has a large radius-1 neighbourhood. Consequently, a high RA score indicates that π_x and π_y tend to appear near the same specific, low-degree patterns in the training data, whereas a low RA score arises either when they share few common neighbours or when these neighbours are predominantly generic, high-degree patterns. Connecting π_x and π_y is therefore favoured when they share a strong, specific local context in the learned compatibility structure.

Greedy stub wiring. Once each admissible candidate pair $\{x, y\}$ has been assigned a score $\mathrm{RA}(\pi_x, \pi_y)$, the connection step proceeds greedily. All candidate pairs are sorted in descending order of (3.55), and then processed one by one. Ties are broken deterministically by lexicographic order of the cell creation indices. For a pair $\{x, y\}$, an edge is added between x and y if and only if both cells still have outstanding stubs ($s(x) > 0$ and $s(y) > 0$), and no edge between x and y exists yet. When an edge is added, the adjacency relation is updated symmetrically,

$$y \in \mathrm{Adj}_k(x), \qquad x \in \mathrm{Adj}_k(y),$$

and the stub counts $s(x)$ and $s(y)$ are decremented by one. The greedy process terminates when either all stubs have been exhausted, $s(x) = 0$ for all $x \in \mathcal{S}_k^{\mathrm{stub}}$, or no further admissible candidate pairs remain. In the latter case, some cells may retain positive stub

counts, and the realised degrees $\deg_k^{\text{cell}}(x)$ then fall short of the targets $d(x)$.

From a graph-theoretic perspective, the connection step takes the current partially wired cell graph and incrementally fills the remaining edge slots of collapsed cells using only those links that are supported by the empirical multi-radius compatibility tables on Π_k. Together with the expansion and propagation mechanisms described in Sections 3.2.3 and 3.2.4, this stub-wiring procedure drives the evolving cell graph towards a configuration in which vertex degrees and local neighbourhood patterns resemble those of the training graph G_k, while keeping all edges consistent with the learned pattern statistics. The particular combination of RA-based scoring and greedy stub selection used here is an implementation choice adopted for clarity and simplicity.

3.2.6 Termination

Termination policy. For a fixed training graph G_k, synthesis proceeds in two phases, a generation phase and a clean-up phase, governed by the frontier $\mathcal{F}_k$, the settled set $\mathcal{S}_k$ from Equations (3.22) and (3.23), and the nominal target size N_k^{tar} from Equation (3.16). The size-based transition threshold N_k^{low} and the hard upper cap N_k^{max} are defined in the initialisation step (Equations (3.17) and (3.18)).

During the generation phase, the algorithm alternates between entropy-based collapse of frontier cells (Section 3.2.2), graph expansion around newly collapsed cells (Section 3.2.3), constraint propagation (Section 3.2.4), and the connection step (Section 3.2.5). This phase terminates as soon as one of the following conditions holds:

- the current cell count reaches the transition threshold, $|\mathcal{X}_k| \geq N_k^{\text{low}}$;
- no further entropy-based collapse is possible because $\mathcal{F}_k = \varnothing$.

In addition, if at any point a propagation step produces an empty domain $D(x) = \varnothing$ for some cell x, a hard contradiction has been detected and the current generation run is aborted and restarted for the same training graph.

The clean-up phase starts once the generation phase has stopped. It continues to apply the same four primitives as in the generation phase: entropy-based collapse, local expansion, constraint propagation, and connection, but under a progressively stricter expansion policy and under the hard size cap N_k^{max}. Let B_k^{exp} be the base expansion cap from Equation (3.19). A size-dependent decay factor $\alpha_k \in [0, 1]$ is defined by

$$\alpha_k := \begin{cases} 1, & |\mathcal{X}_k| \leq N_k^{\text{tar}}, \\ 0, & |\mathcal{X}_k| \geq N_k^{\text{max}}, \\ 1 - \dfrac{|\mathcal{X}_k| - N_k^{\text{tar}}}{N_k^{\text{max}} - N_k^{\text{tar}}}, & N_k^{\text{tar}} < |\mathcal{X}_k| < N_k^{\text{max}}, \end{cases} \tag{3.56}$$

so that the real-valued product $\alpha_k B_k^{\text{exp}}$ decreases linearly from B_k^{exp} (at $|\mathcal{X}_k| = N_k^{\text{tar}}$) to 0

(at $|\mathcal{X}_k| = N_k^{\max}$). The effective integer cap is then

$$B_k^{\text{lin}} \;:=\; \lceil \alpha_k \, B_k^{\exp} \rceil, \tag{3.57}$$

which follows this linear schedule up to integer rounding. In addition, let

$$s_{\text{open}} \;:=\; \sum_{x \in \mathcal{S}_k} s(x) \tag{3.58}$$

be the total number of open stubs Equation (3.50) across all settled cells. Since the current frontier $\mathcal{F}_k$ already offers $|\mathcal{F}_k|$ potential attachment sites, and since expansion cannot exceed the remaining size quota

$$q_k^{\max} \;:=\; \max\{0,\, N_k^{\max} - |\mathcal{X}_k|\}, \tag{3.59}$$

the expansion allowance used in this iteration is

$$B_k^{\text{allow}} \;:=\; \min\{\, B_k^{\text{lin}},\; \max\{0,\, s_{\text{open}} - |\mathcal{F}_k|\},\; q_k^{\max} \,\}. \tag{3.60}$$

Thus no more new cells are created than are needed to serve outstanding stubs, and this allowance decreases to 0 as the hard size cap is approached.

The clean-up loop terminates either when the hard upper bound $|\mathcal{X}_k| = N_k^{\max}$ is reached, or when $\mathcal{F}_k = \varnothing$ and the subsequent connection attempt adds no further edges (meaning, no admissible candidate pair remains in the sense of Section 3.2.5). In the former case, a final pass collapses any remaining frontier cells without further expansion and performs one last global connection attempt. At the end of this process, every cell in $\mathcal{X}_k$ belongs to $\mathcal{S}_k$ and carries a singleton domain $D(x)$.

Readout. Once synthesis has terminated, every cell $x \in \mathcal{X}_k$ is collapsed and carries a singleton domain $D(x) = \{\pi_x\}$ with $\pi_x \in \Pi_k$. The synthetic graph $\tilde{G}_k$ is read out from the final cell configuration and the cell-adjacency structure. Let

$$\mathcal{X}_k = \{x_0, \ldots, x_{n-1}\}$$

be the enumeration of cells in increasing order of their creation index, and define the vertex set

$$\tilde{V}_k := \{0, \ldots, n-1\},$$

where index i corresponds to cell x_i. For each $i \in \tilde{V}_k$, choose any $v_i \in V_k$ such that $\pi_{x_i} = [H_k^{(r_{\max})}(v_i)]_{\cong_{r\ell}}$, and define the label of vertex i in $\tilde{G}_k$ by

$$\tilde{\ell}_k(i) \;:=\; \ell_k(v_i).$$

This definition is well defined because root-preserving labelled isomorphisms preserve the root label. The labelling map $\tilde{\ell}_k : \tilde{V}_k \to \mathcal{L}$ is therefore a vertex labelling in the sense of

definition 2.1.3. The edge set $\tilde{E}_k$ is obtained from the symmetric adjacency relation Adj_k on cells by inserting, for each pair $0 \leq i < j \leq n-1$, an undirected edge $\{i, j\}$ whenever $x_j \in \mathrm{Adj}_k(x_i)$. The result is a simple vertex-labelled graph $\tilde{G}_k = (\tilde{V}_k, \tilde{E}_k, \tilde{\ell}_k)$ in the sense of definition 2.1.1.

3.3 Configuration and Reproducibility

The behaviour of Graph WFC is controlled by a small set of configuration parameters and a single source of randomness. This section summarises the main choices and explains under which conditions runs are reproducible.

Pattern radius and compatibility radii. The pattern vocabulary and compatibility tables are determined by the maximal pattern radius $r_{\max}$ and the associated radius set $\mathcal{R}$ from Equation (3.9) (see Sections 3.1.2 and 3.1.3). For fixed $r_{\max}$ and a training graph G_k, the pattern-extraction and compatibility stages are deterministic: the radius-wise pattern sets $\mathcal{P}_k^{(r)}$, the radius-$r_{\max}$ pattern vocabulary $\Pi_k = \mathcal{P}_k^{(r_{\max})}$, the weights w_k, and the compatibility neighbourhoods $\mathcal{N}_k^{(r)}$ are uniquely determined by G_k and do not involve any random choices.

Target size and growth parameters. For each training graph G_k, the desired size of the synthetic graph $\tilde{G}_k$ is specified via the target size N_k^{tar} in Equation (3.16), which is obtained from the vertex count $n_k = |V_k|$ and a constant size factor $c_{\mathrm{size}} > 0$. In addition, the expansion cap B_k^{exp} from Equation (3.19) controls how many new cells may be introduced per expansion step, and the lower and upper size fractions used by the growth and clean-up phases (Sections 3.2.3 and 3.2.6) determine when expansion is throttled or halted. Once $(r_{\max}, c_{\mathrm{size}}, B_k^{\mathrm{exp}})$ and these phase thresholds have been fixed, the expansion, connection, and propagation routines (Sections 3.2.3 to 3.2.5) are deterministic given the current WFC state, provided that all internal tie-breaking and enumerations are performed deterministically.

Seeding and randomness. In the presented implementation, the only stochastic choices in Graph WFC occur during entropy-based collapse (Section 3.2.2). First, when the minimum-entropy rule (3.28) admits multiple frontier cells, one of them is selected uniformly at random. Second, after a target cell has been selected, the collapsed pattern is sampled from its domain according to the frequency-weighted distribution (3.26). All remaining components (pattern extraction, compatibility computation, expansion, constraint propagation, connection, and readout) are deterministic given G_k, the configuration parameters, and a fixed deterministic traversal order. In particular, whenever an ordering is required (for example, when iterating over source batches, traversing adjacency lists, enumerating candidate pairs, or sorting scored candidates), iteration is performed in increasing cell creation index, and unordered pairs $\{x, y\}$ are ordered lexicographically by $(\min\{i(x), i(y)\}, \max\{i(x), i(y)\})$, where $i(\cdot)$ denotes the cell creation index.

In the implementation, a single pseudo-random number generator is used for the random choices in collapse. Initialising this generator with a fixed seed yields reproducible runs for a fixed implementation and configuration: with the same seed and the same settings $(r_{\max}, \mathcal{R}, c_{\text{size}}, B_k^{\exp})$ and growth/clean-up thresholds, the same sequence of collapses is produced and the resulting synthetic graph $\tilde{G}_k$ is identical. If a generation run is aborted and restarted after a contradiction, reproducibility of the full run additionally requires that the pseudo-random number generator state be reset deterministically (by re-initialising with the same seed).

Chapter 4

Methods and Experimental Evaluation

4.1 Overview

This chapter investigates whether dataset augmentation with Graph WFC affects the supervised graph classification performance of a fixed GCN. In line with the goals defined in Section 1.5, the benchmark datasets DD, MSRC_21, and PROTEINS are used to train this fixed architecture, where the training set is augmented with graphs synthesised by Graph WFC under varying augmentation configurations. The central question concerns how the presence and amount of such augmentation affect test-set performance on these datasets. Performance is measured using accuracy, macro-F1, and balanced accuracy on held-out test data. Overall, the results indicate that Graph WFC augmentation can improve performance for some datasets and configurations, while effects remain small for others.

