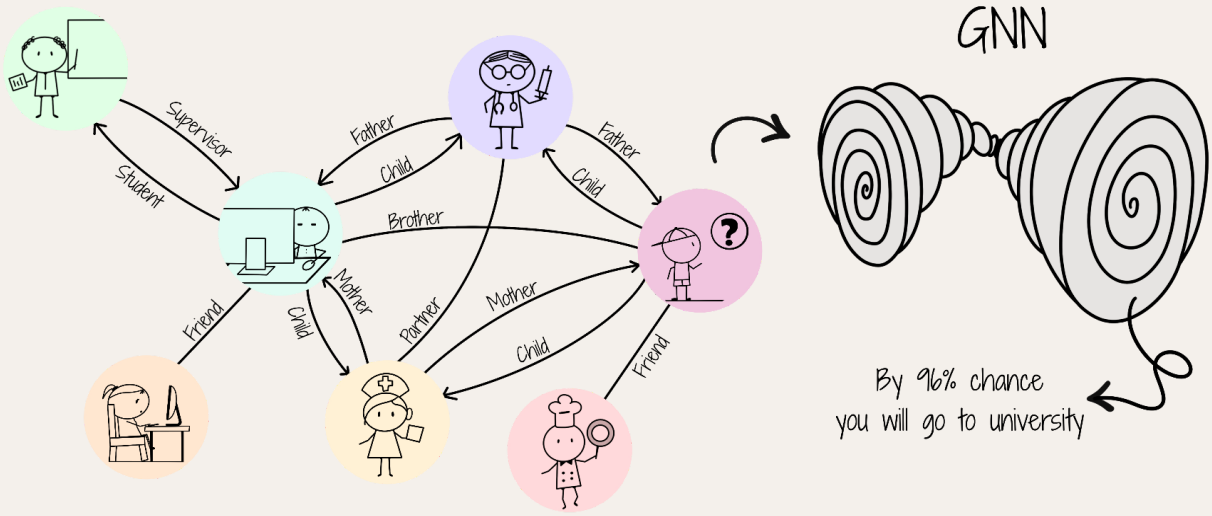
The GNN Booklet

Part I "An Introductory Overview"



Yalda Shankar

November 2022

Back to Top

Contents

List of Figures

Acknowledgements

I would like to thank Dr. Sukrit Shashi Shankar for helpful discussions and suggestions on various GNN and ML topics.

1. Introduction

The **GNN Booklet Part I** presents an **introductory overview** for the topic of **Graph Neural Networks (GNNs)**.

- ➢ We start by delineating **graphs** and **their generalizations**, and intuitively touch upon the **topological** and **geometrical** aspects of the graph data.
- ➢ We then discuss the various **graph types**, based on design, (underlying) grid and data topologies, and present some GNN-useful **graph concepts** and **operations**.
- ➢ We conclude by exhibiting the different **learning tasks** related to GNNs, while evincing the habitual algorithmic procedures for each.

The **figurative-narrative** is intended for a **general audience** (from beginners to researchers), as a collated and lucidly curated account of multiple **research papers**, **tutorials**, and **courses**, directly or indirectly related to the topic of GNNs. The pertinent works are mentioned in *Useful Readings.*

What are Graphs?

Graphs are mathematical objects, depicting pairwise relationships through nodes and edges.
Graphs generalize regular grids and sequences, such as images, audio & video.

A graph is represented as $G = (V, E)$:

- V set of nodes/vertices in the graph
- E set of edges, where each edge connects two nodes in V

(Fig. 1, pg. 7)

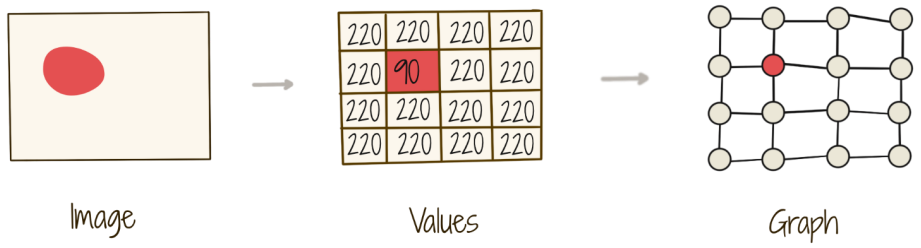


Fig. 1 - Images as Graphs

An image is a set of pixel values over a regular grid on a Euclidean plane.
A graph (with a chosen adjacent node connectivity) generalizes this, since the pixel values may then lie over an irregular grid (as shown).

Further, a graph may also represent data over a non-Euclidean structure.

Topology and Geometry in Graphs

- Topology is about the notion of **neighborhood.**
- Geometry is more about the **local shape and structure**.
- Since through shapes, geometry can also define neighborhood ideas, **topology can be seen as qualitative geometry**.

Graphs provide a natural topology to the data:

- Data over nodes and connected through edges, naturally implies neighborhood ideas.

Graphs, in their very naive form, do not model the geometry the data may be put over:

- Graphs, as such, do not consider coordinate association to nodes.
- The edges are not modeled as line segments.

Since, the data upon the graph, comes from an underlying mathematical space, the data induces its own geometry. This geometry is partially and implicitly considered during representational learning, since we want that similar pieces of data are mapped near to each other in the embedding space, although with GNNs, similarity is decided based on the topology of the data node as well.

Back to Top

Generalization of Graphs

- **Graphs** can only model dyadic (pair-wise) relations.
- **Complexes** and **Hypergraphs** generalize graphs, as they can model polyadic (involving three or more quantities) relations as well.

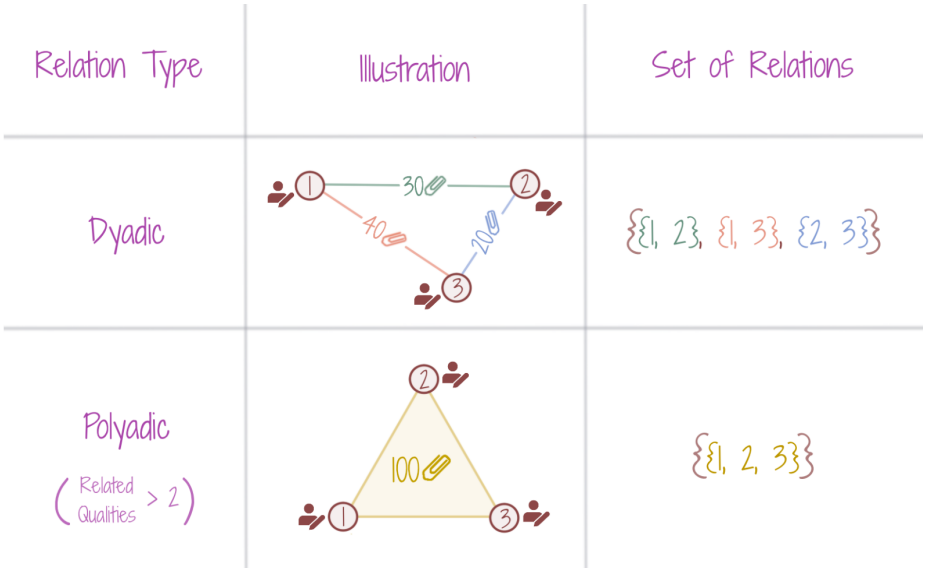


Fig. 2 - Dyadic & Polyadic Relations

A graph, when connecting 3 entities, explicitly suggests that each pair is related.
The edges are thus endowed with pairwise data (features), e.g. citation count of paper authored by 2 researchers.
What if three researchers together co-author a paper? We then require a collective feature between the three nodes. This relation is a polyadic relation, modeled by complexes or hypergraphs.

While defining a polyadic relation, both Complexes and Hypergraphs may implicitly suggest the existence of underlying dyadic relations also (through combinatorial construction properties). However, pairwise features do not mandate themselves to be incorporated.

Complexes & Hypergraphs: The Basic Notions

A Complex

can be seen as a set containing elements like points, line segments and their higher-dimensional counterparts, where these elements are **glued in a certain way**, to model nodes and their relations.

- ➢ Examples of higher-dimensional counterparts are triangles/polyhedral cells, tetrahedrons, etc.
- ➢ Complexes model polyadic relations through the higher-dimensional structures.
- ➢ Different types of complexes are studied based on:
  - The family of elements (triangular, polyhedral/cell-like)
  - The gluing constraints

A Hypergraph

can connect any number of vertices through a hyperedge.

- In a graph (V, E_G), an edge $e \in E_G$ is a **2-vertex** subset of V.
- In a hypergraph (V, E_H), a hyperedge $e \in E_H$ is a **k-vertex** subset of V, where $1 \leq k \leq |V|$.

- ➢ Hyperedges in a hypergraph enable polyadic relations.

(Fig. 3, pg. 10)

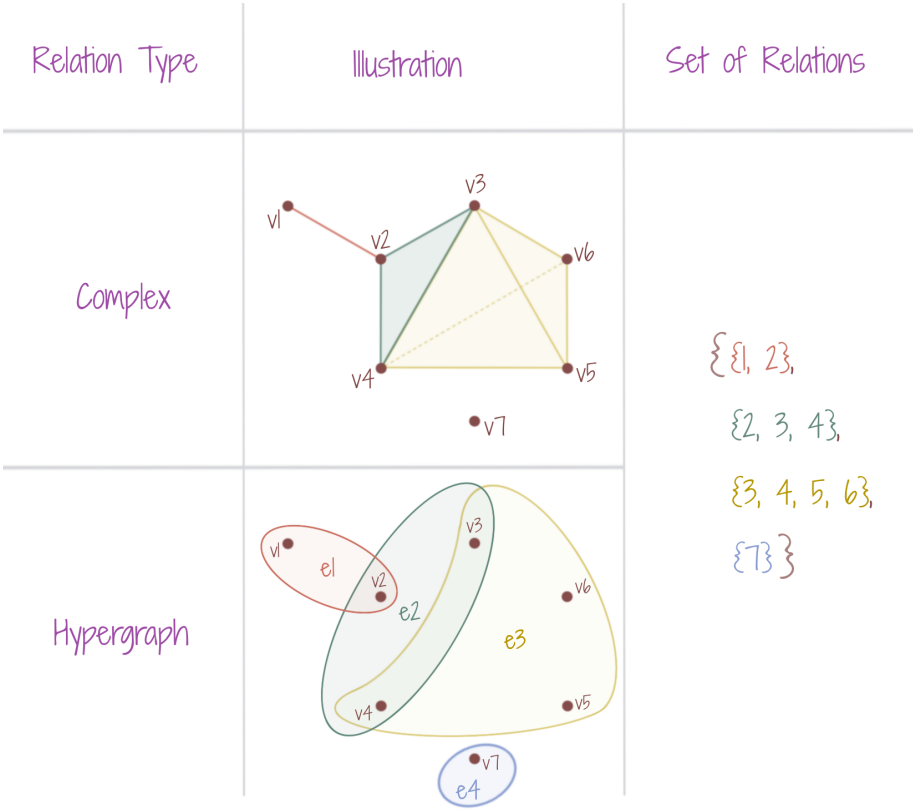


Fig. 3 - Complexes & Hypergraphs

A complex makes use of higher-dimensional structures, like triangles, tetrahedrons (instead of only line segments), while a hypergraph groups multiple vertices (instead of only two), to model polyadic relations.
Where a complex would also generally retain the notion of individual line segments as edges, a hypergraph would usually be free from such downward inclusion ideas, and only the sets of connected vertices form hyperedges (e_1, e_2, e_3, e_4).

The figure shows an undirected simplicial complex, with an isolated node, and the corresponding hypergraph.

Back to Top

(Undirected) Complexes & Hypergraphs

Simplicial Complex

Contains elements (called **simplices**) such as:

point (**0**-simplex), line segment (**1**-simplex), triangle (**2**-simplex), tetrahedra (**3**-simplex), etc.

The gluing mechanism requires that **any subset of nodes within a simplex also forms a simplex (property of downward-inclusion)**.
Downward-inclusion may seem restricting, since if three objects collaborate together, then any two amongst them should also collaborate.

Closure-Finite & Weak Topology (CW) Complex

Contains elements such as polyhedral cells (squares, hexagons, etc.) and their higher-dimensional counterparts, instead of triangles and tetrahedra, as in a simplicial complex.

- ➢ The **gluing mechanism does not require downward-inclusion, but instead requires hierarchical gluing,** i.e. gluing should be done in the order 0-cell, 1-cell, 2-cell, etc.

Cell Complex

When we **relax the gluing order in a CW complex**, we arrive at a Cell Complex.

Hypergraphs

A hypergraph **does not necessarily have downward-inclusion**, although such a condition may be enforced.

- ➢ In comparison to cell complexes, hypergraphs are helpful when a lot of **overlapping relations need to be modeled**.

Back to Top

(Directed) Complexes & Hypergraphs

The complexes and hypergraphs that generalize undirected graphs can also have a sense of direction.

Directed Complexes

In undirected complexes, putting directions on the edges of a simplex/cell, gives it an orientation.

- These orientations induce directions (of flow/travel) over the entire complex.

Directed Hypergraphs

For each directed edge in a graph, we have a **head node** (from-node) and a **tail node** (to-node). Similarly, in hypergraphs, a hyperedge is directed if the set of its connected vertices is seen as two node sets, a **head node set** and a **tail node set.**

- The directed hypergraphs can be used to model complex chemical reactions, which are non-reversible (as is often the case), thereby making the concept of direction important.

