Εθνικο Μετσοβιο Πολυτεχνειο

Σχολη Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων

Τομεας Τεχνολογιας Πληροφορικης και Υπολογιστων

Class-Representative Graph Summaries Through Graph Matching Networks

Diploma Thesis

by

Achilleas Tsimichodimos

Επιβλέπων: Γεώργιος Στάμου
Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούλιος 2024

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών
Εργαστήριο Συστημάτων Τεχνητής Νοημοσύνης και Μάθησης

Class-Representative Graph Summaries Through Graph Matching Networks

Diploma Thesis

by

Achilleas Tsimichodimos

Επιβλέπων: Γεώργιος Στάμου
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 15η Ιουλίου, 2024.

.....	.....	.....
Γεώργιος Στάμου	Αθανάσιος Βουλόδημος	Ανδρέας-Γεώργιος Σταφυλοπάτης
Καθηγητής Ε.Μ.Π.	Επ. Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούλιος 2024

.....

Αχιλλεας Τσιμιχοδημος
Διπλωματούχος Ηλεκτρολόγος Μηχανικός
και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περίληψη

Τα Νευρωνικά Δίκτυα Γράφων (ΝΔΓ) έχουν αναδειχθεί ως ένα σημαντικό μοντέλο στον τομέα της μηχανικής μάθησης, λόγω της έμφυτης ικανότητάς τους να χειρίζονται δεδομένα δομημένα ως γραφήματα. Η αναπαραστατική δύναμη των ΝΔΓ έχει οδηγήσει σε σημαντική πρόοδο σε διάφορους τομείς, όπως η ανάλυση κοινωνικών δικτύων, τα βιολογικά δίκτυα, η μοριακή χημεία και τα συστήματα συστάσεων. Στους τομείς αυτούς, η σύνοψη γράφων παίζει καθοριστικό ρόλο στη δημιουργία συνοπτικών αναπαραστάσεων μεγάλων γράφων, διατηρώντας παράλληλα τις ουσιώδεις δομικές ιδιότητες και πληροφορίες. Τα ΝΔΓ παρέχουν ένα ισχυρό πλαίσιο για αποδοτική και αποτελεσματική σύνοψη, επιτρέποντας την εξαγωγή σημαντικών πληροφοριών από σύνθετα και μεγάλης κλίμακας δεδομένα γράφων.

Στην παρούσα διπλωματική εργασία, θα αντιμετωπίσουμε το πρόβλημα της σύνοψης γράφων εξερευνώντας την αναπαραστατική δύναμη των Δικτύων Αντιστοίχισης Γράφων (ΔΑΓ), ενός εξειδικευμένου τύπου ΝΔΓ που υπολογίζει μετρικές ομοιότητας μεταξύ ζευγών γράφων μέσω ενός μηχανισμού αντιστοίχισης βασισμένου στην προσοχή μεταξύ γράφων. Συγκεκριμένα, δεδομένου ενός συνόλου δεδομένων γράφων με πολλαπλές κλάσεις, στοχεύουμε να εξαγάγουμε από κάθε γράφο έναν υπογράφο, ο οποίος διατηρεί τα βασικά χαρακτηριστικά της κλάσης στην οποία ανήκει ο αρχικός γράφος. Για τον σκοπό αυτό, εκπαιδεύουμε ένα ΔΑΓ σε ένα πρόβλημα ομοιότητας γράφων και αναπτύσσουμε μεθοδολογίες για την αναγνώριση των μοτίβων που μαθαίνει το μοντέλο κατά την εκπαίδευση, τα οποία θα καθοδηγήσουν τη διαδικασία δημιουργίας συνόψεων. Για την αξιολόγηση της απόδοσης της προσέγγισής μας, δημιουργούμε ένα συνθετικό σύνολο δεδομένων γεωμετρικών σχημάτων, ενισχυμένο με θόρυβο, το οποίο παρέχει ένα ελεγχόμενο περιβάλλον με γνωστό ground truth για ακριβή αξιολόγηση, και το συγκρίνουμε με υπάρχουσες αρχιτεκτονικές ΝΔΓ για τη σύνοψη γράφων. Επιπλέον, πειραματιζόμαστε με το σύνολο δεδομένων MUTAG, το οποίο δεν διαθέτει ground truth, αλλά διαθέτει πρωτότυπα (prototypes), καθοδηγώντας την ποιοτική μας αξιολόγηση. Τα αποτελέσματα, αξιολογημένα τόσο σε ποσοτικό όσο και σε ποιοτικό επίπεδο, υποδεικνύουν ότι το ΔΑΓ έχει καλύτερη απόδοση σε σύγκριση με τα άλλα μοντέλα και είναι σε θέση να αναγνωρίσει με συνέπεια ακριβείς συνόψεις που αντιπροσωπεύουν τις κλάσεις, οι οποίες προσφέρουν πολύτιμες πληροφορίες για τα μοτίβα που μαθαίνει το μοντέλο κατά την εκπαίδευση και για τις διαδικασίες λήψης αποφάσεών του.