The remainder of this chapter is organised as follows. Section 4.2 introduces the datasets and associated classification tasks. Section 4.3 specifies the GCN architecture, the training procedure, and the repeated-seed evaluation protocol. Section 4.4 details the Graph WFC data augmentation pipeline, from dataset splitting and feature encoding through export, synthesis, and re-import of synthetic graphs to the construction of the effective training sets. Section 4.5 defines the evaluation protocol and metrics and explains how results are aggregated and compared across configurations. Section 4.6 presents the quantitative results for baseline and augmented conditions.

4.2 Datasets and Tasks

The experiments are conducted on the graph-classification benchmarks DD, MSRC_21, and PROTEINS from the TUDataset collection [1]. All three datasets are loaded via PyTorch Geometric's `TUDataset` interface [17]. Each sample is an undirected graph $G = (V, E)$ with a graph label $y \in \{0, \ldots, C-1\}$ and an input node-feature matrix $X \in \mathbb{R}^{|V| \times d}$. DD and MSRC_21 provide discrete node labels but no continuous node at-

tributes, whereas PROTEINS additionally provides continuous node attributes [1, 3]. The underlying combinatorial graph G and the basic concepts of neighbourhoods, distances, and induced subgraphs are defined in Section 2.1. Basic structural statistics (numbers of graphs, classes, and average numbers of vertices and edges per graph) follow the TUDataset documentation and are summarised in Table 4.1 [1, 3].

Table 4.1: Basic statistics of the three TUDataset graph-classification benchmarks used in the experiments, following the TUDataset documentation [1, 3].

Dataset	#G	#C	Avg. |V|	Avg. |E|	Node labels	Node attributes
DD	1 178	2	284.32	715.66	yes	no
MSRC_21	563	20	77.52	198.32	yes	no
PROTEINS	1 113	2	39.06	72.82	yes	yes

Edge counts in Table 4.1 follow the TUDataset documentation convention. In PyTorch Geometric, undirected graphs are commonly represented with bi-directional edges in `edge_index`, so the stored number of edges can be twice the number of unique undirected edges.

For the two binary datasets DD and PROTEINS, the overall class distributions are moderately imbalanced: in DD, the majority class contains 691 of 1 178 graphs (approximately 0.59), and in PROTEINS it contains 663 of 1 113 graphs (approximately 0.60), as shown in Table 4.2. These counts are used later to motivate the augmentation strategy in Section 4.6, where the goal is to mitigate class imbalance by augmenting the smaller class more heavily. For MSRC_21, only fractional augmentation levels are considered, and the absolute per-class graph counts are not required for the subsequent analysis.

Table 4.2: Number of graphs per class for the binary TUDataset benchmarks DD and PROTEINS, computed from the TUDataset versions used in the experiments.

Dataset	Class	#Graphs
DD	0	691
	1	487
PROTEINS	0	663
	1	450

The three datasets instantiate two types of supervised graph-classification tasks: binary classification in DD and PROTEINS, and multi-class classification in MSRC_21. The following paragraphs describe the domain and label definitions of each dataset and its associated prediction task.

DD The DD dataset contains protein structures represented as graphs. According to Shervashidze et al. and the TUDataset documentation, each node represents an amino acid residue [1, 22]. Edges connect residues that are close in the folded protein (that is, close in 3D space) [1, 22]. The underlying enzyme-versus-non-enzyme prediction task originates

from the benchmark of Dobson and Doig [23]. In the graph-classification setting adopted by TUDataset, each graph is assigned a binary label indicating enzyme or non-enzyme [1].

PROTEINS The PROTEINS dataset models proteins as undirected graphs whose vertices represent secondary-structure elements (helices, sheets, and turns). Edges connect vertices if the corresponding elements are neighbours along the amino-acid sequence or among nearest neighbours in 3D space [24]. The dataset is derived from the enzyme-versus-non-enzyme benchmark of Dobson and Doig [1, 23]. In the TUDataset version, PROTEINS provides discrete node labels and continuous node attributes, which are encoded into the input feature matrix used by the GNN [1, 3].

MSRC_21 The MSRC_21 dataset is derived from the Microsoft Research Cambridge image database, which is commonly described as a 21-class image segmentation dataset and provides dense ground-truth labels for each image (for example, sky, grass, or building) [25]. In the graph benchmark distributed via TUDataset, each image is represented as a region-adjacency graph whose vertices correspond to image regions and whose edges connect regions that touch in the image [1, 26]. Although the original dataset is associated with 21 semantic classes, the TUDataset version used in these experiments provides $C = 20$ classes [3]. Accordingly, $C = 20$ is assumed for MSRC_21 throughout this thesis.

4.3 Model Architecture and Training Protocol

The experiments employ a single, fixed GCN classifier across all datasets and augmentation settings. This section specifies the architecture of this classifier, the training objective, and the protocol for repeating each configuration under multiple random seeds.

4.3.1 GCN Classifier Architecture

The classifier follows the GCN formulation introduced in Section 2.2, using the propagation rule of (2.1) and the permutation-invariant graph-level readout of (2.2). The network is intentionally simple, with few layers and a moderate size, so that observed performance differences can be attributed primarily to Graph WFC augmentation rather than to architectural complexity. Across all datasets and configurations, the classifier is instantiated as a standard, fully supervised GCN with the following structure:

- three graph-convolutional layers, each implementing the propagation rule of (2.1) using the `GCNConv` operator from PyTorch Geometric [17];
- hidden node embeddings of fixed dimension 64 in all intermediate layers;
- rectified linear units (ReLU) applied after each graph-convolutional layer;
- dropout with rate 0.3 applied to the node embeddings after each nonlinearity during training;

- a global mean pooling operator that aggregates the final-layer node embeddings into a graph-level representation $h_{\mathcal{G}}$ as in (2.2);
- a final linear layer that maps $h_{\mathcal{G}}$ to C real-valued class scores (logits).

For the binary classification tasks DD and PROTEINS, the output layer produces $C = 2$ logits corresponding to the two classes, and for the multi-class task on MSRC_21, the same architecture is used with $C = 20$ output logits. No architectural modifications tailored to specific datasets or configurations are introduced. The baseline and all Graph WFC augmented training sets use exactly the same GCN architecture.

4.3.2 Training Objective

Training is performed at the graph level using the standard cross-entropy loss $\mathcal{L}(\theta)$ on the graph-level logits and labels. No label smoothing or class-weighting is applied. The loss treats all classes symmetrically and reflects the natural class frequencies in the training data. Model parameters are optimised with the Adam optimiser, and the following hyperparameters are used uniformly across datasets (DD, MSRC_21, PROTEINS) and configurations (baseline and augmented):

- learning rate 10^{-3};
- ℓ_2 weight decay $5 \cdot 10^{-4}$;
- mini-batch size of 32 graphs.

Training proceeds for exactly 200 epochs and no training run is terminated early. A `ReduceLROnPlateau` learning-rate scheduler monitors validation accuracy (`mode=max`) and reduces the learning rate by a factor of 0.5 whenever validation accuracy has not improved for 10 consecutive epochs. During training, the implementation logs training loss and validation accuracy at each epoch. Model selection uses the checkpoint with the highest validation accuracy, and this checkpoint is used for the final evaluation on the held-out test graphs (or test fold in the cross-validation setting).

4.3.3 Random Seeds and Experimental Repetitions

To obtain robust performance estimates and quantify variability due to random initialisation and stochastic training effects, each experimental configuration is evaluated under multiple random seeds. A *configuration* is defined by several choices:

- the dataset;
- a Graph WFC augmentation setting, which covers the pattern radius and the size of the synthesised graphs as described in Section 3.2.1;
- an export setting that specifies how many training graphs are synthesised and added to the training set;

- all other hyperparameters that remain fixed throughout the experiments.

For each configuration with Graph WFC augmentation enabled, the synthetic training graphs are generated once and the resulting augmented training set is then kept fixed for all repetitions of that configuration. For a fixed dataset, configuration, and random seed (and, for MSRC_21, a specific cross-validation fold), a *training run* consists of the following steps:

- initialising the GCN parameters with the chosen seed;
- training for 200 epochs on the corresponding (possibly augmented) training set according to the fixed training objective, optimiser, and optimisation setup described above;
- selecting the checkpoint with the highest validation accuracy;
- evaluating this checkpoint on the associated held-out test graphs.

Each training run therefore produces one set of test metrics for a given configuration and seed (and fold).

For the datasets DD and PROTEINS, each configuration is evaluated under 100 independent random seeds, yielding 100 training runs per configuration. For these datasets, a single train/validation/test split is chosen once per dataset and reused across all seeds and configurations, so that only the random seed of the GCN training changes between runs. For MSRC_21, each configuration is evaluated under 20 independent random seeds. For each seed, a stratified 10-fold split is generated once and reused across baseline and augmented conditions for that seed, yielding 200 training runs per configuration (one per seed–fold combination).

For a fixed dataset and configuration, the full collection of its training runs is referred to as the *run ensemble* of that configuration. All reported performance values for a configuration are obtained by aggregating the test metrics over its run ensemble, where the tables in Section 4.6 report the mean and standard deviation of these metrics.

4.4 GCN Training and Graph WFC Augmentation Pipeline

This section presents the experimental pipeline that integrates Graph WFC with a fixed GCN classifier. The first step is to split the data into training, validation, and test sets, as described in Section 4.4.1. Training graphs are then encoded using a discrete-feature codec, enabling Graph WFC to operate on graphs with discrete vertex labels. Depending on an export setting, a collection of training graphs is selected and exported to Graph WFC in the required vertex-labelled format. Graph WFC is applied to this augmentation set, generating additional class-conditional training graphs, as explained in Section 4.4.2. These synthetic graphs are decoded back into graphs with vertex features and merged with the original training graphs to create the effective training set for the GCN, while the validation and test sets remain unchanged. The pipeline is visualised in Figure 4.1.

4.4.1 Data Preparation and Export

For the DD and PROTEINS datasets, evaluation is performed using a single stratified split, dividing the data into training (70%), validation (15%), and test (15%) sets. The split is generated once per dataset using a fixed random seed and reused for all random seeds and augmentation configurations. For MSRC_21, evaluation uses stratified 10-fold cross-validation with a new shuffle for each evaluation seed. For a fixed evaluation seed, the dataset is shuffled and partitioned into 10 stratified folds once, and this fold partition is reused across baseline and augmented conditions for that seed. In each fold, 10% of the graphs serve as test data. From the remaining graphs, 10% are held out as a stratified validation set, and the rest form the training set.

Given one such training set, all vertex-feature vectors from all training graphs are collected into the finite set

$$X_{\text{train}} := \{\, x_v \in \mathbb{R}^d \mid v \in V(G),\ G \text{ in the training set} \,\}.$$

A discrete codebook, referred to as the *feature codec*, is constructed as an injective mapping

$$c : X_{\text{train}} \to \mathcal{L},$$

where $\mathcal{L}$ is a finite label alphabet. For each training graph $G = (V, E, X, y)$, this mapping induces a vertex labelling $\ell_G : V \to \mathcal{L}$ as in definition 2.1.3 via

$$\ell_G(v) := c(x_v),$$

where x_v denotes the feature vector of vertex v. The resulting labelled graphs (V, E, ℓ_G, y) are undirected and unweighted, with a single discrete label on each vertex and no edge labels, matching the input requirements of Graph WFC. The codec is constructed from training graphs only and is recomputed whenever the training set changes.