(Fig. 4, pg. 13)

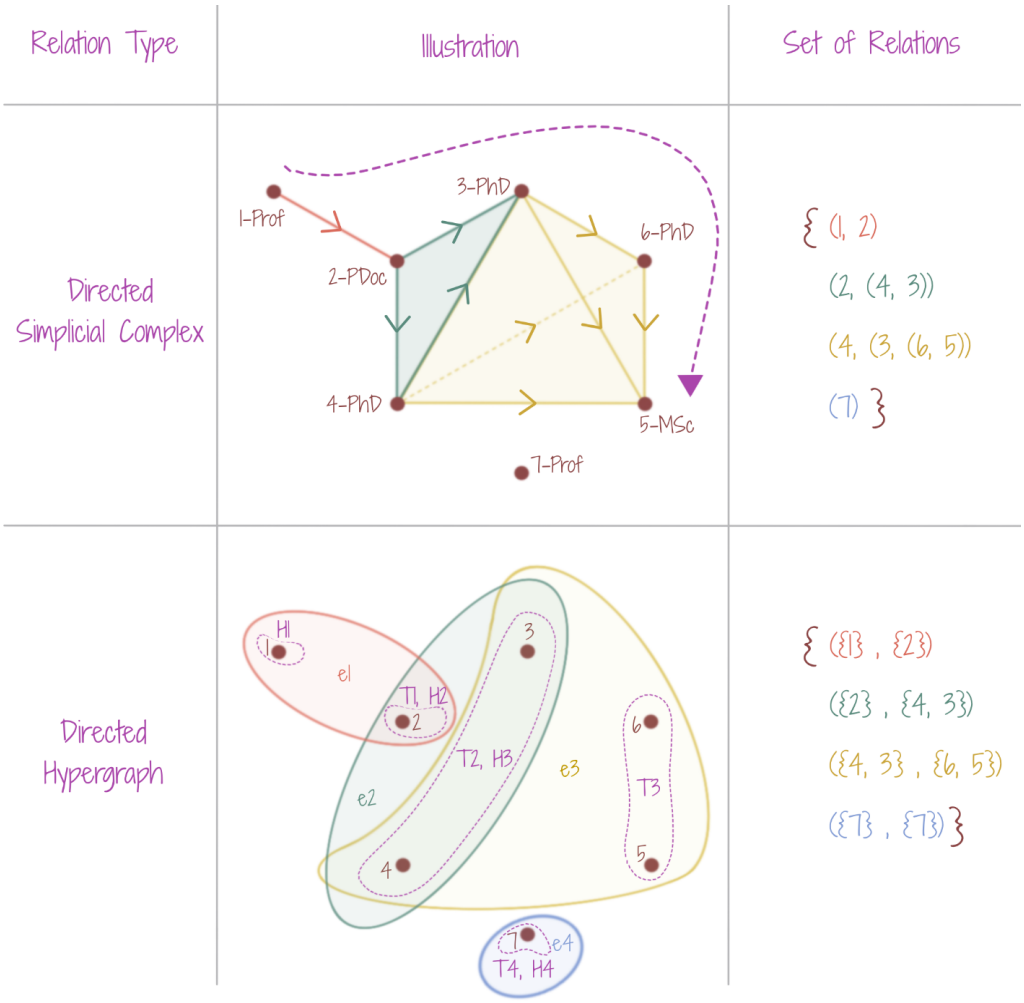


Fig. 4 - Directed Complexes & Hypergraphs

The figure shows an interesting situation, for authoring a research paper through sequential collaboration, using a directed simplicial complex, as well as, a directed hypergraph.
A professor (node 1) gives the first paper draft to his PostDoc (node 2), who then collaborates with his PhD students (nodes 3 and 4) to make further edits.
The paper then passes to another PhD student (node 6) and a Masters student (node 5), who collaborate further, to put together the final version of the paper.
Due to the directionality over the complex/hypergraph, the paper does not travel back to the main Professor (node 1), and the other Professor (node 7) never collaborates.

Note that if the direction between nodes 2 & 3 had been reversed, this would mean that the paper never passes onto the PhD student (node 6) and the Masters student (node 5) for collaboration, once it has been initiated by Professor (node 1). The PhD students (nodes 4, 5, 6) and the Masters student (node 5) although can collaborate for writing a research paper, but independent of Professor (node 1) and PostDoc (node 2).

Back to Top

Graph Neural Network (GNN)

A **GNN** is a Neural Network where the inputs are graphs.

Mostly, a **GNN learns**:

> A multidimensional (Rd) embedding/representation for each node of the graph, which is then used to perform a certain task, e.g. classification.

The representations are generally learnt with the following **objectives**:

a. Similar nodes in the graph should also be near in the embedding space.

b. The embedding space should have a lower dimension than that of the inputs.

c. The inputs ought to be linearly separable in the embedding space.

(Fig. 5, pg. 15) (Fig. 6, pg. 15)

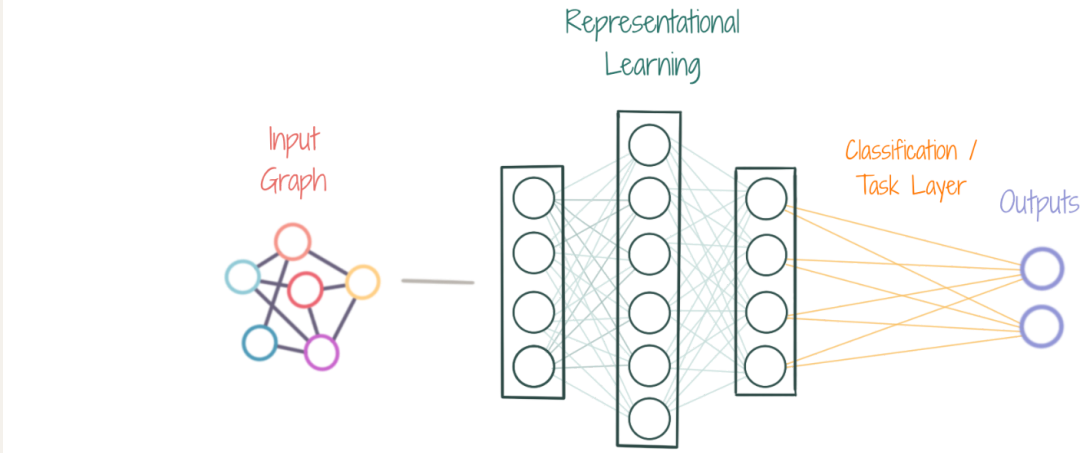


Fig. 5 - An illustrative pipeline of a GNN

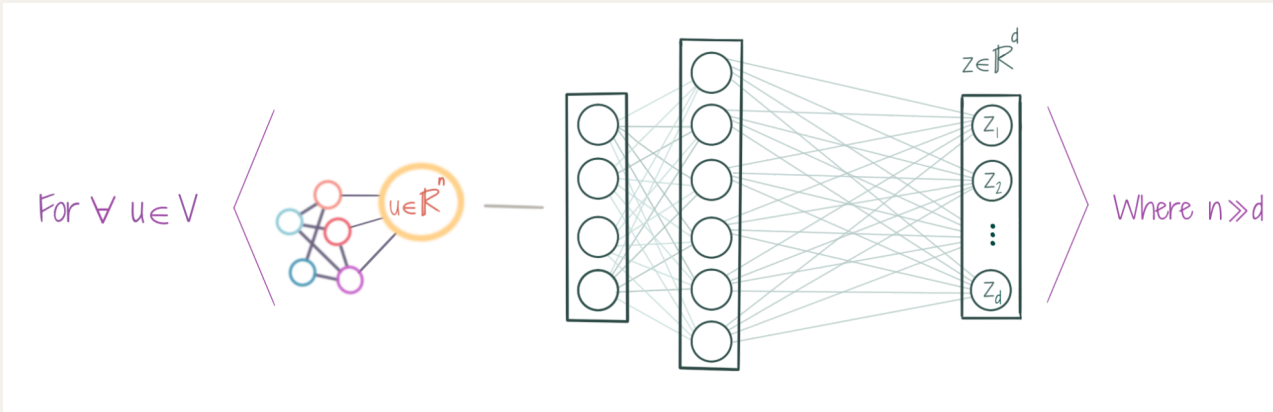


Fig. 6 - Representation Learning Illustration

Representational Learning is the automatic learning of Features/embeddings (with an NN) instead of engineering them manually. In case of a GNN, the embedding of the node u is learnt by taking into account the local graph structure around u, i.e. the information contained in the neighboring nodes and edges of u is also encoded in z.

Back to Top

GNNs vs CNNs

GNNs admit relatively **wider characteristics** of **inputs** than CNNs.

The common factors of difference are:

Size

- GNNs generally have variable sized input graphs.
- CNNs generally have fixed sized input grids (imageNet classification), although variable input size CNNs exist.

Topology

- GNNs can process a complex input topology.
- CNNs are designed for a regular input topology.

Reference Point & Ordering

- In GNNs reference points and ordering in the inputs does not exist.
- In CNNs due to a regular grid structure of inputs, an implicit reference point & ordering exists, but these are never explicitly utilized for representational learning.

Multimodal Features

- In GNNs, the inputs generally have heterogeneous data (Multimodality)
  Examples:
  Users and their movie ratings for recommender systems.
  Molecules, drug types and genetic information for biomedical applications.
- In CNNs, Multimodal input data exists, but here multimodality generally refers to different modes of communication between a human and a computer.
  Examples: Visuals, text, audio, tactile (touch) (each is one modality).
- The inputs for GNNs have more modalities than those in CNNs.

(Fig. 7, pg. 17)

	GNN	CNN
Size	Variable	Generally fixed although Variable size CNNs exist
Topology	Complex	Regular
Reference Point	Doesn't Exist	Implicitly Exists
Ordering	Doesn't Exist	Implicitly Exists
Multimodal Features	Exist	Exist More

Fig. 7 - Inputs in CNNs and GNNs

The table shows key differences in the characteristics of the inputs, admitted by GNNs and CNNs. It is clear that the GNNs allow a wider variety in their input data.

The **multimodality**, should, roughly be understood as, **being of different types**. However, the way the term has been used with CNNs versus that with GNNs, invites some mention.
Usually, with **CNNs and in the field of CV/ML**, multimodality has referred to the types that arise from **distinct modes of human-computer interaction**, e.g. textual, visual, auditory.
However, in **GNNs**, **multimodality may simply mean different types of data**, where these types may even come from the same mode of human-computer interaction, e.g. in a knowledge graph. This essence of multimodality can be seen implicitly within the CNNs, e.g. different parts of a face (nose, eyes, mouth, head, cheeks) can be seen as multi-modal features, as they produce modes in the joint probability distribution of the facial images.
In principle, this is very justifiable; however, in practice, the term has precluded CNNs for such purposes.

Back to Top

Can GNNs be applied for CNN-style Inputs?

Graphs generalize regular grids, therefore GNNs can definitely be applied for CNN-style inputs. However, the output from the GNNs may only be more useful, if the application benefits from explicit encoding of geometrical structure (not just topological) of the underlying grid of the input data.

For example given some part of the grid:

- which type of neighbors are associated with it (topological)
- and in what directions (geometrical).

Note, CNNs encode such geometry only implicitly. This is one of the reasons why transformers (a type of GNNs, but not conventional GNNs) have recently started exceeding CNN performance for vision applications, as they explicitly encode the relative positioning of the input patches, for learning embeddings.

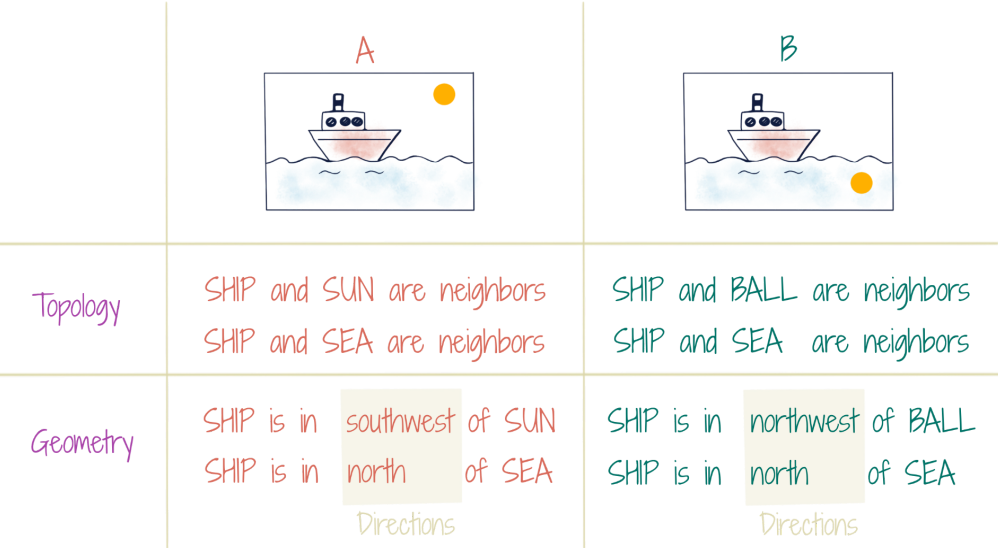


Fig. 8 - Topology & Geometry Comparison

Using only topological information (conventional GNNs), the ball in image-B can be predicted as the sun. Adding explicit geometry (like in transformers) resolves this, while implicit geometry (like in CNNs) may/may not.

Back to Top

2. Graph Types and Operations

Node & Edge Features

Nodes and edges can have some features, which come from an underlying mathematical space. Features can also be seen as attributes or weights across applications.

Graph Types based on Node & Edge Variations

Undirected Graph

A graph is undirected/both-ways relationship If:

elements of E are unordered pairs of the form $\{u, v\};\ u, v \in V$,

there is no direction on the edges.

Directed Graph

A graph is directed If:

elements of E are ordered pairs of the form $(u, v);\ u, v \in V$,

there is an implication of direction from node u to node v.

Mixed Graph

A graph is mixed If:

Contains both directed and undirected edges.