Λέξεις-κλειδιά — Νευρωνικά Δίκτυα Γράφων, Δίκτυα Αντιστοίχισης Γράφων, Σύνοψη Γράφων, Ομοιότητα Γράφων, Πρωτότυπα Γράφων

Abstract

Graph Neural Networks (GNNs) have emerged as a key model in the field of machine learning, due to their inherent ability to handle graph-structured data. The representation power of GNNs has led to significant advancements in diverse fields, including social network analysis, biological networks, molecular chemistry, and recommendation systems. In these domains, graph summarization plays an important role in creating compact representations of large graphs while preserving essential structural properties and information. In this context, GNNs provide a powerful framework for efficient and effective graph summarization, enabling the extraction of meaningful insights from complex and large-scale graph data.

In this thesis, we will address the problem of graph summarization by exploring the representation power of Graph Matching Networks (GMNs), a specialized type of GNNs that computes similarity scores between pairs of graphs through a cross-graph attention-based matching mechanism. Specifically, given a multi-class graph dataset, we aim to extract a class-representative subgraph from each graph, that effectively represents the class to which the graph belongs. To this end, we train the GMN on a graph similarity task and propose two methodologies to identify the patterns learned by the model during training, which will inform the summary creation process. To assess the performance of our approach, we create a synthetic dataset of Geometric Shapes, enhanced with noise, which provides a controlled environment with known ground truth summaries for precise evaluation, and compare it against existing GNN architectures for graph summarization. Additionally, we experiment with the real-world MUTAG dataset, which lacks ground truth summaries, but offers ground truth prototypes, guiding our qualitative evaluation. The results, evaluated both at quantitative and qualitative level, indicate that the GMN outperforms the other models and is able to consistently and accurately identify class-representative summaries, which offer valuable insights into the patterns that the model learns during training and its decision making processes, enhancing its explainability.

Keywords — Graph Neural Networks, Graph Matching Networks, Graph Summarization, Graph Similarity, Graph Prototypes

Ευχαριστίες

Θα ήθελα να ευχαριστήσω θερμά τον επιβλέποντά μου, κ. Γεώργιο Στάμου, και τον κ. Αθανάσιο Βουλόδημο για την ευκαιρία που μου έδωσαν να εκπονήσω τη διπλωματική μου εργασία στο AILS Lab και για τη διαρκή βοήθεια και καθοδήγηση που μου έχουν προσφέρει. Θα ήθελα επίσης να ευχαριστήσω την Αγγελική Δημητρίου και τον Νίκο Χάιδο για την πολύτιμη βοήθεια που μου παρείχαν καθ' όλη τη διάρκεια αυτής της εργασίας, η ολοκλήρωση της οποίας δεν θα ήταν εφικτή χωρίς τη συμβολή τους. Τέλος, ευχαριστώ την οικογένειά μου, που με ενθαρρύνει και με στηρίζει σε κάθε μου βήμα, και τους φίλους μου, με τους οποίους μοιραστήκαμε αυτό το ακαδημαϊκό ταξίδι.

Αχιλλέας Τσιμιχόδημος, Ιούλιος 2024

Contents

Contents xi

List of Figures xiii

1 Εκτεταμένη Περίληψη στα Ελληνικά 1
- 1.1 Θεωρητικό Υπόβαθρο . . . 2
  - 1.1.1 Θεωρία Γράφων . . . 2
  - 1.1.2 Αντιστοίχιση και Ομοιότητα Γράφων . . . 3
  - 1.1.3 Νευρωνικά Δίκτυα Γράφων . . . 5
  - 1.1.4 Σύνοψη Γράφων . . . 8
- 1.2 Προτεινόμενες Προσεγγίσεις . . . 10
  - 1.2.1 Συνεισφορά . . . 10
  - 1.2.2 Ορισμός Προβλήματος . . . 11
  - 1.2.3 Προτεινόμενο Μοντέλο . . . 11
- 1.3 Πειραματικό μέρος . . . 15
  - 1.3.1 Σύνολα Δεδομένων . . . 15
  - 1.3.2 Μετρικές Αξιολόγησης . . . 16
  - 1.3.3 Βασικά Μοντέλα . . . 17
  - 1.3.4 Λεπτομέρειες Μοντέλων . . . 17
  - 1.3.5 Αποτελέσματα . . . 18
- 1.4 Συμπεράσματα . . . 25
  - 1.4.1 Συζήτηση . . . 25
  - 1.4.2 Μελλοντικές Κατευθύνσεις . . . 26

2 Introduction 27

3 Background 29
- 3.1 Machine Learning . . . 30
  - 3.1.1 Categories of Learning . . . 30
  - 3.1.2 Data Modalities . . . 30
- 3.2 Deep Learning . . . 31
  - 3.2.1 Core Components of Neural Networks . . . 31
  - 3.2.2 Training Neural Networks . . . 33
  - 3.2.3 Evaluation and Validation . . . 37
  - 3.2.4 Embeddings . . . 38
  - 3.2.5 Transformers . . . 38