The labelled training graphs are then grouped by their class labels. Let C denote the number of classes and let n_k^{train} be the number of available training graphs in class $k \in \{0, \dots, C-1\}$. An *export setting* specifies an *augmentation level* B, which determines how many training graphs of each class are selected and copied for export to Graph WFC. Two forms of B are used:

$$\begin{aligned} B &\in \big\{\mathrm{Frac}(\lambda) \mid \lambda \in \mathbb{R}_{>0}\big\} && \text{(fractional level)},\\ B &\in \big\{\mathrm{Count}(m_0, \dots, m_{C-1}) \mid m_k \in \mathbb{N}_0\big\} && \text{(per-class level)}. \end{aligned}$$

For each class k, define the requested export count as

$$m_k^{\text{exp}} := \begin{cases} \mathrm{round}\big(\lambda\, n_k^{\text{train}}\big), & \text{if } B = \mathrm{Frac}(\lambda),\\ m_k, & \text{if } B = \mathrm{Count}(m_0, \dots, m_{C-1}). \end{cases}$$

The exported multiset for class k is constructed from its n_k^{train} labelled training graphs as follows. If $m_k^{\text{exp}} \leq n_k^{\text{train}}$, sample m_k^{exp} graphs uniformly at random without replacement. If $m_k^{\text{exp}} > n_k^{\text{train}}$, write

$$m_k^{\text{exp}} \;=\; q\, n_k^{\text{train}} + r, \qquad q := \left\lfloor \frac{m_k^{\text{exp}}}{n_k^{\text{train}}} \right\rfloor, \qquad r \in \{0, \ldots, n_k^{\text{train}} - 1\}.$$

Then q full copies of the class-k training set are exported, and the remaining r graphs are sampled uniformly at random without replacement. Consequently, each training graph is exported either q or $q+1$ times. For the binary tasks, the shorthand $B = \text{Count}(m_0, m_1)$ is used.

4.4.2 Graph WFC Synthesis

On the augmentation set exported in Section 4.4.1, Graph WFC acts as a class-conditional generator that synthesises additional vertex-labelled training graphs. For each exported training graph in this set, Graph WFC attempts to generate one synthetic graph of the same class, as described in Chapter 3. Let n_{exp} denote the number of exported graphs (counting duplicates) and let n_{syn} denote the number of decoded synthetic graphs produced by Graph WFC. Since backtracking is not implemented, unsuccessful synthesis attempts are handled by restarting the synthesis from scratch. For each exported graph, up to 100 restarts are performed. If no terminating synthesis is obtained after these restarts, the corresponding attempt yields no output graph. Consequently, n_{syn} may be smaller than n_{exp}. To avoid evaluating unstable augmentation settings, a configuration is excluded from evaluation if more than 5% of the exported graphs fail to yield a terminating synthetic graph after 100 restarts, formally if

$$\frac{n_{\text{exp}} - n_{\text{syn}}}{n_{\text{exp}}} > 0.05. \tag{4.1}$$

Configurations that fail this exclusion process are omitted from the result tables in Section 4.6. In the experimental pipeline, each configuration uses a fixed pair of synthesis parameters

$$\gamma = (r, S),$$

which is shared by all runs of that configuration. The components r and S are defined as follows.

- *Pattern radius r.* The parameter r is the pattern radius used in the pattern-extraction stage of Section 3.1.2. In each configuration, r equals the maximal pattern radius $r_{\max}$ in $\mathcal{R}$ (see (3.9)).

- *Size range S.* The interval $S = [s_{\min}, s_{\max}] \subset \mathbb{R}_{>0}$ summarises the size band induced by the Graph WFC size parameters $(c_{\text{size}}, \delta_{\text{size}})$ in Section 3.2.1, where $s_{\min} := c_{\text{size}} - \delta_{\text{size}}$ and $s_{\max} := c_{\text{size}} + \delta_{\text{size}}$. For a training graph G_k with $n_k = |V_k|$, the target

is $N_k^{\text{tar}} = \lfloor c_{\text{size}}\, n_k \rfloor$ (3.16), the clean-up phase starts once $|\mathcal{X}_k| \geq N_k^{\text{low}} \approx s_{\min} n_k$ (3.17), and expansion is capped at $N_k^{\max} \approx s_{\max} n_k$ (3.18).

The export setting of Section 4.4.1 fixes the augmentation level B and thereby the augmentation set supplied to Graph WFC, while the pair $\gamma = (r, S)$ specifies how Graph WFC synthesises new graphs from that set in a given experimental configuration. In the result tables of Section 4.6, each entry summarises the repeated runs of one configuration (as defined in Section 4.3.3). The parameters B and γ there indicate the augmentation level and Graph WFC setting used for that configuration.

4.4.3 Data Import and Training

After Graph WFC has finished generating synthetic graphs, the resulting outputs are re-imported into the experimental pipeline. The Importer applies the inverse codec label by label to reconstruct vertex-feature vectors, thereby turning each vertex-labelled graph (V, E, ℓ, y) back into a sample (V, E, X, y) with vertex-feature matrix

$$X(v) := c^{-1}\big(\ell(v)\big) \quad \text{for all } v \in V.$$

All decoded graphs are collected as synthetic training graphs and retain the class labels produced by Graph WFC. The effective training set is then constructed by concatenating these synthetic graphs with the original training set defined in Section 4.4.1. In the baseline configuration, where no graphs are exported, and Graph WFC produces no synthetic graphs, the synthetic set is empty, and the effective training set is just the original training set. Finally, the GCN classifier of Section 4.3 is trained on this effective training set using the datasets and training setup specified in Sections 3.1, 4.2 and 4.3.

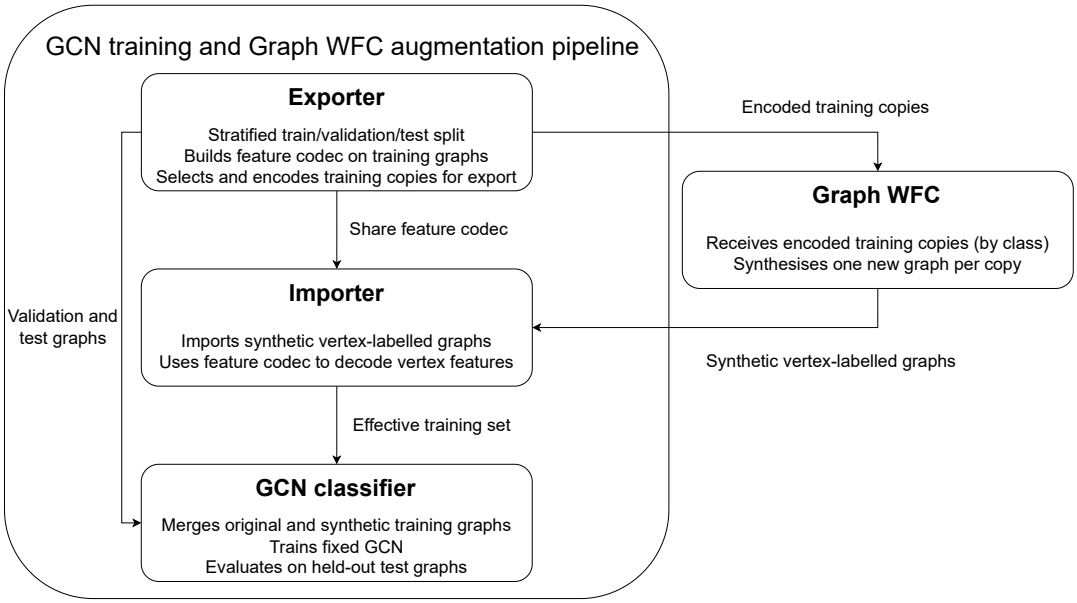

Figure 4.1: End-to-end experimental pipeline. The *Exporter* splits each dataset into training, validation, and test portions and encodes the training graphs using a shared feature codec. According to an export setting, a set of training graphs is passed to *Graph WFC*, which synthesises vertex-labelled graphs for each exported graph. The *Importer* decodes these graphs back into vertex-feature space and merges them with the original training graphs to form the effective training set. A fixed GCN classifier is trained on this set and evaluated on the held-out test graphs.

4.5 Evaluation and Aggregation

Given the trained GCN models obtained from the effective training sets of Section 4.4, this section specifies how performance is measured on held-out graphs and how the resulting scores are aggregated across random seeds and, for MSRC_21, cross-validation folds. The goal is to quantify, for each configuration, whether Graph WFC augmentation improves test performance relative to the baseline.

For a single training run in the sense of Section 4.3.3, the checkpoint corresponding to the epoch with the highest validation accuracy is evaluated once on the held-out test graphs (or on the test fold in the cross-validation setting). Let $(\mathcal{G}_i, y_i)_{i=1}^{N_{\text{test}}}$ denote the test graphs and their labels, and let $\hat{y}_i$ be the corresponding predicted labels produced by the GCN. The primary metric is graph classification accuracy,

$$\text{Acc} \;=\; \frac{1}{N_{\text{test}}} \sum_{i=1}^{N_{\text{test}}} \mathbf{1}\{\hat{y}_i = y_i\}, \tag{4.2}$$

where $\mathbf{1}\{\cdot\}$ denotes the indicator function. In addition to accuracy, macro-F1 and balanced accuracy are reported. To define them, each class $c \in \{0, \ldots, C-1\}$ is, in turn, treated as the positive class and all remaining classes as negative. This yields, for every class c,

- TP_c: number of true positives,

- FP_c: number of false positives,
- FN_c: number of false negatives,
- TN_c: number of true negatives.

Per-class precision, recall, and F1 score are then given by

$$\mathrm{Prec}_c = \frac{\mathrm{TP}_c}{\mathrm{TP}_c + \mathrm{FP}_c + \varepsilon}, \tag{4.3}$$

$$\mathrm{Rec}_c = \frac{\mathrm{TP}_c}{\mathrm{TP}_c + \mathrm{FN}_c + \varepsilon}, \tag{4.4}$$

$$\mathrm{F1}_c = \frac{2\,\mathrm{Prec}_c\,\mathrm{Rec}_c}{\mathrm{Prec}_c + \mathrm{Rec}_c + \varepsilon}, \tag{4.5}$$

where a small constant $\varepsilon > 0$ is added in the implementation to avoid division by zero in degenerate cases. These per-class quantities are aggregated into macro-averaged F1 and balanced accuracy:

$$\text{Macro-F1} = \frac{1}{C}\sum_{c=0}^{C-1} \mathrm{F1}_c, \tag{4.6}$$

$$\mathrm{BalAcc} = \frac{1}{C}\sum_{c=0}^{C-1} \mathrm{Rec}_c. \tag{4.7}$$

Macro-F1 gives equal weight to all classes and highlights performance on minority classes, while balanced accuracy compensates for class imbalance by averaging recall across classes.