(Fig. 9, pg. 20)

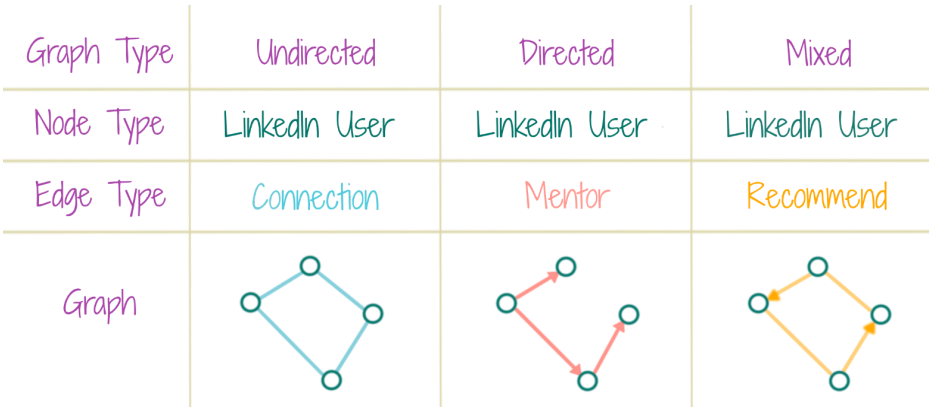


Fig. 9 - Undirected, Directed & Mixed Graphs (Graphs based on Edge Variations)

The edges in a graph can be directed, undirected, or both (mixed). An undirected edge conveys a two-way relationship, while a directed edge indicates only a one-way connection.

Heterogeneous Graph

In a graph, the nodes and edges can be assigned types.

A graph $G = (V, E, R, T)$ where:

- R set of all relation/edge types
- T set of all node types

Heterogeneity is synonymous to multimodality in GNNs, e.g. a Knowledge Graph.

Multi-Graph

A graph, where multiple connections (directed/undirected) are allowed between the nodes.
A multi-graph is more meaningful for Heterogeneous graphs, where multiple connections can be of different types.

(Fig. 10, pg. 21)

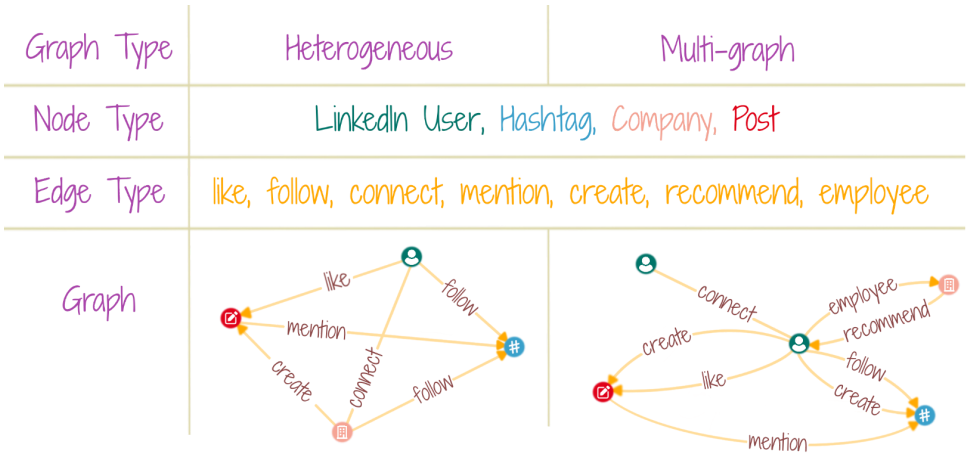


Fig. 10 - Heterogenous & Multi-graphs

By default, all nodes (and edges) in a graph are of the same type(s), and only one edge is allowed between any two nodes of a graph, be it directed or undirected.
A heterogenous graph allows nodes to be of different types, and so, for the edges.
A multi-graph allows for multiple edges between any two nodes in a graph.

The figure shows examples from a conceptual LinkedIn network. Each example has multiple node types (individual user, company, post, hashtag) and multiple edge types (like, follow, connect, mention, create, recommend, employee).
Some edge types like connect are undirected, since they indicate a two-way relationship, while those like follow are directed, since it's not always necessary that two connected entities follow each other.
A user can both create and like a post; thus, two edge types are possible between users and posts. However, these edges are of a directed variation, since only a user may create/like a post, and not vice-versa.

Back to Top

Special Topologies

Bipartite Graphs

A graph where, the vertices can be divided into two disjoint sets V_1 and V_2, such that:

- The edges only connect vertices between V_1 and V_2.
- No vertices within V_1 or V_2 are connected.

➢ Bipartite Graphs are mostly used in the Recommender Systems.

Multipartite Graphs

An extension of bipartite graphs to more disjoint sets $\{V_1, V_2, \dots, V_k\}$, where the edges never connect vertices within the same disjoint set.

Multipartite graphs have been recently used to represent genomic data.

Folded (Bipartite/Multipartite) Graphs

Folding is a general concept in graphs, where:

- For non-adjacent nodes u & v with a common neighbor w, folding puts an edge between u & v.

A folded bipartite graph forms two graphs F_1 and F_2, corresponding to V_1 and V_2 respectively, such that vertices in F_1 or F_2 may be connected.

(Fig. 11, pg. 23)

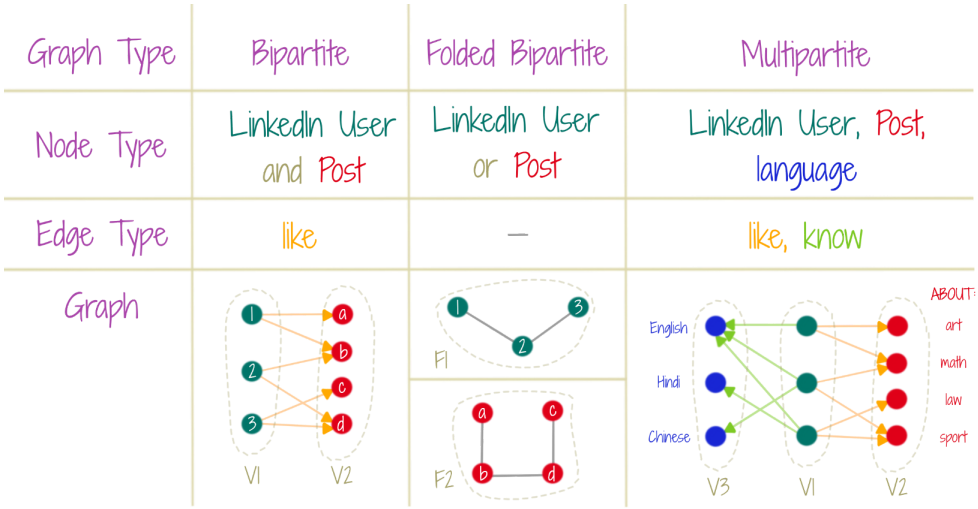


Fig. 11 - Bipartite, Multi-partite & Folded Bipartite Graphs

A bipartite graph has two disjoint sets of nodes, such that no nodes within each set are connected.
A multi-partite graph extends this idea to more than two disjoint sets of nodes.
A folded bipartite graph establishes a sense of (indirect) connections between the nodes of each disjoint set of a bipartite graph.

Thus, in the figure, the folded bipartite graph for the red nodes, shows that red nodes (a,c) are not (indirectly) connected through green nodes, while (a,b), (b,d), (c,d) are (indirectly) connected.

A similar folding operation may also be applied to the disjoint sets of a multi-partite graph.

Dense/Sparse Graph

Density/Sparsity of a graph can be measured by a simple and standard graph metric:

Graph Density = Number of edges in the graph/Maximum number of possible edges

Dense Graph

A graph in which the number of edges are close to the maximal number of edges, i.e. majority of the nodes of the graph are connected**.**
A dense graph is therefore a graph in which it's not easy to take out some vertices as a lot of edge connections need to be broken. This idea is synonymous with the idea of dense sets in Mathematics.

- Dense Graphs are used in GNNs to handle noisy data on graph nodes and edges.

Sparse Graph

It's opposite of a dense graph, where the number of edges are very less in comparison to the maximal possible**.**

- Most real-world graphs are sparse.
- Learning GNNs on sparse graphs is a challenge.

(Fig. 12, pg. 25)

Back to Top

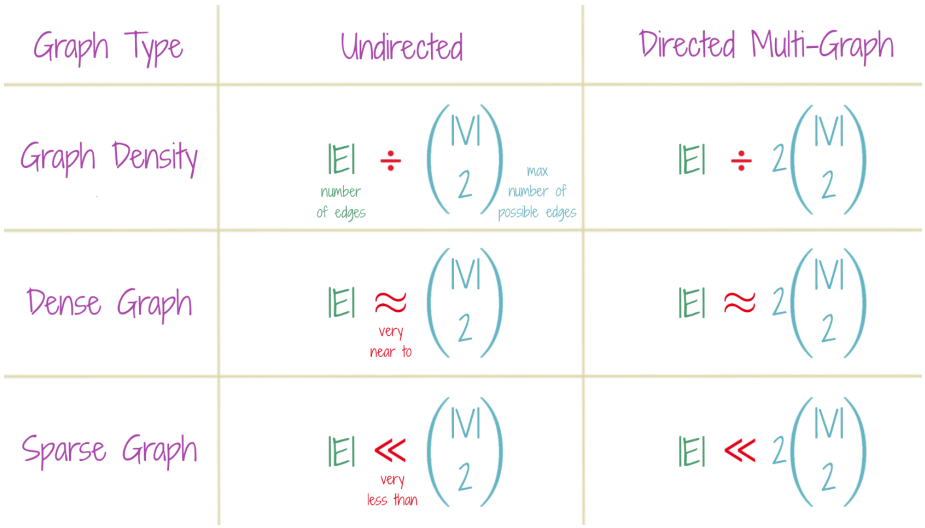


Fig. 12 - Density and Dense/Sparse Graphs

Density of a graph is a measure of how connected a given graph is, in comparison to what it possibly be.
A sparse graph is very less connected, while a dense graph is almost maximally connected.
Note that in the case of directed graphs, the maximal number of edges are twice than those in undirected graphs, in order to account for directedness.

All given expressions are for simple graphs, i.e. multiple edges are not allowed between any two nodes.

Regular, Complete and Random Graphs

These types of graphs are poor for real-world applications, but useful for establishing theoretical bounds on real-world graphs.

Regular Graphs

Each vertex has the same number of neighbors.

Complete Graphs

Each pair of nodes is connected.

Random Graphs

In random graphs, edges and nodes may be seen:

- to be generated by a random process,
- or coming from a probability distribution.

➢ Properties of random graphs may change or remain the same under certain graph functions; these properties are used to study real-world graphs through approximations to random graphs.

➢ Random graphs are also used in the watermarking of GNNs.

(Fig. 13, pg. 27)

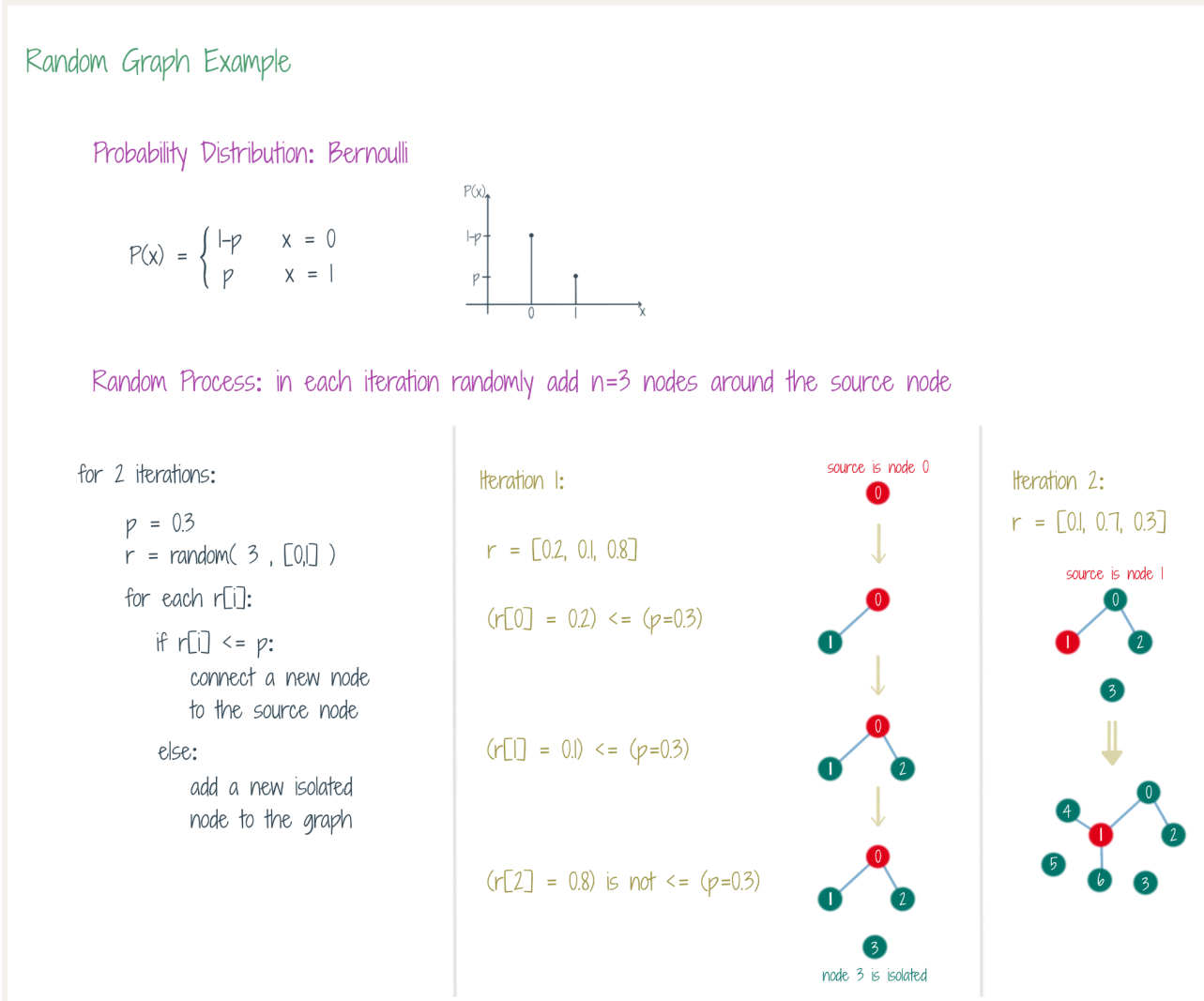


Fig. 13 - Random Graphs

The figure shows how the edges of a *simple* graph may be generated (independently of other edges) through a random process, where at each processing time-step (iteration), the random variable follows a Bernoulli Distribution; Here 2 iterations are illustrated. Criteria may also be defined to add/delete nodes of the graph, through another random process.

The Erdos-Renyi Model is a famous random graph generation model, which is used to provide guarantees on the existence of properties of interest, for a general family of graphs.

Back to Top

Special Feature Topologies

Homophily, Heterophily and Disassortativity in Graphs are special topologies, between the features of the neighboring nodes. These topologies may be seen over various graph topologies (of the previous section).

Homophily

A node is likely to be connected to nodes of the same type.

Heterophily

A node is likely to be connected to nodes of different types.

Disassortativity

A node is likely to receive unimportant/harmful information from the neighboring nodes.

- Homophilic and Heterophilic Graphs are Assortative Graphs.
- GNNs assume that the graph geometry is homophilic, but many real world graphs are heterophilic and disassortative.
- Learning GNNs for heterophilic and disassortative graphs can be a challenge.

(Fig. 14, pg. 29)

Noisy Graph

A graph where the data on nodes and edges may be noisy, i.e. incorrect.
Examples: A knowledge graph auto-mined from the web, or, a social network with fraudulent connections.

- Training GNNs with noisy graphs is a challenge, as the representations (which are usually obtained through recursive aggregation in GNNs) get corrupted.

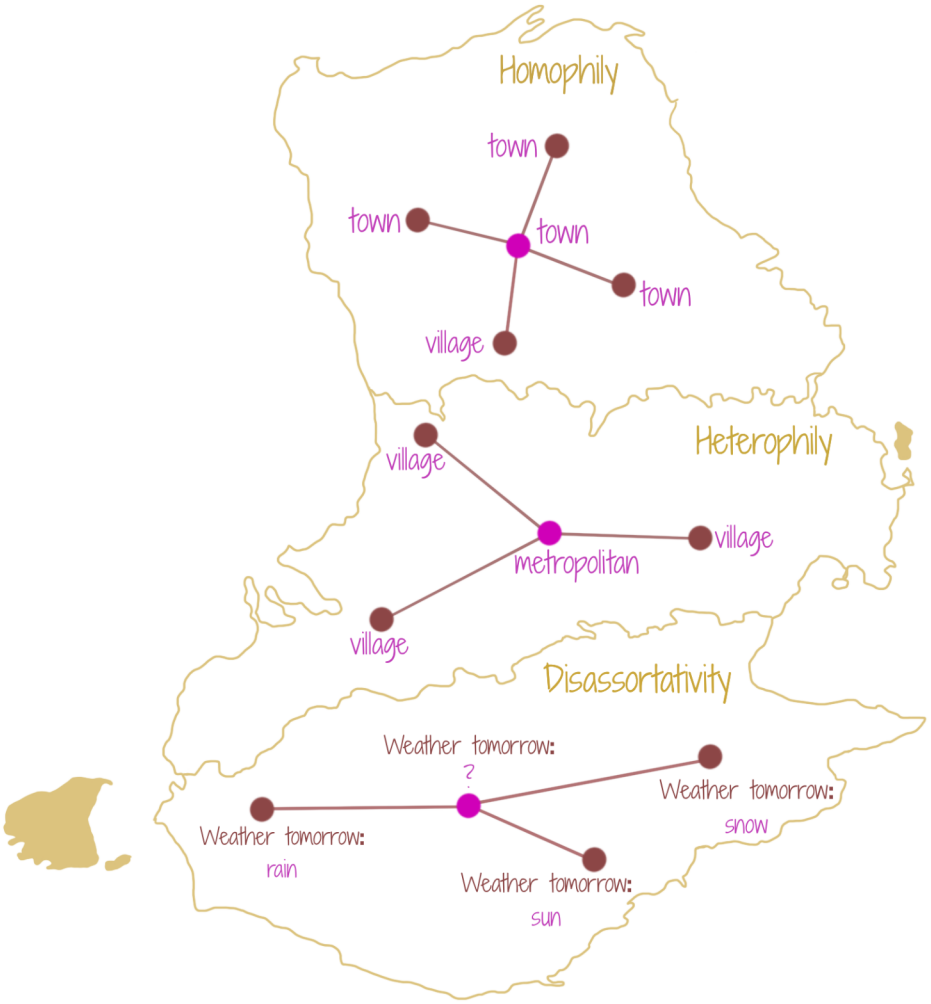


Fig. 14 - Homophilic, Heterophilic & Disassortative Graphs

A homophilic graph contains similar neighbors (nodes) to a given node, while a heterophilic graph is likely to contain dissimilar neighbors to a node. In both cases, however, a node receives useful/important information from its neighbors.
When the neighbors of a node do not provide any useful information, the graph is disassortative. Disassortative graphs, therefore, often require access to long-range relations for gaining useful information.

The figure shows the homophilic & heterophilic concepts through nodes that may depict *towns, villages, metropolitan cities* in a country. The disassortativity is shown through a hypothetical construction, where saying everything about the weather forecast, does not give any clear idea for weather prediction, and therefore, is not useful.

Note that homophilic graphs are also called associative **graphs**, since a node of a given type is likely to be **associated** with the nodes of the same type. Under similar parlance, heterophilic graphs are also called **dissociative graphs,** since a node of a given type is likely to be **not-associated** to nodes of the same type.

Back to Top

Graph Operations

Degree Matrix (D)

a $V \times V$ diagonal matrix, which specifies the number of edges attached to each vertex $v \in V$ in the graph.

- ➢ In a directed graph, the edges can attach as incoming or outgoing, so we have an incoming and an outgoing degree matrix.
- ➢ In case of undirected graphs, a self-loop counts as two edges to the vertex (by convention), since there are two points of edge attachment to the node.

Adjacency Matrix (A)

A $V \times V$ square matrix, that tells which vertices are adjacent to a given vertex.

- ➢ In undirected graphs, the matrix is symmetric.
- ➢ In simple directed graphs, the matrix is unsymmetric. (In directed multigraphs, the matrix may or may not be symmetric).

Laplacian Matrix (L)

Laplacian Matrix (L) = Degree Matrix (D) − Adjacency Matrix (A)

- ➢ This can be seen as a slightly modified way of specifying the graph topology.
- ➢ Intuitively, it can be thought of as a discretized version of the Laplacian operator ∇^2.

- Imagine a function/surface being discretized and approximated by a graph, such that the nodes are dense where the function's second derivative is greater.

(Fig. 15, pg. 31)

Back to Top

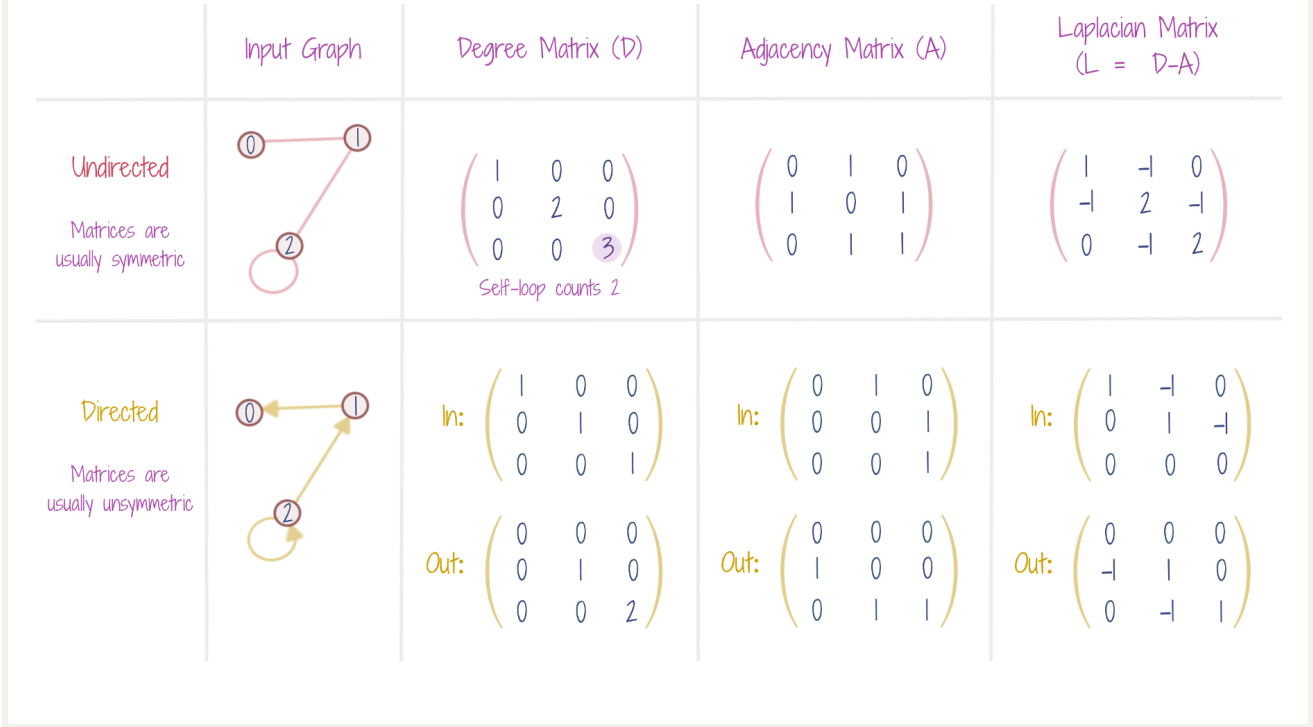


Fig. 15 - Degree, Adjacency and Laplacian Matrices of a Graph

A degree matrix tells the number of neighbors for each node, while an adjacency matrix tells which nodes are the neighbors. The Laplacian matrix, say, being applied to a function f, tells how much *on an average* the value of f at a given node i is greater than the value of f at i's neighbors.

The Laplacian Operator can also be given an **interpretation** as being the **divergence of gradient of the function f** (scalar field). This is by the definition of the Laplacian in terms of the gradient operator **∇**. If one imagines f as a flow function, a near-zero Laplacian (on a region) would then mean that the flow does not form a sink or source (in that region), or, in other words, the flowing fluid does not contract or expand (in that region).

Spectrum of a Graph

It is the set of eigenvalues of A or L matrices.

Importance of Graph Spectrum:

- The eigenvectors carry notions of smoothness of the function being approximated by the graph.
- Thus, studying the spectrum can tell us about some aspects of GNN performance.

Back to Top

Connected Graph/Components

A graph is **connected** if every pair of vertices are connected by a path.

Connected Components (Undirected)

Components of a graph (subgraphs) where each of them is a connected graph.
Across (connected) components no node or edge is shared.

- ➢ A graph that is itself connected has only one connected component.

Strongly Connected Components (directed)

It is a subgraph where there is a path between every pair of nodes, taking into account the directionality of edges. A strongly connected component is defined for a *directed graph.*

Weakly Connected Components (directed)

It is a subgraph where there is a path between every pair of nodes, by not taking into account the edge directions. A weakly connected component is also defined for *directed graphs.*

(Fig. 16, pg. 33)

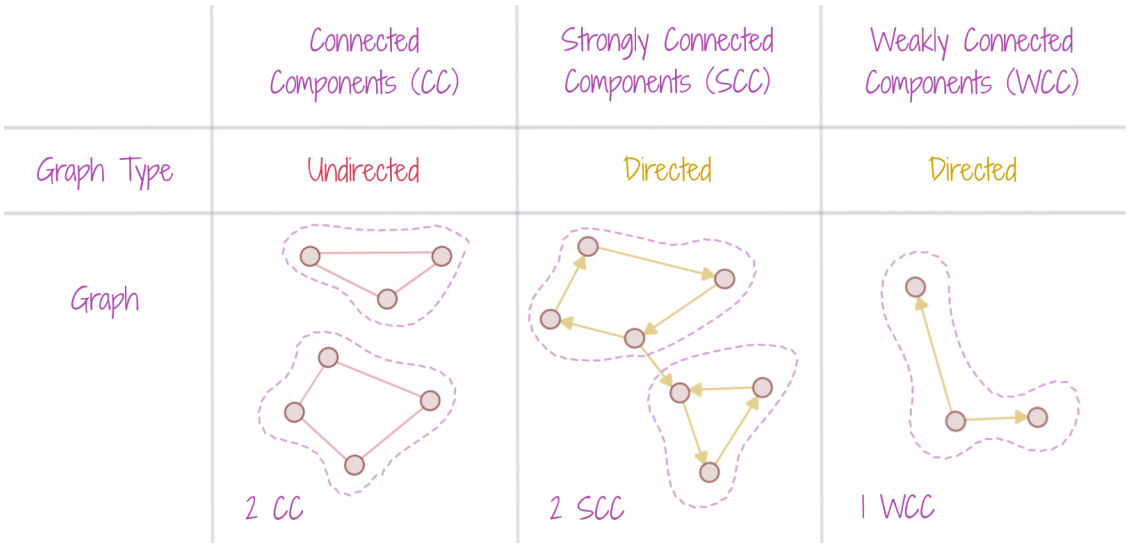


Fig. 16 - Strongly and Weakly Connected Components

A connected component in an *undirected graph* is a subgraph in which there is a path between every pair of nodes.
To extend the notion to directed graphs, we have strongly and weakly connected components.

Note that the basic idea of a connected component is that we can traverse from any node to any other node in the component. It's just that we have more specific terminology for directed graphs.

Fig. 17 - Edge Contraction in a Graph

An edge in a graph can be contracted by merging the two connecting nodes to a single node.
The figure shows that the edge connecting the nodes 0 and a, is contracted, and the new merged node is x.
The neighbors of new node x, are a union of the neighbors of nodes 0 and a.