To obtain robust performance estimates per configuration, the test metrics are aggregated over all training runs in the corresponding run ensemble (see Section 4.3.3). For a fixed dataset and configuration, let N_{runs} be the size of the run ensemble and let $m^{(r)}$ denote the value of a given test metric in run r, for $r = 1, \ldots, N_{\mathrm{runs}}$. The sample mean and standard deviation are

$$\bar{m} \;=\; \frac{1}{N_{\mathrm{runs}}}\sum_{r=1}^{N_{\mathrm{runs}}} m^{(r)}, \tag{4.8}$$

$$s \;=\; \sqrt{\frac{1}{N_{\mathrm{runs}}-1}\sum_{r=1}^{N_{\mathrm{runs}}} \left(m^{(r)} - \bar{m}\right)^2}. \tag{4.9}$$

To accompany these summary statistics, approximate 95% confidence intervals are reported using the normal approximation

$$\mathrm{CI}_{95\%} \;=\; \bar{m} \,\pm\, 1.96\,\frac{s}{\sqrt{N_{\mathrm{runs}}}}, \tag{4.10}$$

where $\bar{m}$ and s are the sample mean and standard deviation from Equations (4.8) and (4.9).

For each dataset and configuration, the tables in Section 4.6 report the mean and standard deviation of the chosen test metrics over the corresponding run ensemble.

4.6 Results

This section reports the empirical effect of Graph WFC augmentation on the supervised graph-classification performance of the GCN classifier described in Section 4.3. For each dataset, multiple augmentation configurations are evaluated, combining export settings B from Section 4.4.1 with Graph WFC synthesis parameters $\gamma = (r, S)$ from Section 4.4.2. Each configuration is trained and evaluated under the repetition protocol of Section 4.3.3, and all test metrics are computed and aggregated for the corresponding run ensemble as described in Section 4.5.

For each dataset, the configuration grid is specified by listing the ranges of r, S, and B used in the experiments. For each dataset and each evaluation metric, a separate table reports the baseline configuration alongside the four augmented configurations with the highest mean value for that metric. Within each table, the augmented configurations are ordered by their corresponding means in descending order.

In each metric column, scores are reported in the format

$$\bar{m} \pm s \; [\mathrm{CI}_{95\%}^{\mathrm{lower}}, \mathrm{CI}_{95\%}^{\mathrm{upper}}],$$

where $\bar{m}$ and s denote the sample mean and standard deviation of the corresponding metric over the run ensemble, as defined in Equations (4.8) and (4.9). The bracketed interval denotes the approximate 95% confidence interval $\mathrm{CI}_{95\%}$ from Equation (4.10), reported by its numerical lower and upper bounds. All interval comparisons are interpreted descriptively under the normal approximation in Equation (4.10), rather than as formal hypothesis tests. The full result tables, covering all combinations from the configuration grids, are presented in Appendix A.

4.6.1 MSRC_21

On MSRC_21, the following configuration grid is used:

$$\begin{aligned} r &\in \{1, 2, 3\}, \\ S &\in \{[0.9, 1.1]\}, \\ B &\in \big\{\mathrm{Frac}(\lambda) \;\big|\; \lambda \in \{0.5, 1.0, 1.5, 2.0, 2.5, 3.0\}\big\}. \end{aligned}$$

Table 4.3: Macro-F1 on MSRC_21

Config.	r	S	B	Macro-F1
Augmented	1	$[0.9, 1.1]$	Frac(2.0)	0.8512 ± 0.0120 [0.8459, 0.8564]
Augmented	1	$[0.9, 1.1]$	Frac(2.5)	0.8507 ± 0.0097 [0.8465, 0.8550]
Augmented	2	$[0.9, 1.1]$	Frac(3.0)	0.8487 ± 0.0090 [0.8448, 0.8527]
Augmented	2	$[0.9, 1.1]$	Frac(2.5)	0.8485 ± 0.0090 [0.8446, 0.8525]
Baseline	–	–	–	0.8302 ± 0.0133 [0.8244, 0.8361]

Table 4.4: Balanced accuracy on MSRC_21

Config.	r	S	B	Balanced accuracy
Augmented	1	$[0.9, 1.1]$	Frac(2.0)	0.8527 ± 0.0111 [0.8478, 0.8576]
Augmented	1	$[0.9, 1.1]$	Frac(2.5)	0.8526 ± 0.0099 [0.8483, 0.8570]
Augmented	1	$[0.9, 1.1]$	Frac(3.0)	0.8496 ± 0.0121 [0.8443, 0.8550]
Augmented	2	$[0.9, 1.1]$	Frac(3.0)	0.8478 ± 0.0089 [0.8439, 0.8517]
Baseline	–	–	–	0.8302 ± 0.0122 [0.8248, 0.8355]

Table 4.5: Test accuracy on MSRC_21

Config.	r	S	B	Test accuracy
Augmented	1	$[0.9, 1.1]$	Frac(2.5)	0.8791 ± 0.0094 [0.8750, 0.8832]
Augmented	1	$[0.9, 1.1]$	Frac(2.0)	0.8782 ± 0.0094 [0.8740, 0.8823]
Augmented	1	$[0.9, 1.1]$	Frac(3.0)	0.8758 ± 0.0108 [0.8711, 0.8806]
Augmented	3	$[0.9, 1.1]$	Frac(2.0)	0.8718 ± 0.0082 [0.8682, 0.8753]
Baseline	–	–	–	0.8532 ± 0.0122 [0.8478, 0.8586]

On MSRC_21, Graph WFC augmentation improves test accuracy over the baseline for all configurations reported in Table 4.5. The best result is obtained with $r = 1$ and $B = \text{Frac}(2.5)$, reaching a mean test accuracy of 0.8791 compared to 0.8532 for the baseline. This corresponds to an absolute improvement of 2.59 pp, where "pp" denotes percentage points. The reported 95% intervals $[0.8750, 0.8832]$ and $[0.8478, 0.8586]$ are disjoint under the normal approximation in Equation (4.10), which is consistent with a positive shift relative to the baseline. The remaining reported settings ($r = 1$ with $B \in \{\text{Frac}(2.0), \text{Frac}(3.0)\}$, and $r = 3$ with $B = \text{Frac}(2.0)$) also exceed the baseline mean, although Frac(3.0) and $r = 3$ are slightly weaker than the best $r = 1$ configuration.

Macro-F1 and balanced accuracy in Tables 4.3 and 4.4 exhibit the same qualitative behaviour. The highest macro-F1 is achieved with $r = 1$ and $B = \text{Frac}(2.0)$, increasing the mean from 0.8302 (baseline) to 0.8512 (an absolute improvement of 2.10 pp). Balanced accuracy is similarly maximised with $r = 1$ and $B = \text{Frac}(2.0)$, increasing the mean from 0.8302 to 0.8527 (an absolute improvement of 2.25 pp). For the top reported configurations in both metrics, the corresponding 95% intervals are disjoint from the baseline intervals under the normal approximation in Equation (4.10), which is consistent with a positive shift relative to the baseline.

Beyond the summary tables, the complete *test-accuracy* table in Table A.3 shows a clear dependence on the export level B. For each pattern radius $r \in \{1, 2, 3\}$, the smallest level $B = \text{Frac}(0.5)$ yields the lowest mean test accuracy among the augmented settings, while larger export levels tend to yield higher mean accuracy up to $B = \text{Frac}(2.0)$ or $B = \text{Frac}(2.5)$. The best mean test accuracy is attained at $r = 1$ with $B = \text{Frac}(2.5)$. Increasing to the maximum export level $B = \text{Frac}(3.0)$ does not improve further and can be slightly worse than the best intermediate settings (for example, for $r = 1$, 0.8758 at Frac(3.0) vs. 0.8791 at Frac(2.5)), indicating diminishing returns and mild non-monotonicity at the

upper end of the grid.

Across the full MSRC_21 tables for all three metrics, only one augmented configuration falls below the baseline in any entry: for macro-F1 at $r = 3$ with $B = \mathrm{Frac}(0.5)$, the mean is 0.8296 ± 0.0133 [0.8237, 0.8354] versus the baseline 0.8302 ± 0.0133 [0.8244, 0.8361], which is 0.06 pp below the baseline. The reported intervals overlap under the normal approximation in Equation (4.10), therefore the difference is not clearly separable from run-to-run variability.

4.6.2 PROTEINS

On PROTEINS, the following configuration grid is used:

$$
\begin{aligned}
&r \in \{1, 2, 3\},\\
&S \in \{[0.9, 1.1], [1.9, 2.1]\},\\
&B \in \big\{\mathrm{Frac}(\lambda) \;\big|\; \lambda \in \{0.5, 1.0, 1.5\}\big\} \cup \big\{\mathrm{Count}(166, 315)\big\}.
\end{aligned}
$$

Count-based export levels are included on PROTEINS to mitigate class imbalance by directly controlling the class composition of the exported augmentation set. Since PROTEINS is imbalanced (see Table 4.2), the count setting $B = \mathrm{Count}(166, 315)$ exports more graphs from the minority class (class 1) than from the majority class (class 0).

Table 4.6: Macro-F1 on PROTEINS

Config.	r	S	B	Macro-F1
Augmented	3	$[0.9, 1.1]$	$\mathrm{Frac}(1.5)$	0.6502 ± 0.0201 [0.6463, 0.6542]
Augmented	1	$[1.9, 2.1]$	$\mathrm{Count}(166, 315)$	0.6448 ± 0.0153 [0.6418, 0.6478]
Augmented	1	$[0.9, 1.1]$	$\mathrm{Frac}(1.0)$	0.6426 ± 0.0252 [0.6377, 0.6476]
Augmented	2	$[1.9, 2.1]$	$\mathrm{Count}(166, 315)$	0.6421 ± 0.0160 [0.6389, 0.6452]
Baseline	–	–	–	0.6364 ± 0.0189 [0.6327, 0.6401]

Table 4.7: Balanced accuracy on PROTEINS

Config.	r	S	B	Balanced accuracy
Augmented	3	$[0.9, 1.1]$	$\mathrm{Frac}(1.5)$	0.6493 ± 0.0173 [0.6459, 0.6527]
Augmented	1	$[1.9, 2.1]$	$\mathrm{Count}(166, 315)$	0.6429 ± 0.0142 [0.6401, 0.6457]
Augmented	1	$[0.9, 1.1]$	$\mathrm{Frac}(1.0)$	0.6417 ± 0.0216 [0.6374, 0.6459]
Augmented	3	$[1.9, 2.1]$	$\mathrm{Frac}(1.5)$	0.6404 ± 0.0207 [0.6363, 0.6444]
Baseline	–	–	–	0.6353 ± 0.0173 [0.6319, 0.6387]

Table 4.8: Test accuracy on PROTEINS

Config.	r	S	B	Test accuracy
Augmented	3	$[0.9, 1.1]$	Frac(1.5)	0.6893 ± 0.0173 [0.6860, 0.6927]
Augmented	1	$[0.9, 1.1]$	Frac(1.0)	0.6796 ± 0.0191 [0.6758, 0.6833]
Augmented	3	$[1.9, 2.1]$	Frac(0.5)	0.6787 ± 0.0178 [0.6752, 0.6822]
Augmented	3	$[1.9, 2.1]$	Frac(1.5)	0.6786 ± 0.0178 [0.6751, 0.6821]
Baseline	–	–	–	0.6720 ± 0.0154 [0.6690, 0.6750]

On PROTEINS, Graph WFC augmentation yields modest improvements over the baseline in the summary tables (Tables 4.6 to 4.8). For test accuracy (Table 4.8), the best reported configuration uses $r = 3$, $S = [0.9, 1.1]$, and $B = \text{Frac}(1.5)$, attaining 0.6893 compared to the baseline 0.6720. This corresponds to an absolute improvement of 1.73 pp. The reported 95% intervals $[0.6860, 0.6927]$ and $[0.6690, 0.6750]$ are disjoint under the normal approximation in Equation (4.10), which is consistent with a positive shift relative to the baseline. The next-best reported settings also exceed the baseline mean ($r = 1$, $S = [0.9, 1.1]$, $B = \text{Frac}(1.0)$: 0.6796, corresponding to 0.76 pp above baseline).