Edge Contraction

Remove an edge from a graph, and merge the two connecting nodes of the edge into a new node.
Edge Contraction is used in GNN pooling methods.

(Fig. 17, pg. 33)

Minimum k-Cut

A partition of a graph into k connected components, such that the cut is minimal in some metric.

- A simple min-cut is a min 2-cut.
- It is used in GNN pooling and also in subgraph mining.

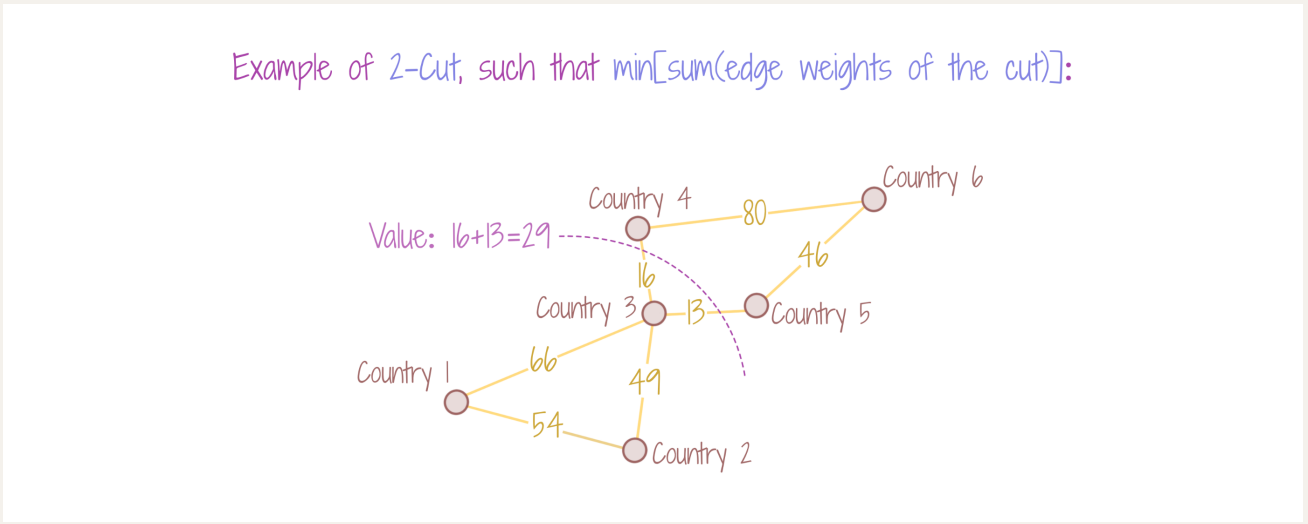


Fig. 18 - Minimum k-Cut

The figure shows an example of min 2-cut, the graph is thus partitioned into 2 connected components.
The graph nodes represent countries and edges represent the value of trade occurring between the countries.
We wish to form two country coalitions, such that discontinuation in trade partnerships after the coalition, has minimal effect on the trade, i.e. the partitioning of the graph is minimal in edge values.

Graph Coloring

Assigns node-types to a graph, such that no adjacent nodes are of the same type (color).

- An NP-complete problem.
- The problem poses both an empirical/theoretical challenge to be approximated by GNNs.

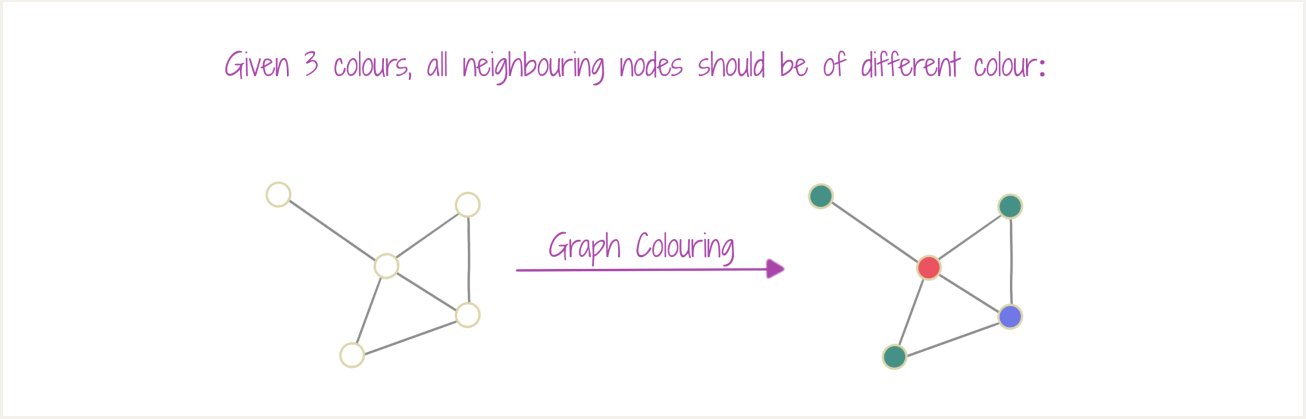


Fig. 19 - Graph Coloring

Graph coloring is the process of labeling nodes of the graph into node types (called *colors*), according to the constraint that no two neighboring nodes are of similar type (color). It is one of Richard Karp's 21 NP-complete problems.
In a more general scenario, one can also label edges with a given set of types (colors), according to a new constraint.

Graph Coloring algorithms are used in Computer Science to solve problems, where we have a limited amount of resources, with some restrictions on how they can be used, e.g. network scheduling, register allocation.
Sudoku puzzles can also be seen as a graph coloring problem, since the resources (numbers) are limited, and they have to be used as per the constraints of the sum along rows, columns, and 3x3 squares.
Training a GNN which can provide good approximation guarantees to **graph coloring problems,** can be challenging.

Back to Top

Graph Isomorphism

Isomorphism

- A structure-preserving one-to-one (bijective) function.
- Meant to exist between mathematical objects of the same type.

Graph Isomorphism

- A function from vertices of graph G_1 to the vertices of graph G_2,
  - such that the topological structure is preserved.
- G_1 and G_2 are the mathematical objects of the same type.
- Graph Isomorphism is an important concept for understanding the expressive power of GNNs.

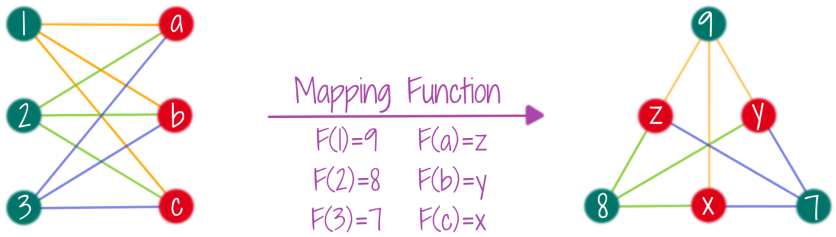


Fig. 20 - Graph Isomorphism

Two graphs are isomorphic, if they are essentially the same graphs (topologically), but with different node names.
The figure shows two isomorphic graphs and a mapping function that matches the nodes across these graphs.
Note that the topology (connection pattern) in the two graphs is the same.

Graph Isomorphism has emerged as a very useful concept to **study the expressive power of GNNs,** i.e. which properties of graphs GNNs can't recognize.

3. Learning Tasks in GNNs

Graph neural networks (GNNs) can be used to learn **multiple types of tasks** for graph-input data. These **learning tasks** can mostly be of **three types**:

a. Classification
b. Structure Mining
c. Generation

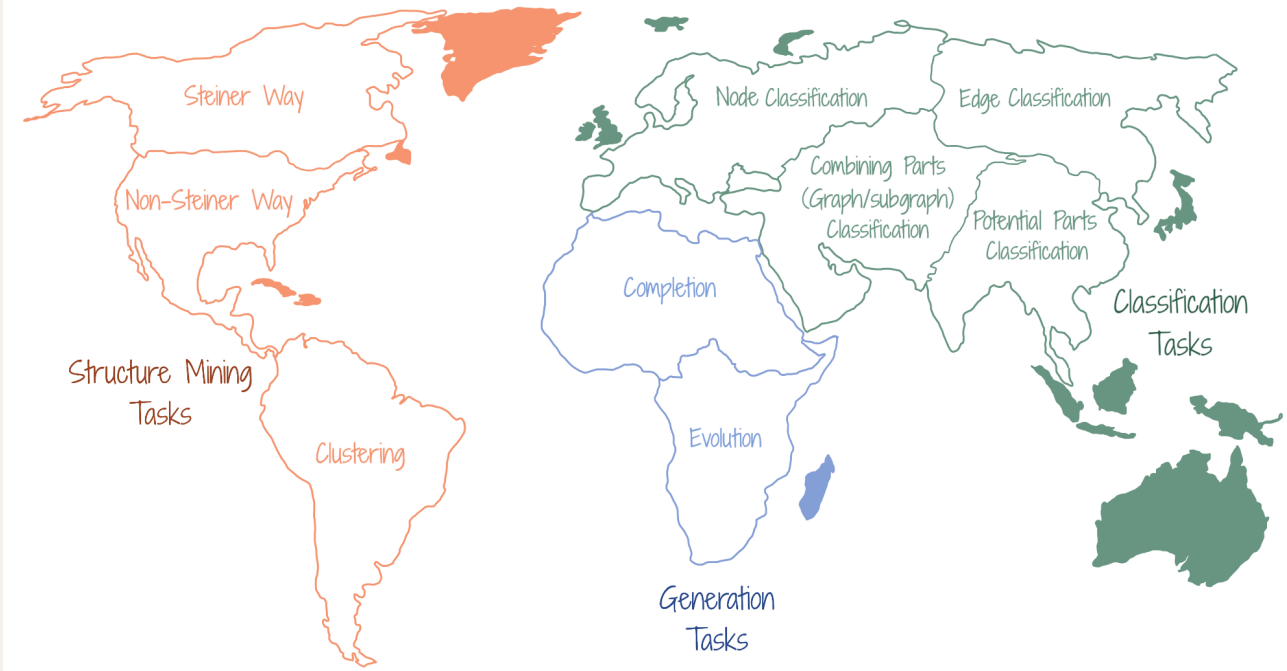


Fig. 21 - Learning Tasks in GNNs

Classification Tasks

The aim is to assign some part (or whole) of the graph to known classes.

Classification Tasks can be categorized into the following:

a. Node Classification
b. Edge classification
c. Combining Parts (Graph/Subgraph) Classification
d. Potential / Missing Parts Classification

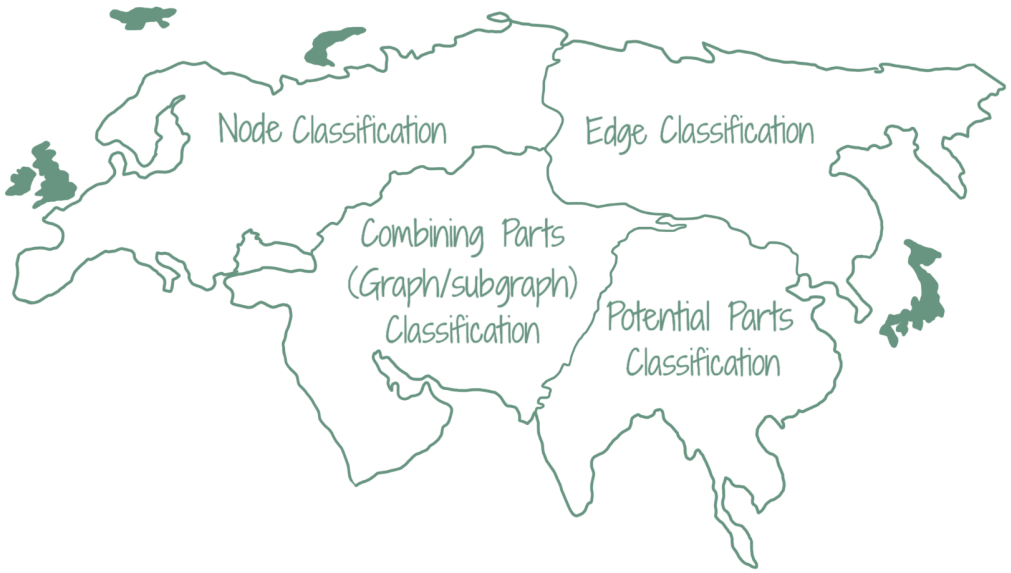


Fig. 22 - Classification Tasks

Node Classification

For an input graph G, a GNN is learnt to assign a class (from a pre-chosen set of node classes) to each given node n.

The assignment is made considering the topology of the node n, i.e. its neighboring nodes.

The neighboring nodes, in turn, come with their own topology.

Thus, a node n, indirectly encompasses the **entire graph topology**.