Macro-F1 and balanced accuracy in Tables 4.6 and 4.7 exhibit the same qualitative ranking at the top. The configuration $r = 3$, $S = [0.9, 1.1]$, $B = \text{Frac}(1.5)$ achieves the highest reported macro-F1, increasing the mean from 0.6364 (baseline) to 0.6502 (an absolute improvement of 1.38 pp), and the highest reported balanced accuracy, increasing the mean from 0.6353 to 0.6493 (an absolute improvement of 1.40 pp). In both cases, the reported intervals are disjoint from the baseline intervals under the normal approximation in Equation (4.10), which is consistent with a positive shift relative to the baseline. The count-based setting $B = \text{Count}(166, 315)$ also ranks among the stronger configurations for macro-F1 and balanced accuracy in the summary tables, but it does not match the best fractional setting.

Beyond the summary tables, the full *test-accuracy* table in Table A.6 shows that the effect of the fractional export level $B = \text{Frac}(\lambda)$ is configuration-dependent and not strictly monotone in λ: increasing λ does not consistently improve accuracy across the grid, and the relative ranking of λ varies with r and the size range S.

For the count-based export level $B = \text{Count}(166, 315)$, the full tables show a consistent but small uplift in macro-F1 and balanced accuracy relative to the baseline across all listed (r, S) combinations (see Tables A.4 and A.5). In test accuracy (Table A.6), only the configuration $r = 1$, $S = [1.9, 2.1]$, $B = \text{Count}(166, 315)$ slightly exceeds the baseline mean (0.6728 vs. 0.6720, an absolute difference of 0.08 pp), and the reported intervals overlap ($[0.6696, 0.6759]$ vs. $[0.6690, 0.6750]$). Overall, Graph WFC augmentation on PROTEINS can improve performance in selected configurations, but the gains are limited in magnitude and depend on the interaction between r, S, and B.

4.6.3 DD

On DD, the following configuration grid is used:

$$
\begin{aligned}
&r \in \{1\},\\
&S \in \{[0.9, 1.1], [1.9, 2.1]\},\\
&B \in \big\{\mathrm{Frac}(\lambda) \;\big|\; \lambda \in \{0.25, 0.5, 1.0\}\big\} \cup \big\{\mathrm{Count}(0, 142), \mathrm{Count}(199, 341)\big\}.
\end{aligned}
$$

Note that the pattern radius is fixed to $r = 1$, since configurations with $r > 1$ frequently failed to terminate and thus exceeded the exclusion threshold in Equation (4.1).

Count-based export levels are included on DD to mitigate class imbalance by directly controlling the class composition of the exported augmentation set. Since DD is imbalanced (see Table 4.2), the chosen count settings export more graphs from the minority class (class 1) than from the majority class (class 0).

Table 4.9: Macro-F1 on DD

Config.	r	S	B	Macro-F1
Augmented	1	$[0.9, 1.1]$	$\mathrm{Count}(199, 341)$	0.6873 ± 0.0157 [0.6842, 0.6904]
Augmented	1	$[1.9, 2.1]$	$\mathrm{Count}(199, 341)$	0.6714 ± 0.0120 [0.6691, 0.6738]
Augmented	1	$[0.9, 1.1]$	$\mathrm{Frac}(0.5)$	0.6713 ± 0.0156 [0.6683, 0.6744]
Augmented	1	$[0.9, 1.1]$	$\mathrm{Frac}(1.0)$	0.6621 ± 0.0180 [0.6586, 0.6656]
Baseline	–	–	–	0.6411 ± 0.0152 [0.6381, 0.6441]

Table 4.10: Balanced accuracy on DD

Config.	r	S	B	Balanced accuracy
Augmented	1	$[0.9, 1.1]$	$\mathrm{Count}(199, 341)$	0.6850 ± 0.0141 [0.6822, 0.6877]
Augmented	1	$[0.9, 1.1]$	$\mathrm{Frac}(0.5)$	0.6712 ± 0.0140 [0.6685, 0.6740]
Augmented	1	$[1.9, 2.1]$	$\mathrm{Count}(199, 341)$	0.6707 ± 0.0110 [0.6685, 0.6728]
Augmented	1	$[0.9, 1.1]$	$\mathrm{Frac}(0.25)$	0.6617 ± 0.0169 [0.6584, 0.6651]
Baseline	–	–	–	0.6432 ± 0.0121 [0.6408, 0.6456]

Table 4.11: Test accuracy on DD

Config.	r	S	B	Test accuracy
Augmented	1	$[0.9, 1.1]$	$\mathrm{Count}(199, 341)$	0.7182 ± 0.0210 [0.7141, 0.7224]
Augmented	1	$[0.9, 1.1]$	$\mathrm{Frac}(0.5)$	0.7124 ± 0.0135 [0.7098, 0.7151]
Augmented	1	$[1.9, 2.1]$	$\mathrm{Count}(199, 341)$	0.7085 ± 0.0136 [0.7058, 0.7111]
Augmented	1	$[0.9, 1.1]$	$\mathrm{Frac}(0.25)$	0.6938 ± 0.0123 [0.6914, 0.6962]
Baseline	–	–	–	0.6828 ± 0.0085 [0.6812, 0.6845]

On DD, Graph WFC augmentation improves test accuracy over the baseline for all configurations reported in Table 4.11. The best-performing setting uses $r = 1$, the near size-preserving range $S = [0.9, 1.1]$, and the count-based export level $B = \mathrm{Count}(199, 341)$.

It achieves a mean test accuracy of 0.7182 compared to 0.6828 for the baseline (an absolute improvement of 3.54 pp). The reported 95% intervals $[0.7141, 0.7224]$ and $[0.6812, 0.6845]$ are disjoint under the normal approximation in Equation (4.10), which is consistent with a positive shift relative to the baseline. A moderate fractional export level $B = \text{Frac}(0.5)$ with $S = [0.9, 1.1]$ also improves test accuracy to 0.7124 (2.96 pp), whereas the weaker setting $B = \text{Frac}(0.25)$ yields 0.6938 (1.10 pp).

In the full *test-accuracy* Table A.9, the fractional setting is not monotone in λ even when $S = [0.9, 1.1]$ is fixed: $B = \text{Frac}(1.0)$ attains 0.6911 (0.83 pp above baseline), which is lower than $B = \text{Frac}(0.25)$ (0.6938). For the count-based setting $B = \text{Count}(199, 341)$, increasing the size range to $S = [1.9, 2.1]$ reduces accuracy from 0.7182 to 0.7085 (a decrease of 0.97 pp), indicating that substantially larger synthetic graphs are less effective in this setting.

Macro-F1 and balanced accuracy in Tables 4.9 and 4.10 identify the same best-performing configuration. The configuration with $r = 1$, $S = [0.9, 1.1]$, and $B = \text{Count}(199, 341)$ achieves the highest reported mean macro-F1 and balanced accuracy. It improves macro-F1 from 0.6411 to 0.6873 (4.62 pp) and balanced accuracy from 0.6432 to 0.6850 (4.18 pp). Under the normal approximation in Equation (4.10), the corresponding 95% intervals $[0.6842, 0.6904]$ (macro-F1) and $[0.6822, 0.6877]$ (balanced accuracy) are disjoint from the baseline intervals $[0.6381, 0.6441]$ and $[0.6408, 0.6456]$. This is consistent with improvements in both class-balanced metrics as well as test accuracy. Since the count-based export levels $\text{Count}(0, 142)$ and $\text{Count}(199, 341)$ (see Table 4.2) explicitly control the class-wise export counts, the strongest DD results combine per-class export control with structural augmentation.

In the full DD tables (Tables A.7 to A.9), only the configuration $r = 1$, $S = [1.9, 2.1]$, and $B = \text{Frac}(1.0)$ is below the baseline in all three metrics. For this configuration, the reported intervals overlap with the baseline in all three metrics (see the corresponding table entries), indicating that the differences are small relative to run-to-run variability under the normal approximation in Equation (4.10).

Overall, the DD results suggest that Graph WFC augmentation is most effective when combined with count-based export control and near size-preserving synthesis, while substantially increasing the size range S reduces the benefit in the reported configurations.

Chapter 5

Conclusions and Future Work

5.1 Conclusion

This thesis addresses the research question (RQ) and goals G1–G2 in Section 1.5 by developing and evaluating Graph WFC as a data augmentation mechanism for GCN-based graph classification. The evaluation is conducted on three benchmark datasets, MSRC_21, DD, and PROTEINS, where MSRC_21 is a multi-class classification task, and DD and PROTEINS are moderately imbalanced binary classification tasks. With respect to RQ, the results indicate that Graph WFC augmentation can improve test-set performance, measured by accuracy, macro-F1, and balanced accuracy, relative to training on the original graphs only. However, the effect is not uniform. Both the direction and magnitude of the change depend on the dataset and on the chosen augmentation configuration.

For goal G1, Graph WFC was implemented for undirected, vertex-labelled graphs (Sections 3.1.2 and 4.4.2) and integrated into the training and evaluation pipeline (Section 4.3). The augmentation procedure is configured by the synthesis parameters $\gamma = (r, S)$ and the export setting B (Section 4.4.1), which jointly control pattern radius, synthetic graph range, and augmentation amount (including class-wise export in the count-based case).

For goal G2, the experiments in Section 4.6 characterise when augmentation is beneficial or detrimental relative to the baseline. On MSRC_21, multiple fractional export settings yield gains over the baseline, with the best configuration improving test accuracy by 2.59 pp (Section 4.6.1). Within the evaluated grid, test accuracy tends to be higher for intermediate-to-high fractional exports (notably $\lambda = 2.0$ or $\lambda = 2.5$), while $\lambda = 3.0$ does not improve further, indicating diminishing returns at the upper end. On DD, the greatest improvements occur when synthesis is near size-preserving and the export setting biases the augmentation set toward the minority class, with the best configuration improving test accuracy by 3.54 pp. On PROTEINS, the best observed gains are modest (up to 1.73 pp in test accuracy), and performance varies across the configuration grid without a consistent monotone trend in r, S, or B. Overall, the results suggest that practical effectiveness depends on the configuration as a whole. The relative impact of export strategy, target size range, and pattern radius varies across datasets, and parameter changes do not translate

into consistent improvements across the evaluated grids. Accordingly, settings that work well in one benchmark cannot be assumed to remain effective in another, and reported gains should be interpreted as configuration-specific rather than as evidence for a generally optimal regime. This pattern supports treating Graph WFC as a sensitive augmentation mechanism that requires empirical validation and tuning for each application, rather than as a plug-in method with robust default parameters. This sensitivity also manifests in synthesis stability: on DD, configurations with $r > 1$ were not evaluated because synthesis frequently exceeded the termination threshold Equation (4.1). This restriction limits conclusions about larger pattern radii and highlights that synthesis failures can constrain the usable parameter range.

Beyond synthesis stability, the empirical evidence is restricted to a single GCN architecture trained with the fixed protocol of Section 4.3, so transfer to other GNN architectures, training regimes, or graph domains is not established. Moreover, the analysis is limited to three standard classification metrics and does not characterise higher-order structural properties of the synthetic graphs. Consequently, the structural factors that drive improvements or degradations remain unresolved.