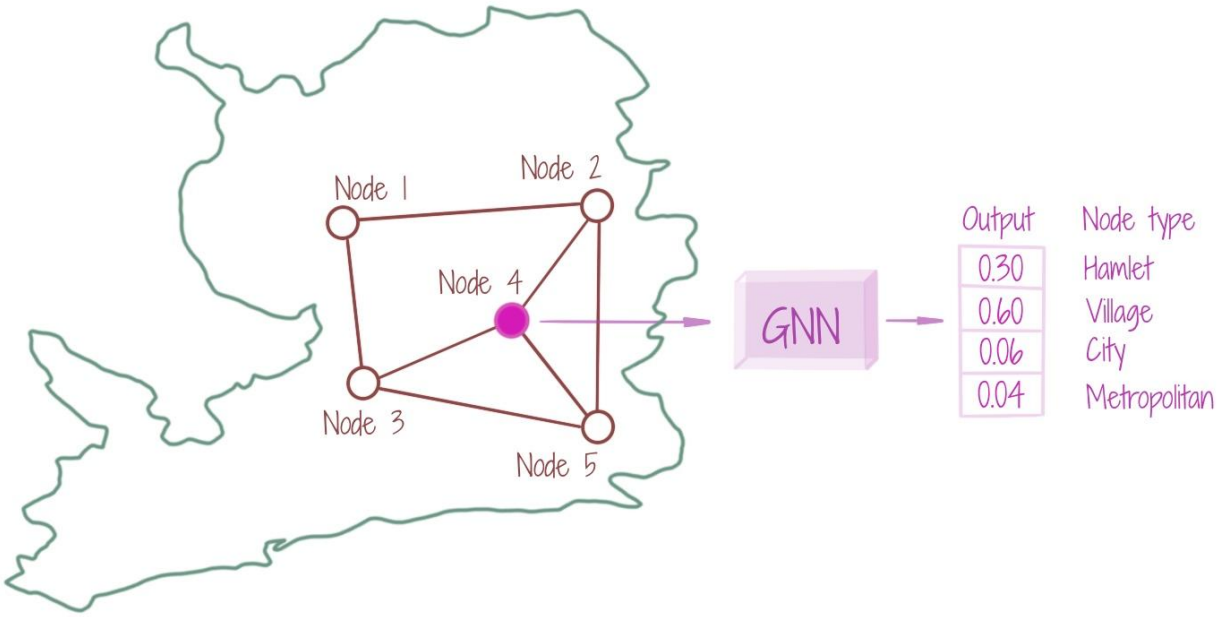


Fig. 23 - Classification of a Node

The figure shows that the pre-chosen set of node classes is {hamlet, village, city, metropolitan}, and the node is assigned the class village based on the highest probability output from the GNN.

Node feature learning (used in node classification) **is widely used in most other GNN learning tasks as well.**

Edge Classification

For an input graph G, a GNN can be learnt to assign a class (from a pre-chosen set of edge classes) to each given edge e.

The assignment is made considering the topologies of the connecting nodes,
which indirectly encompass the **entire graph topology**.
The edge classification is thus dependent on the **node-level representations**.

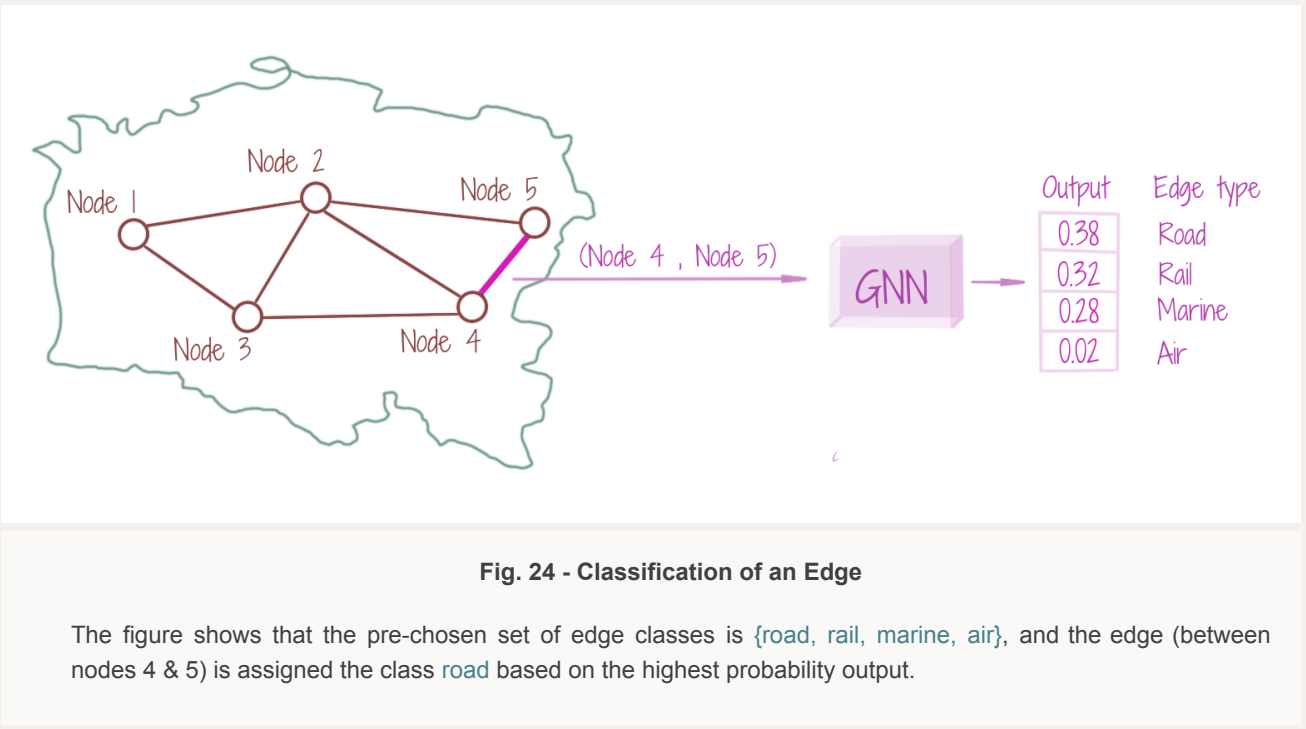


Fig. 24 - Classification of an Edge

The figure shows that the pre-chosen set of edge classes is {road, rail, marine, air}, and the edge (between nodes 4 & 5) is assigned the class road based on the highest probability output.

Combining Parts (Graph/Subgraph) Classification

We may want to assign a class to an entire graph, or a subgraph (which is not a single node/edge).

The procedure:

typically starts with the learning of node-level embeddings, and then combining them with a function of choice, to build the representation for the graph (subgraph).

These combined embeddings can be converted to class-specific probabilities by a simple neural network.

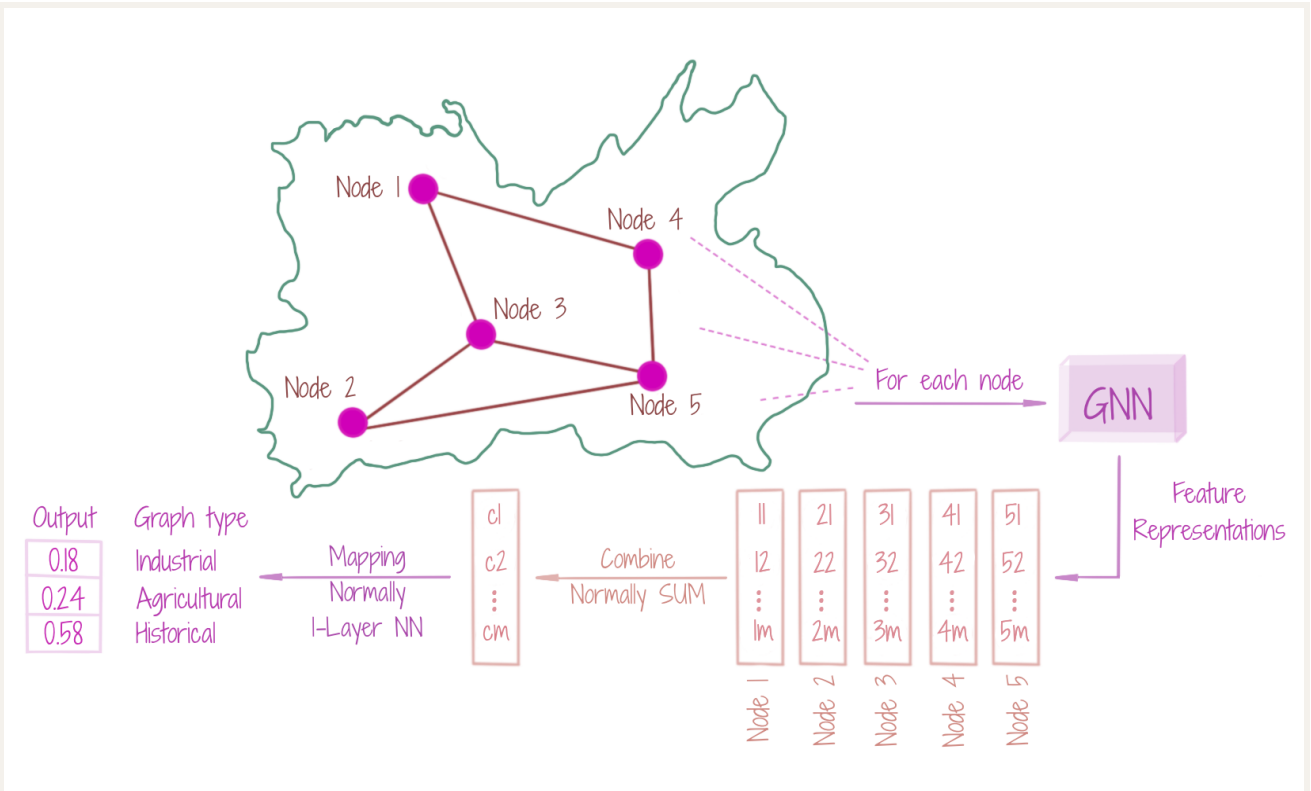


Fig. 25 - Classification of a Graph/Subgraph

In the figure, the nodes come from {village, metropolitan, hamlet, city}, and based on the corresponding feature representations, a selected area may belong to {industrial, agricultural, historical}.

The node representations take into account the connecting edge types that come from **{road, rail, marine, air}.**

Potential Parts Classification

For an input graph G, we may want to predict:

if a missing edge should exist or not, and **if a given node may have a type T**.

The **former** can be seen as a **(binary) edge classification problem**, while the **latter is essentially a (binary) node classification problem**.

Such binary classification problems have an essence of **finding the existence** or **the nature of potential parts** of the graph, when the parts are topologically known.

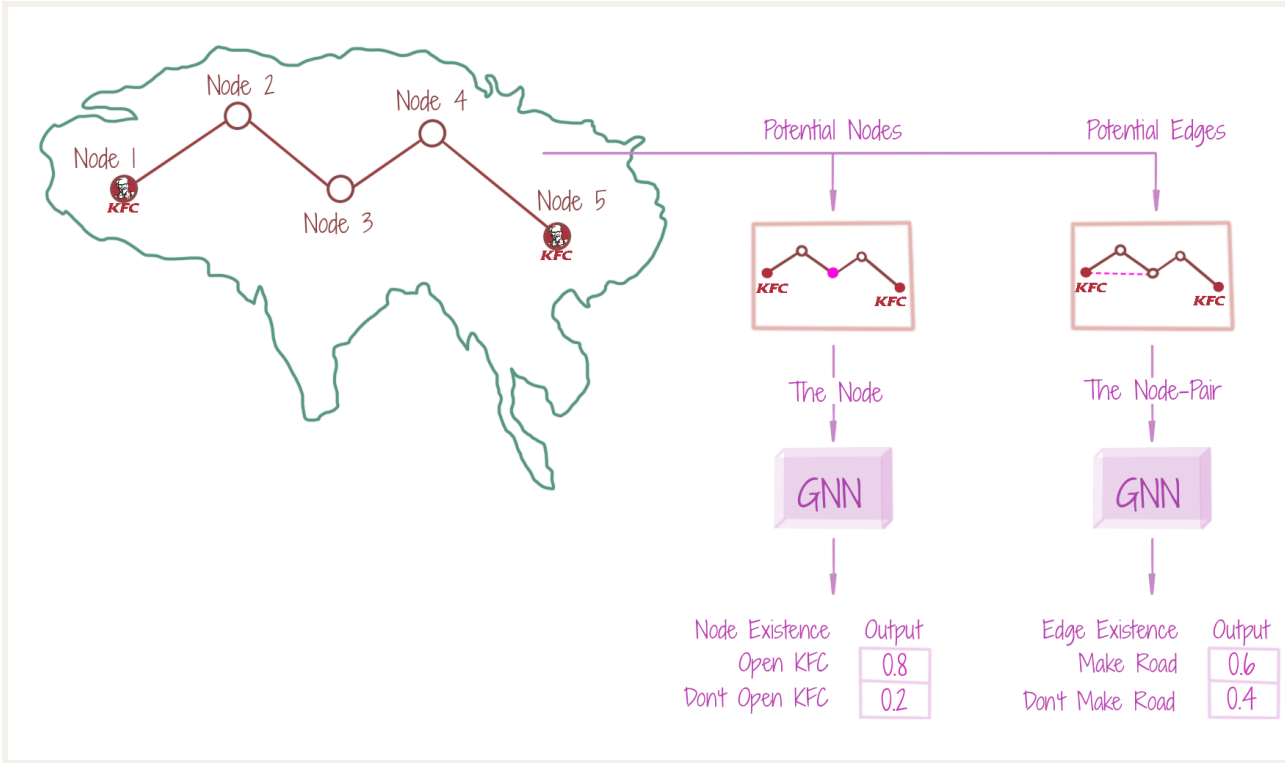


Fig. 26 - Classification of Potential Parts

The figure attempts to depict the scenario, where we wish to decide if a given property (node) should be designated for KFC (node type), and whether a road (edge) may be established between two given eating outlets.

The (binary) edge classification is often referred to as the **link prediction problem**, useful in recommender systems, e.g., an edge may be established between a user and a movie, if the user likes the movie, else not.

Structure Mining Tasks

The aim is to find the part (or mild augmentation) of the graph that may most appropriately represent a target concept.

Structure Mining Tasks can be categorized into the following:

a. Clustering
b. Non-Steiner Way
c. Steiner Way

Fig. 27 - Structural Mining Tasks

Back to Top

Clustering

Given a graph G, we may want to find subgraphs reflecting **similar concepts**.

We can achieve this by:

- **clustering over the node-level feature representations** (learnt through GNN),
- using unsupervised algorithms like K-means (or spectral clustering if the features seem to be in a non-compact geometry).

➢ The node representations take into account the connecting edge types as well.
➢ Here the GNN learning objective has **no notion** of similarity of subgraph-level representations. However, such a criterion may be incorporated using metric learning over the clusters.

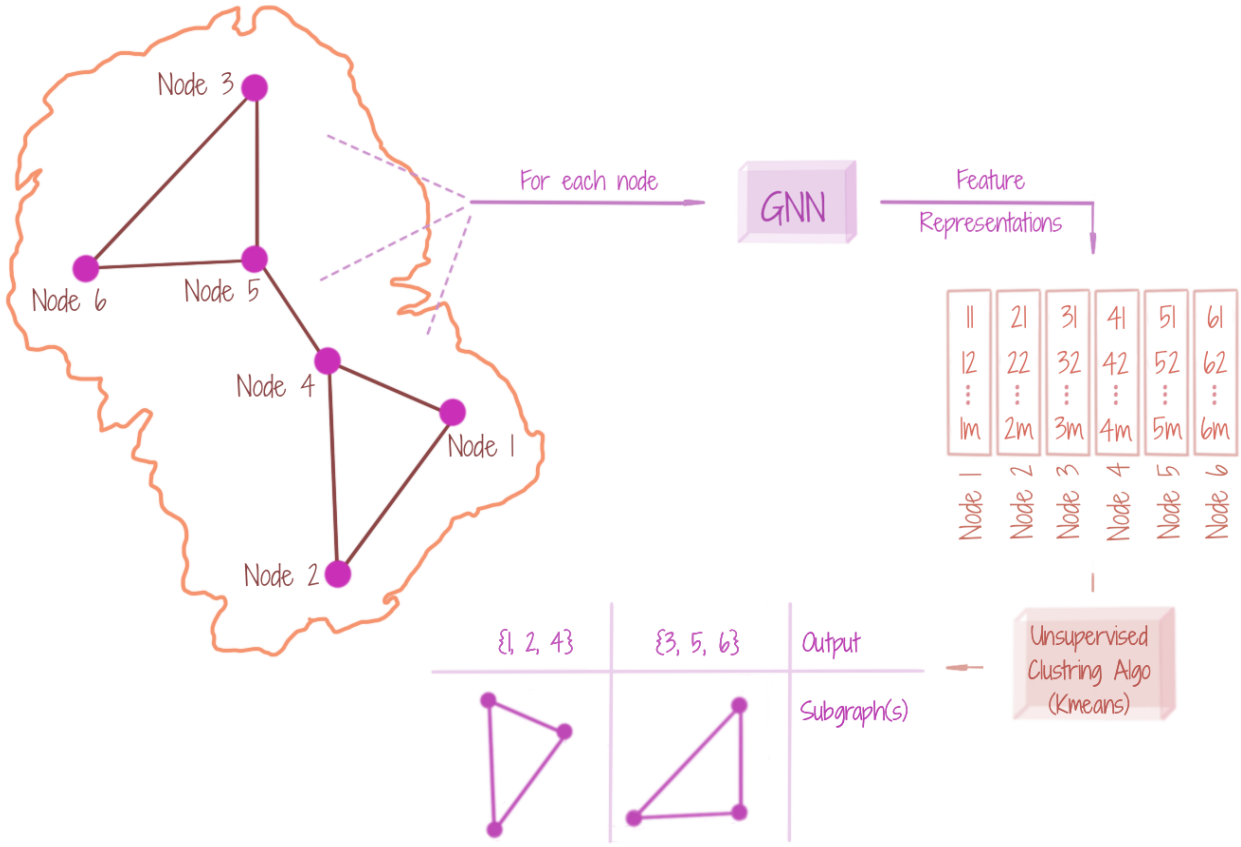


Fig. 28 - Structure Mining by Clustering

Back to Top

Non-Steiner Way

For an input graph G, we may want to **find a subgraph matching a specific objective**, which may not be directly derived from the node and edge attributes.

We can achieve this by:

- collecting the node-level representations (which indirectly consider the network topology with connection types), and
- designing an appropriate subgraph selection algorithm that would translate any selected subgraph to the constraint estimate.

➢ There might be **multiple outputs**, each satisfying the given criteria.
➢ **Here we do not consider any graph expansion for finding the optimal subgraph,** and hence, subgraph mining is non-Steiner.

(Fig. 29, pg. 46)

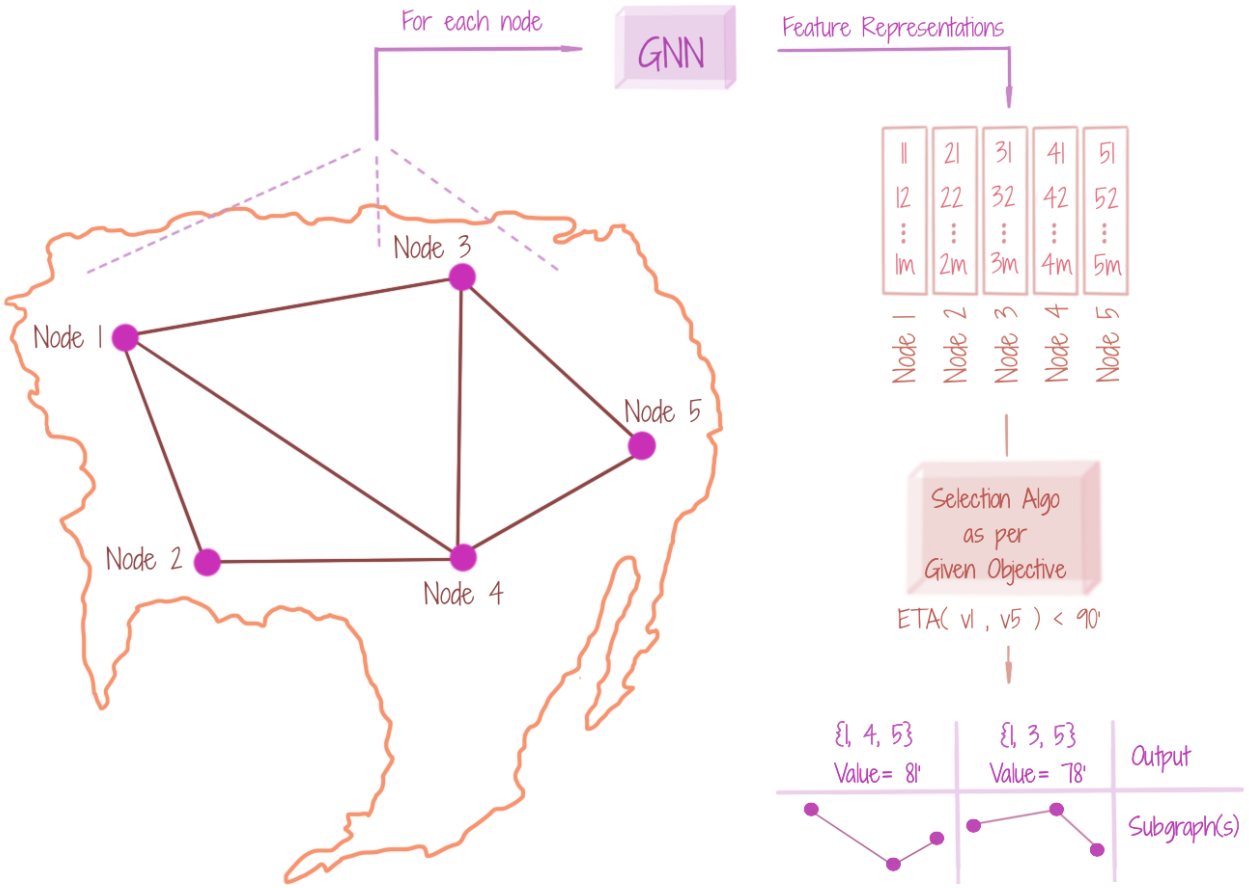


Fig. 29 - Structure Mining the Non-Steiner Way

The figure shows that we want to find the subgraph connecting towns v1 and v5, such that the ETA (estimated time of travel) is less than 90 minutes.
Here, an edge e between any two nodes n and m, carries the attribute of amount of traffic/roundabouts/railway barriers. Thus, it would be advisable to learn this in a data-driven manner, using GNNs.

Steiner Way

For an input graph G, when we want to **find a subgraph matching a specific objective**, using data-driven procedures, we may land into a situation where **no subgraph matches the criteria.**

- In such cases, it may be interesting to *somewhat* expand the graph (to additional nodes & edges),
- in a manner that including these new nodes and edges, might increase the possibility of finding the optimal subgraph.

➢ Since we consider additional graph entities for optimizing our objective, the mining is Steiner and the additional entities may be called Steiner nodes and Steiner edges**.**

We emphasize the Steiner structure mining, due to recent success in AI methods for proving mathematical theorems, where estimating an expansion of the concept is very helpful.

Alternatively, it can be seen as an expansive reasoning mechanism, which tells us that if something is not happening, what more might make it happen.

The Steiner structure mining has a **generative sense**, since extra nodes and edges are augmented to the graph. However, the generation may be seen as somewhat **limited**.

(Fig. 30, pg. 48)

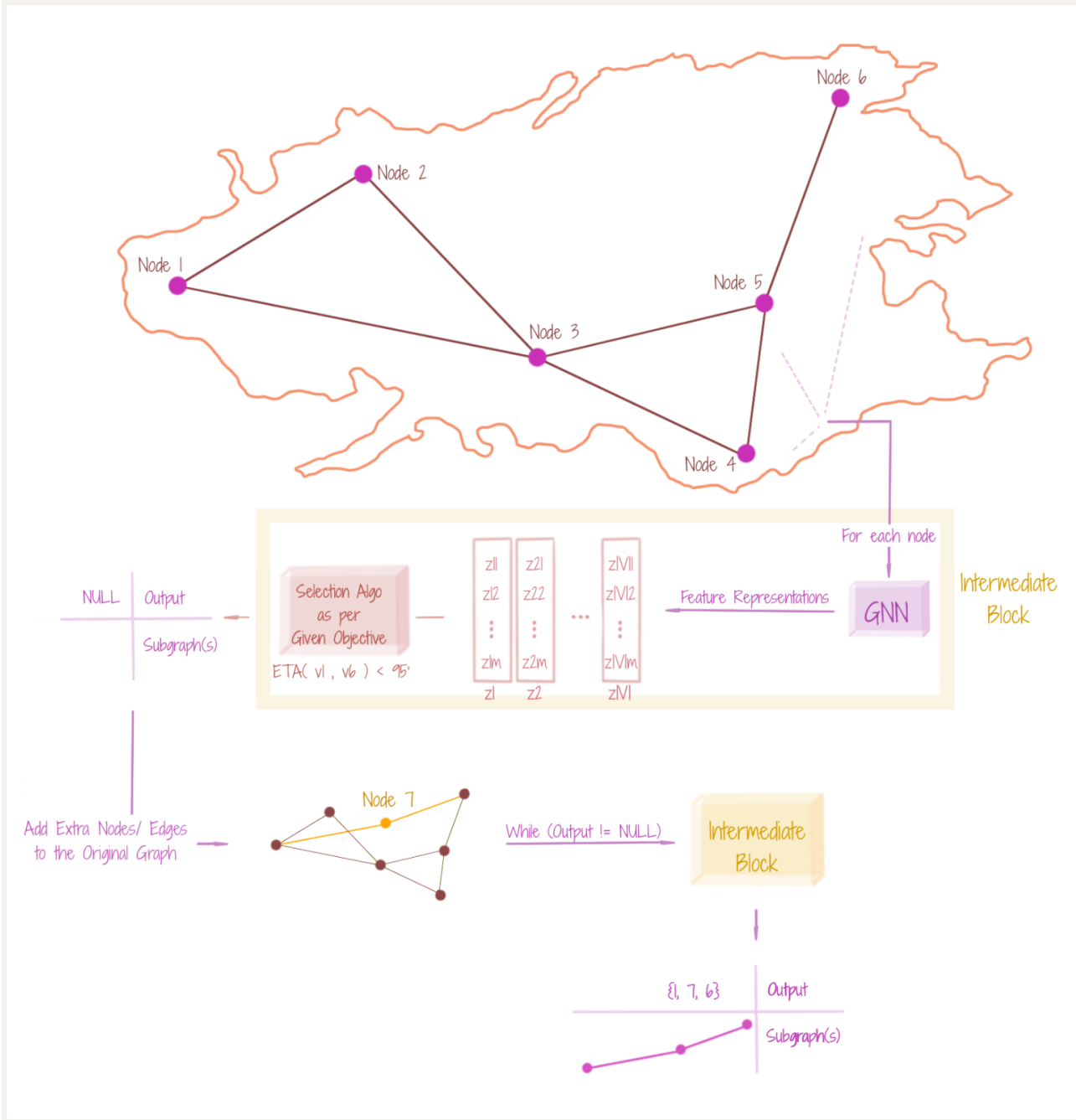


Fig. 30 - Structure Mining the Steiner Way

The figure shows that for optimizing through a potential superset of nodes and edges, one would need to iterate on the typical non-Steiner structure mining procedure.

Back to Top

Generation Tasks

The aim is to generate new graphs.

A graph may be **generated** with:

- a completing/new topology, and new node and/or edge attributes,
- just new node and/or edge attributes, while keeping the same topology.

With a **new topological structure**, a graph can be generated unconditionally, or conditionally (from an existing subgraph) to reflect a chosen concept.

Alternatively, a graph may be evolved from an existing graph, keeping the **same topology**, but modifying the node and edge attributes, to model some system dynamics.

Hence Generation Tasks can be categorized into the following:

a. Completion
b. Evolution.

Fig. 31 - Classification Tasks

Back to Top

Completion

Given a **dataset of graphs**, a **distribution** can be learnt over those graphs, and

- a new graph may be **unconditionally generated**, by sampling from that distribution, or
- a new graph may also be **conditionally generated** (from the same distribution), for instance, to complete a partial graph.

The graph generation may be done

- sequentially (usually achieved through Reinforcement Learning/RNN-based procedures), or
- in one-shot (mostly through encoder-decoder style frameworks).