5.2 Future Work

The implementation and experiments in this thesis suggest several directions for extending Graph WFC and for better characterising its suitability for graph data augmentation.

A first line of work concerns algorithmic extensions of the generator. The current procedure in Section 3.2 relies on fixed heuristics for expansion, collapse, and connection, together with a fixed pattern-extraction protocol and the compatibility notion from Sections 3.1.2 and 3.1.3. Future work could study adaptive variants of the expansion cap $B_k^{\exp}$ from Equation (3.19), for instance by estimating suitable caps during generation rather than fixing them upfront. On the constraint side, alternative definitions of compatibility and multi-radius propagation (Section 3.2.4) could be explored, including weighted or score-based compatibility that incorporates motif statistics or other higher-order structure to more strongly penalise long-range inconsistencies. A further extension is to support richer attributed graphs, such as edge labels and continuous vertex features.

A second direction is to move beyond per-graph vocabularies. The current pipeline (Section 3.1) extracts a separate pattern vocabulary $\mathcal{P}_k^{(r)}$ and compatibility family $\mathcal{C}_k$ for each input graph and attempts to synthesise one graph per exported graph, without mixing information across different training graphs. Future work could construct shared pattern vocabularies by mining patterns across multiple graphs and learning a unified compatibility structure. This could enable synthesis that combines structural information across graphs, potentially increasing diversity and enabling controlled mixing of patterns from multiple training examples. Such a multi-graph variant may also provide auxiliary diagnostics, for example, by relating graphs through shared patterns or compatibility structure.

A third direction is to broaden the augmentation protocol and evaluation. The experiments in Section 4.6 consider a single GCN architecture (Sections 2.2 and 4.3) and three benchmarks. Future work could evaluate stronger GNN architectures and additional TUDataset benchmarks or other graph domains, and could extend the setting to other tasks such as node classification or link prediction. This would clarify whether the improvements observed on DD and MSRC_21 persist under different models and supervision signals, and whether degradation cases such as those observed on PROTEINS can be reduced by architectural or training changes.

A fourth direction is to study how the degree of structural fidelity induced by the generation constraints relates to performance. On MSRC_21, the strongest configurations in the evaluated grid use small pattern radii (Section 4.6.1), which restrict synthesis primarily through short-range constraints, whereas larger radii additionally enforce longer-range consistency. On DD, only $r = 1$ was evaluated because larger radii frequently exceeded the termination-based exclusion threshold (Equation (4.1)), so conclusions about larger pattern sizes are limited. Future work could quantify fidelity via structural statistics and systematically vary constraint strength, for example, by perturbing compatibility constraints or pattern statistics in a controlled manner.

Finally, additional diagnostics of the synthetic graphs would strengthen the empirical analysis. The current evaluation focuses on three standard classification metrics (Section 4.5) and does not quantify how synthetic graphs differ from the original data distribution. Future work could compare real and synthetic graphs via motif statistics, degree and distance distributions, spectral summaries, or graph kernels, and could analyse failure modes where augmentation degrades performance, particularly on PROTEINS. Such diagnostics would help identify parameter settings that balance fidelity and diversity and would relate structural deviations to observed performance changes.

Appendix A

Full Results

A.1 MSRC_21

Configuration grid as in Section 4.6.1:

Table A.1: Macro-F1 on MSRC_21

Config.	r	S	B	Macro-F1
Augmented	1	$[0.9, 1.1]$	Frac(2.0)	0.8512 ± 0.0120 [0.8459, 0.8564]
Augmented	1	$[0.9, 1.1]$	Frac(2.5)	0.8507 ± 0.0097 [0.8465, 0.8550]
Augmented	2	$[0.9, 1.1]$	Frac(3.0)	0.8487 ± 0.0090 [0.8448, 0.8527]
Augmented	2	$[0.9, 1.1]$	Frac(2.5)	0.8485 ± 0.0090 [0.8446, 0.8525]
Augmented	1	$[0.9, 1.1]$	Frac(3.0)	0.8475 ± 0.0127 [0.8419, 0.8530]
Augmented	2	$[0.9, 1.1]$	Frac(2.0)	0.8469 ± 0.0137 [0.8409, 0.8529]
Augmented	3	$[0.9, 1.1]$	Frac(2.0)	0.8463 ± 0.0131 [0.8405, 0.8520]
Augmented	3	$[0.9, 1.1]$	Frac(3.0)	0.8442 ± 0.0114 [0.8392, 0.8492]
Augmented	2	$[0.9, 1.1]$	Frac(1.5)	0.8440 ± 0.0098 [0.8397, 0.8483]
Augmented	2	$[0.9, 1.1]$	Frac(1.0)	0.8440 ± 0.0128 [0.8384, 0.8496]
Augmented	1	$[0.9, 1.1]$	Frac(1.5)	0.8430 ± 0.0139 [0.8369, 0.8491]
Augmented	1	$[0.9, 1.1]$	Frac(1.0)	0.8416 ± 0.0136 [0.8357, 0.8476]
Augmented	3	$[0.9, 1.1]$	Frac(1.5)	0.8408 ± 0.0142 [0.8346, 0.8471]
Augmented	3	$[0.9, 1.1]$	Frac(2.5)	0.8380 ± 0.0109 [0.8332, 0.8428]
Augmented	3	$[0.9, 1.1]$	Frac(1.0)	0.8371 ± 0.0119 [0.8319, 0.8423]
Augmented	1	$[0.9, 1.1]$	Frac(0.5)	0.8366 ± 0.0090 [0.8327, 0.8405]
Augmented	2	$[0.9, 1.1]$	Frac(0.5)	0.8354 ± 0.0156 [0.8285, 0.8422]
Baseline	–	–	–	0.8302 ± 0.0133 [0.8244, 0.8361]
Augmented	3	$[0.9, 1.1]$	Frac(0.5)	0.8296 ± 0.0133 [0.8237, 0.8354]

Table A.2: Balanced accuracy on MSRC_21

Config.	r	S	B	Balanced accuracy
Augmented	1	$[0.9, 1.1]$	Frac(2.0)	0.8527 ± 0.0111 [0.8478, 0.8576]
Augmented	1	$[0.9, 1.1]$	Frac(2.5)	0.8526 ± 0.0099 [0.8483, 0.8570]
Augmented	1	$[0.9, 1.1]$	Frac(3.0)	0.8496 ± 0.0121 [0.8443, 0.8550]
Augmented	2	$[0.9, 1.1]$	Frac(3.0)	0.8478 ± 0.0089 [0.8439, 0.8517]
Augmented	2	$[0.9, 1.1]$	Frac(2.5)	0.8478 ± 0.0090 [0.8439, 0.8518]
Augmented	2	$[0.9, 1.1]$	Frac(2.0)	0.8465 ± 0.0116 [0.8414, 0.8515]
Augmented	3	$[0.9, 1.1]$	Frac(2.0)	0.8461 ± 0.0107 [0.8414, 0.8508]
Augmented	1	$[0.9, 1.1]$	Frac(1.5)	0.8451 ± 0.0135 [0.8391, 0.8510]
Augmented	1	$[0.9, 1.1]$	Frac(1.0)	0.8439 ± 0.0128 [0.8382, 0.8495]
Augmented	3	$[0.9, 1.1]$	Frac(3.0)	0.8438 ± 0.0106 [0.8392, 0.8484]
Augmented	2	$[0.9, 1.1]$	Frac(1.0)	0.8437 ± 0.0116 [0.8386, 0.8488]
Augmented	2	$[0.9, 1.1]$	Frac(1.5)	0.8434 ± 0.0090 [0.8395, 0.8474]
Augmented	3	$[0.9, 1.1]$	Frac(1.5)	0.8410 ± 0.0124 [0.8355, 0.8464]
Augmented	3	$[0.9, 1.1]$	Frac(2.5)	0.8383 ± 0.0094 [0.8342, 0.8424]
Augmented	3	$[0.9, 1.1]$	Frac(1.0)	0.8376 ± 0.0114 [0.8326, 0.8426]
Augmented	1	$[0.9, 1.1]$	Frac(0.5)	0.8374 ± 0.0096 [0.8332, 0.8415]
Augmented	2	$[0.9, 1.1]$	Frac(0.5)	0.8356 ± 0.0147 [0.8292, 0.8421]
Augmented	3	$[0.9, 1.1]$	Frac(0.5)	0.8307 ± 0.0115 [0.8256, 0.8357]
Baseline	–	–	–	0.8302 ± 0.0122 [0.8248, 0.8355]

Table A.3: Test accuracy on MSRC_21

Config.	r	S	B	Test accuracy
Augmented	1	$[0.9, 1.1]$	Frac(2.5)	0.8791 ± 0.0094 [0.8750, 0.8832]
Augmented	1	$[0.9, 1.1]$	Frac(2.0)	0.8782 ± 0.0094 [0.8740, 0.8823]
Augmented	1	$[0.9, 1.1]$	Frac(3.0)	0.8758 ± 0.0108 [0.8711, 0.8806]
Augmented	3	$[0.9, 1.1]$	Frac(2.0)	0.8718 ± 0.0082 [0.8682, 0.8753]
Augmented	1	$[0.9, 1.1]$	Frac(1.5)	0.8717 ± 0.0121 [0.8664, 0.8770]
Augmented	1	$[0.9, 1.1]$	Frac(1.0)	0.8712 ± 0.0120 [0.8660, 0.8765]
Augmented	2	$[0.9, 1.1]$	Frac(2.5)	0.8709 ± 0.0080 [0.8674, 0.8744]
Augmented	3	$[0.9, 1.1]$	Frac(3.0)	0.8701 ± 0.0100 [0.8657, 0.8745]
Augmented	2	$[0.9, 1.1]$	Frac(3.0)	0.8700 ± 0.0069 [0.8669, 0.8730]
Augmented	2	$[0.9, 1.1]$	Frac(2.0)	0.8696 ± 0.0080 [0.8661, 0.8731]
Augmented	2	$[0.9, 1.1]$	Frac(1.0)	0.8687 ± 0.0103 [0.8641, 0.8732]
Augmented	2	$[0.9, 1.1]$	Frac(1.5)	0.8679 ± 0.0090 [0.8639, 0.8718]
Augmented	3	$[0.9, 1.1]$	Frac(1.5)	0.8673 ± 0.0103 [0.8628, 0.8718]
Augmented	3	$[0.9, 1.1]$	Frac(2.5)	0.8654 ± 0.0078 [0.8620, 0.8688]
Augmented	3	$[0.9, 1.1]$	Frac(1.0)	0.8647 ± 0.0106 [0.8601, 0.8694]
Augmented	1	$[0.9, 1.1]$	Frac(0.5)	0.8624 ± 0.0106 [0.8578, 0.8671]
Augmented	2	$[0.9, 1.1]$	Frac(0.5)	0.8605 ± 0.0128 [0.8549, 0.8661]
Augmented	3	$[0.9, 1.1]$	Frac(0.5)	0.8576 ± 0.0096 [0.8534, 0.8619]
Baseline	–	–	–	0.8532 ± 0.0122 [0.8478, 0.8586]

A.2 PROTEINS

Configuration grid as in Section 4.6.2:

Table A.4: Macro-F1 on PROTEINS

Config.	r	S	B	Macro-F1
Augmented	3	$[0.9, 1.1]$	Frac(1.5)	0.6502 ± 0.0201 [0.6463, 0.6542]
Augmented	1	$[1.9, 2.1]$	Count(166, 315)	0.6448 ± 0.0153 [0.6418, 0.6478]
Augmented	1	$[0.9, 1.1]$	Frac(1.0)	0.6426 ± 0.0252 [0.6377, 0.6476]
Augmented	2	$[1.9, 2.1]$	Count(166, 315)	0.6421 ± 0.0160 [0.6389, 0.6452]
Augmented	3	$[1.9, 2.1]$	Count(166, 315)	0.6414 ± 0.0141 [0.6386, 0.6441]
Augmented	3	$[1.9, 2.1]$	Frac(1.5)	0.6410 ± 0.0242 [0.6362, 0.6457]
Augmented	2	$[0.9, 1.1]$	Frac(1.0)	0.6405 ± 0.0253 [0.6356, 0.6455]
Augmented	2	$[0.9, 1.1]$	Count(166, 315)	0.6390 ± 0.0208 [0.6349, 0.6430]
Augmented	3	$[1.9, 2.1]$	Frac(1.0)	0.6386 ± 0.0217 [0.6344, 0.6429]
Augmented	3	$[1.9, 2.1]$	Frac(0.5)	0.6379 ± 0.0179 [0.6344, 0.6414]
Augmented	3	$[0.9, 1.1]$	Count(166, 315)	0.6376 ± 0.0176 [0.6342, 0.6411]
Augmented	1	$[0.9, 1.1]$	Count(166, 315)	0.6371 ± 0.0173 [0.6337, 0.6405]
Baseline	–	–	–	0.6364 ± 0.0189 [0.6327, 0.6401]
Augmented	1	$[1.9, 2.1]$	Frac(1.0)	0.6351 ± 0.0243 [0.6304, 0.6399]
Augmented	3	$[0.9, 1.1]$	Frac(0.5)	0.6350 ± 0.0184 [0.6314, 0.6386]
Augmented	1	$[1.9, 2.1]$	Frac(0.5)	0.6345 ± 0.0201 [0.6306, 0.6385]
Augmented	3	$[0.9, 1.1]$	Frac(1.0)	0.6332 ± 0.0264 [0.6280, 0.6383]
Augmented	2	$[1.9, 2.1]$	Frac(1.0)	0.6331 ± 0.0270 [0.6278, 0.6383]
Augmented	2	$[0.9, 1.1]$	Frac(0.5)	0.6325 ± 0.0213 [0.6283, 0.6366]
Augmented	2	$[1.9, 2.1]$	Frac(0.5)	0.6310 ± 0.0218 [0.6268, 0.6353]
Augmented	1	$[0.9, 1.1]$	Frac(1.5)	0.6291 ± 0.0208 [0.6250, 0.6332]
Augmented	1	$[1.9, 2.1]$	Frac(1.5)	0.6254 ± 0.0216 [0.6212, 0.6297]
Augmented	1	$[0.9, 1.1]$	Frac(0.5)	0.6250 ± 0.0183 [0.6214, 0.6286]
Augmented	2	$[0.9, 1.1]$	Frac(1.5)	0.6184 ± 0.0195 [0.6146, 0.6222]
Augmented	2	$[1.9, 2.1]$	Frac(1.5)	0.6148 ± 0.0229 [0.6103, 0.6192]

Table A.5: Balanced accuracy on PROTEINS

Config.	r	S	B	Balanced accuracy
Augmented	3	$[0.9, 1.1]$	Frac(1.5)	0.6493 ± 0.0173 [0.6459, 0.6527]
Augmented	1	$[1.9, 2.1]$	Count(166, 315)	0.6429 ± 0.0142 [0.6401, 0.6457]
Augmented	1	$[0.9, 1.1]$	Frac(1.0)	0.6417 ± 0.0216 [0.6374, 0.6459]
Augmented	3	$[1.9, 2.1]$	Frac(1.5)	0.6404 ± 0.0207 [0.6363, 0.6444]
Augmented	2	$[1.9, 2.1]$	Count(166, 315)	0.6402 ± 0.0148 [0.6373, 0.6431]
Augmented	2	$[0.9, 1.1]$	Frac(1.0)	0.6397 ± 0.0220 [0.6354, 0.6440]
Augmented	3	$[1.9, 2.1]$	Count(166, 315)	0.6395 ± 0.0133 [0.6369, 0.6421]
Augmented	3	$[1.9, 2.1]$	Frac(1.0)	0.6383 ± 0.0191 [0.6346, 0.6420]
Augmented	3	$[1.9, 2.1]$	Frac(0.5)	0.6376 ± 0.0166 [0.6344, 0.6409]
Augmented	2	$[0.9, 1.1]$	Count(166, 315)	0.6375 ± 0.0192 [0.6337, 0.6413]
Augmented	3	$[0.9, 1.1]$	Count(166, 315)	0.6361 ± 0.0158 [0.6330, 0.6392]
Augmented	1	$[0.9, 1.1]$	Count(166, 315)	0.6356 ± 0.0160 [0.6324, 0.6387]
Augmented	1	$[1.9, 2.1]$	Frac(1.0)	0.6354 ± 0.0210 [0.6313, 0.6395]
Baseline	–	–	–	0.6353 ± 0.0173 [0.6319, 0.6387]
Augmented	3	$[0.9, 1.1]$	Frac(0.5)	0.6348 ± 0.0167 [0.6316, 0.6381]
Augmented	1	$[1.9, 2.1]$	Frac(0.5)	0.6338 ± 0.0178 [0.6303, 0.6373]
Augmented	3	$[0.9, 1.1]$	Frac(1.0)	0.6335 ± 0.0223 [0.6291, 0.6379]
Augmented	2	$[1.9, 2.1]$	Frac(1.0)	0.6329 ± 0.0237 [0.6283, 0.6376]
Augmented	2	$[0.9, 1.1]$	Frac(0.5)	0.6323 ± 0.0187 [0.6286, 0.6359]
Augmented	2	$[1.9, 2.1]$	Frac(0.5)	0.6309 ± 0.0194 [0.6271, 0.6347]
Augmented	1	$[0.9, 1.1]$	Frac(1.5)	0.6305 ± 0.0181 [0.6270, 0.6341]
Augmented	1	$[1.9, 2.1]$	Frac(1.5)	0.6276 ± 0.0182 [0.6241, 0.6312]
Augmented	1	$[0.9, 1.1]$	Frac(0.5)	0.6253 ± 0.0165 [0.6221, 0.6285]
Augmented	2	$[0.9, 1.1]$	Frac(1.5)	0.6232 ± 0.0162 [0.6201, 0.6264]
Augmented	2	$[1.9, 2.1]$	Frac(1.5)	0.6182 ± 0.0196 [0.6144, 0.6221]

Table A.6: Test accuracy on PROTEINS

Config.	r	S	B	Test accuracy
Augmented	3	$[0.9, 1.1]$	Frac(1.5)	0.6893 ± 0.0173 [0.6860, 0.6927]
Augmented	1	$[0.9, 1.1]$	Frac(1.0)	0.6796 ± 0.0191 [0.6758, 0.6833]
Augmented	3	$[1.9, 2.1]$	Frac(0.5)	0.6787 ± 0.0178 [0.6752, 0.6822]
Augmented	3	$[1.9, 2.1]$	Frac(1.5)	0.6786 ± 0.0178 [0.6751, 0.6821]
Augmented	3	$[1.9, 2.1]$	Frac(1.0)	0.6771 ± 0.0205 [0.6731, 0.6811]
Augmented	2	$[0.9, 1.1]$	Frac(1.0)	0.6757 ± 0.0177 [0.6722, 0.6792]
Augmented	3	$[0.9, 1.1]$	Frac(0.5)	0.6753 ± 0.0158 [0.6722, 0.6784]
Augmented	1	$[0.9, 1.1]$	Frac(1.5)	0.6749 ± 0.0158 [0.6718, 0.6779]
Augmented	1	$[1.9, 2.1]$	Frac(1.0)	0.6748 ± 0.0178 [0.6713, 0.6783]
Augmented	3	$[0.9, 1.1]$	Frac(1.0)	0.6734 ± 0.0176 [0.6700, 0.6769]
Augmented	1	$[1.9, 2.1]$	Count(166, 315)	0.6728 ± 0.0160 [0.6696, 0.6759]
Augmented	2	$[0.9, 1.1]$	Frac(1.5)	0.6728 ± 0.0142 [0.6700, 0.6755]
Baseline	–	–	–	0.6720 ± 0.0154 [0.6690, 0.6750]
Augmented	1	$[1.9, 2.1]$	Frac(1.5)	0.6710 ± 0.0153 [0.6680, 0.6740]
Augmented	2	$[1.9, 2.1]$	Frac(1.0)	0.6710 ± 0.0195 [0.6671, 0.6748]
Augmented	2	$[0.9, 1.1]$	Frac(0.5)	0.6709 ± 0.0156 [0.6678, 0.6739]
Augmented	2	$[1.9, 2.1]$	Count(166, 315)	0.6706 ± 0.0134 [0.6680, 0.6732]
Augmented	1	$[1.9, 2.1]$	Frac(0.5)	0.6696 ± 0.0156 [0.6666, 0.6727]
Augmented	3	$[0.9, 1.1]$	Count(166, 315)	0.6693 ± 0.0128 [0.6668, 0.6718]
Augmented	2	$[1.9, 2.1]$	Frac(0.5)	0.6690 ± 0.0158 [0.6659, 0.6721]
Augmented	3	$[1.9, 2.1]$	Count(166, 315)	0.6690 ± 0.0138 [0.6663, 0.6717]
Augmented	2	$[0.9, 1.1]$	Count(166, 315)	0.6682 ± 0.0173 [0.6648, 0.6716]
Augmented	1	$[0.9, 1.1]$	Count(166, 315)	0.6654 ± 0.0164 [0.6622, 0.6686]
Augmented	1	$[0.9, 1.1]$	Frac(0.5)	0.6649 ± 0.0143 [0.6621, 0.6677]
Augmented	2	$[1.9, 2.1]$	Frac(1.5)	0.6640 ± 0.0162 [0.6608, 0.6671]

A.3 DD

Configuration grid as in Section 4.6.3:

Table A.7: Macro-F1 on DD

Config.	r	S	B	Macro-F1
Augmented	1	$[0.9, 1.1]$	Count(199, 341)	0.6873 ± 0.0157 [0.6842, 0.6904]
Augmented	1	$[1.9, 2.1]$	Count(199, 341)	0.6714 ± 0.0120 [0.6691, 0.6738]
Augmented	1	$[0.9, 1.1]$	Frac(0.5)	0.6713 ± 0.0156 [0.6683, 0.6744]
Augmented	1	$[0.9, 1.1]$	Frac(1.0)	0.6621 ± 0.0180 [0.6586, 0.6656]
Augmented	1	$[0.9, 1.1]$	Frac(0.25)	0.6617 ± 0.0210 [0.6576, 0.6658]
Augmented	1	$[0.9, 1.1]$	Count(0, 142)	0.6614 ± 0.0153 [0.6584, 0.6644]
Augmented	1	$[1.9, 2.1]$	Count(0, 142)	0.6609 ± 0.0238 [0.6563, 0.6656]
Augmented	1	$[1.9, 2.1]$	Frac(0.5)	0.6532 ± 0.0152 [0.6511, 0.6553]
Augmented	1	$[1.9, 2.1]$	Frac(0.25)	0.6518 ± 0.0120 [0.6494, 0.6541]
Baseline	–	–	–	0.6411 ± 0.0152 [0.6381, 0.6441]
Augmented	1	$[1.9, 2.1]$	Frac(1.0)	0.6381 ± 0.0210 [0.6340, 0.6422]