Note that graph generation (for completion of a graph or a completely new graph) **invites a new topological structure.**

(Fig. 32, pg. 51)

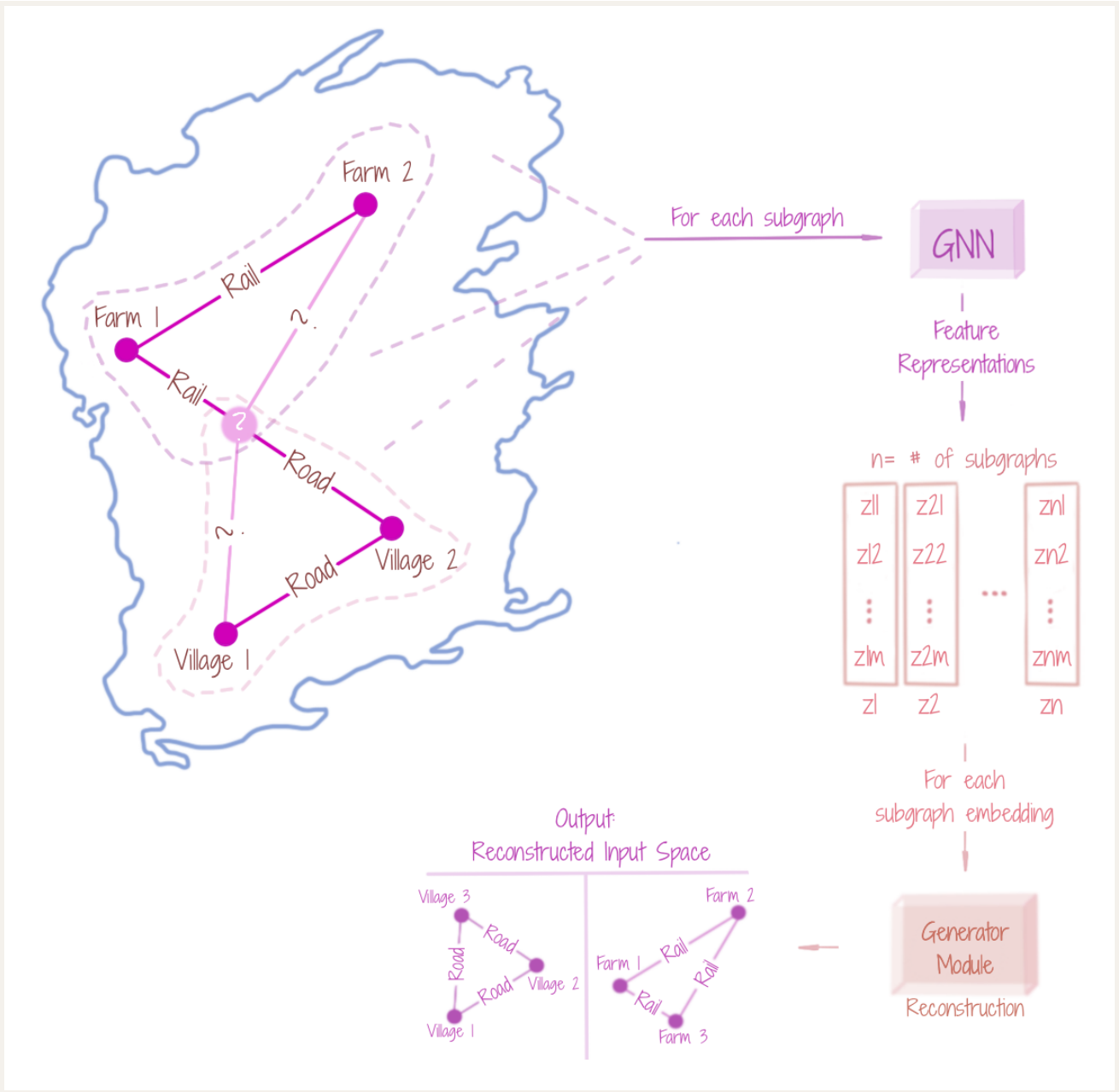


Fig. 32 - Completion as Graph Generation

The figure shows that a GNN can be used to learn embeddings (encodings) of the various (randomly-selected) subgraphs (of the input graphs), which may be then passed to a Generator (Decoder) module, learning to reconstruct the graph unconditionally or conditionally.

GNN-based graph generation has analogues to data-driven image and 3D mesh model generation, where the first aim is to learn a well-structured distribution of the input space, to sample from, later.

Evolution

A graph may be **generated** not to add or construct a new topological structure, but only to **update its node and edge attributes, under the same topology**, this is called graph evolution.

Graph evolution is generally used to model the temporal dynamics of physical systems, for instance, particle-particle interactions in physics may be modeled in a graph as node-node interactions with edge attributes.

- During evolution, if additional nodes and edges are also incorporated, we would say that the graph is evolved with a flavor of topological generation.

(Fig. 33, pg. 53)

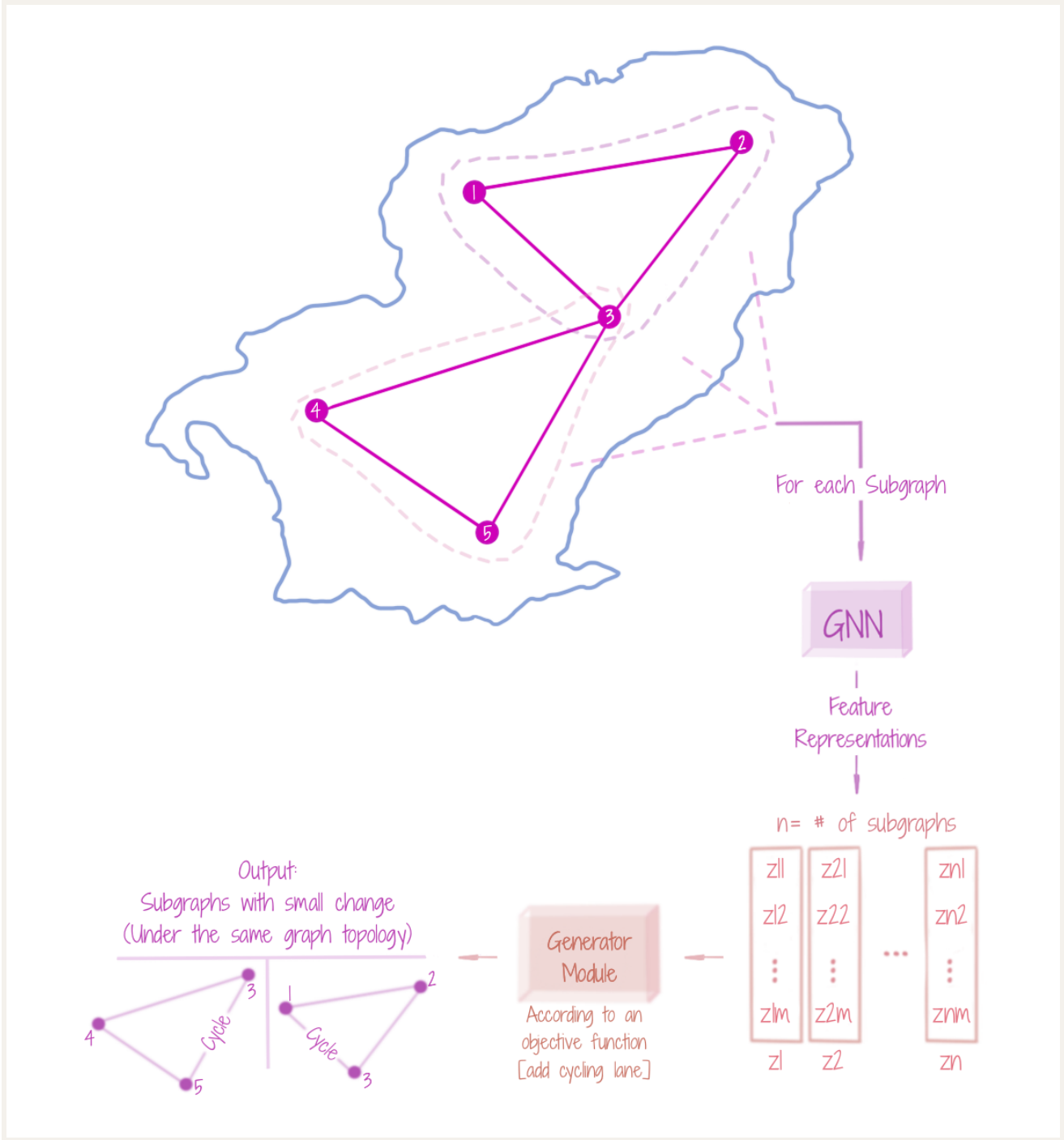


Fig. 33 - Evolution as Graph Generation

The figure shows that encodings for a number of input subgraphs are learnt through a GNN, which are then learnt to be decoded through a Generator (Decoder) module, that evolves a given input graph, owing to specific criteria.

Useful Readings

[1] Riihimäki, H., 2022. Simplicial $ q $-connectivity of directed graphs with applications to network analysis. arXiv preprint arXiv:2202.07307

[2] Hilbert, D., 1950. The foundations of geometry. Prabhat Prakashan.

[3] Kuryliak, Y., Emmerich, M. and Dosyn, D., 2021. On the Effect of Complex Network Topology in Managing Epidemic Outbreaks. In MoMLeT+ DS (pp. 1-15).

[4] Torres, L., Blevins, A.S., Bassett, D. and Eliassi-Rad, T., 2021. The why, how, and when of representations for complex systems. SIAM Review, 63(3), pp.435-485.

[5] Tinarrage, R., 2021. Simplicial approximation to CW complexes in practice. arXiv preprint arXiv:2112.07573.

[6] Erickson, J., 2020. One-dimensional Computational Topology, Lecture Notes in Computer Science, University of Illinois Urbana Champaign

[7] Leksovec, J., 2021. Machine Learning with Graphs, Lecture Notes in Computer Science, Stanford University

[8] Xu, J. and Picek, S., 2021. Watermarking Graph Neural Networks based on Backdoor Attacks. arXiv preprint arXiv:2110.11024.

[9] Pei, H., Wei, B., Chang, K.C.C., Lei, Y. and Yang, B., 2020. Geom-gcn: Geometric graph convolutional networks. arXiv preprint arXiv:2002.05287

[10] Dai, E., Jin, W., Liu, H. and Wang, S., 2022, February. Towards robust graph neural networks for noisy graphs with sparse labels. In Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining (pp. 181-191).

[11] Lovász, L., 2010. Discrete and continuous: two sides of the same?. In Visions in mathematics (pp. 359-382). Birkhäuser Basel.

[12] Sato, R., 2020. A survey on the expressive power of graph neural networks. arXiv preprint arXiv:2003.04078.

[13] Bianchi, F.M., Grattarola, D. and Alippi, C., 2019. Mincut pooling in graph neural networks.

[14] Li, W., Li, R., Ma, Y., Chan, S.O. and Yu, B., 2020. Rethinking Graph Neural Networks for Graph Coloring.

[15] Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C. and Philip, S.Y., 2020. A comprehensive survey on graph neural networks. IEEE transactions on neural networks and learning systems, 32(1), pp.4-24.

[16] Bronstein, M.M., Bruna, J., Cohen, T. and Veličković, P., 2021. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges. arXiv preprint arXiv:2104.13478.

[17] Hamilton, W.L., Ying, R. and Leskovec, J., 2017. Representation learning on graphs: Methods and applications. arXiv preprint arXiv:1709.05584.

[18] Li, Y., Vinyals, O., Dyer, C., Pascanu, R. and Battaglia, P., 2018. Learning deep generative models of graphs. arXiv preprint arXiv:1803.03324.

[19] Ahmed, R., Turja, M.A., Sahneh, F.D., Ghosh, M., Hamm, K. and Kobourov, S., 2021. Computing Steiner Trees using Graph Neural Networks. arXiv preprint arXiv:2108.08368.

[20] Zhu, Y., Du, Y., Wang, Y., Xu, Y., Zhang, J., Liu, Q. and Wu, S., 2022. A Survey on Deep Graph Generation: Methods and Applications. arXiv preprint arXiv:2203.06714.

[21] You, J., Ying, R., Ren, X., Hamilton, W. and Leskovec, J., 2018, July. Graphrnn: Generating realistic graphs with deep auto-regressive models. In the International conference on machine learning (pp. 5708-5717). PMLR.

[22] Zang, C. and Wang, F., 2020, August. MoFlow: an invertible flow model for generating molecular graphs. In Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (pp. 617-626).

[23] Zhu, Y., Xu, W., Zhang, J., Du, Y., Zhang, J., Liu, Q., Yang, C. and Wu, S., 2021. A Survey on Graph Structure Learning: Progress and Opportunities. arXiv e-prints, pp.arXiv-2103.

[24] Gilmer, J., Schoenholz, S.S., Riley, P.F., Vinyals, O. and Dahl, G.E., 2017, July. Neural message passing for quantum chemistry. In the International conference on machine learning (pp. 1263-1272). PMLR.

[25] Yehudai, G., Fetaya, E., Meirom, E., Chechik, G. and Maron, H., 2021, July. From local structures to size generalization in graph neural networks. In International Conference on Machine Learning (pp. 11975-11986). PMLR.

[26] Sanchez-Gonzalez, A., Godwin, J., Pfaff, T., Ying, R., Leskovec, J. and Battaglia, P., 2020, November. Learning to simulate complex physics with graph networks. In the International Conference on Machine Learning (pp. 8459-8468). PMLR.

[27] Zhang, M. and Chen, Y., 2018. Link prediction based on graph neural networks. Advances in neural information processing systems, 31.

[28] Lample, G., Lachaux, M.A., Lavril, T., Martinet, X., Hayat, A., Ebner, G., Rodriguez, A. and Lacroix, T., 2022. HyperTree Proof Search for Neural Theorem Proving. arXiv preprint arXiv:2205.11491.

[29] Mitchell, J.S., 2000. Geometric Shortest Paths and Network Optimization. Handbook of computational geometry, 334, pp.633-702.

[30] Nash, C., Ganin, Y., Eslami, S.A. and Battaglia, P., 2020, November. Polygen: An autoregressive generative model of 3d meshes. In the International conference on machine learning (pp. 7220-7229). PMLR.