Table A.8: Balanced accuracy on DD

Config.	r	S	B	Balanced accuracy
Augmented	1	$[0.9, 1.1]$	Count$(199, 341)$	0.6850 ± 0.0141 [0.6822, 0.6877]
Augmented	1	$[0.9, 1.1]$	Frac(0.5)	0.6712 ± 0.0140 [0.6685, 0.6740]
Augmented	1	$[1.9, 2.1]$	Count$(199, 341)$	0.6707 ± 0.0110 [0.6685, 0.6728]
Augmented	1	$[0.9, 1.1]$	Frac(0.25)	0.6617 ± 0.0169 [0.6584, 0.6651]
Augmented	1	$[0.9, 1.1]$	Count$(0, 142)$	0.6613 ± 0.0142 [0.6585, 0.6640]
Augmented	1	$[0.9, 1.1]$	Frac(1.0)	0.6607 ± 0.0162 [0.6575, 0.6638]
Augmented	1	$[1.9, 2.1]$	Count$(0, 142)$	0.6605 ± 0.0214 [0.6563, 0.6647]
Augmented	1	$[1.9, 2.1]$	Frac(0.5)	0.6529 ± 0.0137 [0.6510, 0.6548]
Augmented	1	$[1.9, 2.1]$	Frac(0.25)	0.6509 ± 0.0106 [0.6488, 0.6529]
Baseline	–	–	–	0.6432 ± 0.0121 [0.6408, 0.6456]
Augmented	1	$[1.9, 2.1]$	Frac(1.0)	0.6412 ± 0.0162 [0.6380, 0.6444]

Table A.9: Test accuracy on DD

Config.	r	S	B	Test accuracy
Augmented	1	$[0.9, 1.1]$	Count$(199, 341)$	0.7182 ± 0.0210 [0.7141, 0.7224]
Augmented	1	$[0.9, 1.1]$	Frac(0.5)	0.7124 ± 0.0135 [0.7098, 0.7151]
Augmented	1	$[1.9, 2.1]$	Count$(199, 341)$	0.7085 ± 0.0136 [0.7058, 0.7111]
Augmented	1	$[0.9, 1.1]$	Frac(0.25)	0.6938 ± 0.0123 [0.6914, 0.6962]
Augmented	1	$[0.9, 1.1]$	Frac(1.0)	0.6911 ± 0.0136 [0.6885, 0.6938]
Augmented	1	$[1.9, 2.1]$	Count$(0, 142)$	0.6897 ± 0.0306 [0.6837, 0.6957]
Augmented	1	$[1.9, 2.1]$	Frac(0.5)	0.6886 ± 0.0133 [0.6868, 0.6905]
Augmented	1	$[0.9, 1.1]$	Count$(0, 142)$	0.6852 ± 0.0217 [0.6809, 0.6895]
Augmented	1	$[1.9, 2.1]$	Frac(0.25)	0.6842 ± 0.0086 [0.6825, 0.6859]
Baseline	–	–	–	0.6828 ± 0.0085 [0.6812, 0.6845]
Augmented	1	$[1.9, 2.1]$	Frac(1.0)	0.6812 ± 0.0137 [0.6785, 0.6839]

Bibliography

[1] Christopher Morris et al. "TUDataset: A collection of benchmark datasets for learning with graphs". In: *CoRR* abs/2007.08663 (2020). arXiv: 2007.08663.

[2] Shirui Pan and Xingquan Zhu. "Graph Classification with Imbalanced Class Distributions and Noise". In: *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013*. IJCAI/AAAI, 2013, pp. 1586–1592. URL: https://www.ijcai.org/Proceedings/13/Papers/236.pdf.

[3] TU Dortmund University. *Datasets | TUDataset*. Statistics table. 2023. URL: https://chrsmrrs.github.io/datasets/docs/datasets/ (visited on 10/19/2025).

[4] Marisa Thoma et al. "Near-optimal Supervised Feature Selection among Frequent Subgraphs". In: *Proceedings of the SIAM International Conference on Data Mining, SDM 2009, April 30 - May 2, 2009, Sparks, Nevada, USA*. SIAM, 2009, pp. 1076–1087. DOI: 10.1137/1.9781611972795.92.

[5] Zemin Liu et al. "A Survey of Imbalanced Learning on Graphs: Problems, Techniques, and Future Directions". In: *IEEE Trans. Knowl. Data Eng.* 37.6 (2025), pp. 3132–3152. DOI: 10.1109/TKDE.2025.3549299.

[6] Thomas N. Kipf and Max Welling. "Semi-Supervised Classification with Graph Convolutional Networks". In: *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24–26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL: https://openreview.net/forum?id=SJU4ayYgl.

[7] Tong Zhao et al. "Graph Data Augmentation for Graph Machine Learning: A Survey". In: *IEEE Data Eng. Bull.* 46.2 (2023), pp. 140–165. URL: http://sites.computer.org/debull/A23june/p140.pdf.

[8] Maxim Gumin. *WaveFunctionCollapse*. Computer software (GitHub repository). 2016. URL: https://github.com/mxgmn/WaveFunctionCollapse (visited on 11/27/2025).

[9] Isaac Karth and Adam M. Smith. "WaveFunctionCollapse is constraint solving in the wild". In: *Proceedings of the International Conference on the Foundations of Digital Games, FDG 2017, Hyannis, MA, USA, August 14–17, 2017*. ACM, 2017, 68:1–68:10. DOI: 10.1145/3102071.3110566.

[10] Ben Dzaebel. “Erweiterung des Wave Function Collapse Algorithmus zur Synthese von 3D-Voxel-Modellen anhand von nutzergenerierten Beispielen”. Bachelor’s thesis. Hamburg, Germany: Hochschule für Angewandte Wissenschaften Hamburg, 2020. URL: https://users.informatik.haw-hamburg.de/~infwrb267/abschlussarbeiten/ba_dzaebel.pdf (visited on 11/27/2025).

[11] Aladar Apponyi. *Wave Function Collapse on a Hex Grid*. Computer software (GitHub repository). July 8, 2020. URL: https://github.com/papadar/wavefunctionhexgridGM/tree/99b2c8a (visited on 11/27/2025).

[12] Reinhard Diestel. *Graph Theory*. 5th ed. Vol. 173. Graduate Texts in Mathematics. Berlin, Heidelberg: Springer, 2017. DOI: 10.1007/978-3-662-53622-3.

[13] Douglas B. West. *Introduction to Graph Theory*. 2nd ed. Upper Saddle River, NJ: Prentice Hall, 2001. ISBN: 0-13-014400-2.

[14] Russell Lyons and Yuval Peres. *Probability on Trees and Networks*. Vol. 42. Cambridge Series in Statistical and Probabilistic Mathematics. New York, NY: Cambridge University Press, 2016. DOI: 10.1017/9781316672815.

[15] Peter M. Winkler. “Existence of graphs with a given set of r-neighborhoods”. In: *J. Comb. Theory B* 34.2 (1983), pp. 165–176. DOI: 10.1016/0095-8956(83)90016-3.

[16] László Lovász. *Large Networks and Graph Limits*. Vol. 60. Colloquium Publications. Providence, RI: American Mathematical Society, 2012. ISBN: 978-0-8218-9085-1. DOI: 10.1090/coll/060.

[17] Matthias Fey and Jan Eric Lenssen. “Fast Graph Representation Learning with PyTorch Geometric”. In: *CoRR* abs/1903.02428 (2019). arXiv: 1903.02428.

[18] Jie Zhou et al. “Graph neural networks: A review of methods and applications”. In: *AI Open* 1 (2020), pp. 57–81. DOI: 10.1016/j.aiopen.2021.01.001.

[19] Stephen Sherratt. *Procedural Generation with Wave Function Collapse*. Blog post. Feb. 21, 2019. URL: https://www.gridbugs.org/wave-function-collapse/ (visited on 11/29/2025).

[20] Aleksandar Bojchevski et al. “NetGAN: Generating Graphs via Random Walks”. In: *Proceedings of the 35th International Conference on Machine Learning*. Vol. 80. Proceedings of Machine Learning Research. PMLR, 2018, pp. 609–618. URL: http://proceedings.mlr.press/v80/bojchevski18a.html.

[21] Itai Benjamini and Oded Schramm. “Recurrence of Distributional Limits of Finite Planar Graphs”. In: *Electron. J. Probab.* 6.23 (2001), pp. 1–13. DOI: 10.1214/EJP.v6-96.

[22] Nino Shervashidze et al. “Weisfeiler-Lehman Graph Kernels”. In: *J. Mach. Learn. Res.* 12 (2011), pp. 2539–2561. DOI: 10.5555/1953048.2078187.

[23] Paul D. Dobson and Andrew J. Doig. “Distinguishing enzyme structures from non-enzymes without alignments”. In: *Journal of Molecular Biology* 330.4 (2003), pp. 771–783. DOI: 10.1016/S0022-2836(03)00628-4.

[24] Karsten M. Borgwardt et al. “Protein function prediction via graph kernels”. In: *Bioinformatics* 21.suppl_1 (2005), pp. i47–i56. DOI: 10.1093/bioinformatics/bti1007.

[25] Jamie Shotton et al. “TextonBoost for Image Understanding: Multi-Class Object Recognition and Segmentation by Jointly Modeling Texture, Layout, and Context”. In: *Int. J. Comput. Vis.* 81.1 (2009), pp. 2–23. DOI: 10.1007/s11263-007-0109-1.

[26] Marion Neumann et al. “Propagation kernels: efficient graph kernels from propagated information”. In: *Mach. Learn.* 102.2 (2016), pp. 209–245. DOI: 10.1007/s10994-015-5517-9.

Erklärung

gemäss Art. 30 RSL Phil.-nat.18

Name/Vorname:	Brubacher Marco
Matrikelnummer:	21-124-441
Studiengang:	BSc Informatik
	Bachelor ☑ Master ☐ Dissertation ☐
Titel der Arbeit:	Graph-Based Wave Function Collapse for Data Augmentation in Graph Classification An Experimental Evaluation Using a Graph Convolutional Network
LeiterIn der Arbeit:	PD Dr. Kaspar Riesen

Ich erkläre hiermit, dass ich diese Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen benutzt habe. Alle Stellen, die wörtlich oder sinngemäss aus Quellen entnommen wurden, habe ich als solche gekennzeichnet. Mir ist bekannt, dass andernfalls der Senat gemäss Artikel 36 Absatz 1 Buchstabe r des Gesetzes vom 5. September 1996 über die Universität zum Entzug des auf Grund dieser Arbeit verliehenen Titels berechtigt ist.
Für die Zwecke der Begutachtung und der Überprüfung der Einhaltung der Selbständigkeitserklärung bzw. der Reglemente betreffend Plagiate erteile ich der Universität Bern das Recht, die dazu erforderlichen Personendaten zu bearbeiten und Nutzungshandlungen vorzunehmen, insbesondere die schriftliche Arbeit zu vervielfältigen und dauerhaft in einer Datenbank zu speichern sowie diese zur Überprüfung von Arbeiten Dritter zu verwenden oder hierzu zur Verfügung zu stellen.

Bern 17.12.2025

Ort/Datum

Unterschrift