4 Graphs 41
- 4.1 Graph Theory . . . 42
- 4.2 Graph Similarity . . . 43
  - 4.2.1 Graph Edit Distance . . . 43
  - 4.2.2 Graph Kernels . . . 44
- 4.3 Graph Matching . . . 46

4.3.1 Exact Graph Matching (Graph Isomorphism) . . . 46
4.3.2 Inexact Graph Matching . . . 47
4.4 Maximum Common Subgraph . . . 47
4.4.1 Maximum Common Induced Subgraph (MCIS) . . . 47
4.4.2 Maximum Common Edge Subgraph (MCES) . . . 47

5 Graph Neural Networks (GNN) 49
5.1 Machine Learning on Graphs . . . 50
5.1.1 Motivation . . . 50
5.1.2 Permutation Invariance and Equivariace . . . 50
5.2 Graph Convolution . . . 52
5.2.1 Spectral Graph Convolution . . . 52
5.2.2 Spatial Graph Convolution . . . 53
5.3 GNN Architectures . . . 54
5.3.1 Spectral Variants . . . 54
5.3.2 Spatial Variants . . . 55
5.4 Graph Matching Networks . . . 58
5.4.1 Graph Matching Network Architecture . . . 58
5.4.2 Training Graph Matching Networks . . . 59

6 Graph Summarization 61
6.1 Definitions . . . 62
6.2 Prior Work . . . 62
6.2.1 Traditional Methods . . . 62
6.2.2 GNN-based Methods . . . 63

7 Proposal 67
7.1 Contributions . . . 67
7.2 Problem Definition . . . 67
7.3 Proposed Model . . . 68
7.3.1 Method I . . . 68
7.3.2 Method II . . . 70

8 Experiments 73
8.1 Preliminaries . . . 74
8.1.1 Datasets . . . 74
8.1.2 Evaluation Metrics . . . 75
8.1.3 Baseline Models . . . 75
8.2 Training and Inference Details . . . 76
8.2.1 GMN . . . 76
8.2.2 Baseline Models . . . 76
8.3 Results . . . 77
8.3.1 Quantitative Results . . . 77
8.3.2 Qualitative Results . . . 79

9 Conclusion 84
9.1 Discussion . . . 84
9.2 Future Work . . . 85

10 Bibliography 86

List of Figures

1.1.1 Ένα παράδειγμα κατευθυνόμενου και μη κατευθυνόμενου γράφου. . . . 3
1.1.2 Μια οπτική απεικόνιση δύο ισομορφικών γράφων. . . . 4
1.1.3 Απόσταση Επεξεργασίας Γράφων μεταξύ ενός ζεύγους γράφων. . . . 4
1.1.4 Απεικόνιση ενός Νευρωνικού Δικτύου Γράφων (αριστερά) και ενός Δικτύου Αντιστοίχισης Γράφων (δεξιά) [46]. . . . 8
1.1.5 Μια αρχιτεκτονική ΝΔΓ όπου τα επίπεδα μεταφοράς μηνυμάτων ακολουθούνται από ένα επίπεδο MinCutPool [4]. . . . 10
1.2.1 Επισκόπηση της Μεθόδου I. . . . 13
1.2.2 Επισκόπηση της Μεθόδου II. . . . 15
1.3.1 Τα τέσσερα βασικά γεωμετρικά σχήματα στο συνόλου δεδομένων Geometric Shapes και παραδείγματα γράφων με θόρυβο. . . . 16
1.3.2 Παραδείγματα γράφων από κάθε μία από τις δύο κλάσεις στο σύνολο δεδομένων MUTAG (μη μεταλλαξιογόνος και μεταλλαξιογόνος), μαζί με τα πρωτότυπα του συνόλου δεδομένων. . . . 16
1.3.3 Παραδείγματα ακριβούς ταιριάσματος για το σύνολο δεδομένων Geometric Shapes (8). . . . 21
1.3.4 Παραδείγματα προσεγγιστικού ταιριάσματος για το σύνολο δεδομένων Geometric Shapes (8). . . . 21
1.3.5 Λανθασμένα παραδείγματα για το σύνολο δεδομένων Geometric Shapes (8). . . . 22
1.3.6 Ακριβή παραδείγματα για την μη μεταλλαξιογόνο κλάση του συνόλου δεδομένων MUTAG. . . . 22
1.3.7 Ακριβή παραδείγματα για την μεταλλαξιογόνο κλάση του dataset MUTAG. . . . 23
1.3.8 Λανθασμένα παραδείγματα με περισσότερους κόμβους για την μη μεταλλαξιογόνο κλάση (αριστερά) και την μεταλλαξιογόνο κλάση (δεξιά). . . . 23
1.3.9 Λανθασμένα παραδείγματα με λιγότερους κόμβους για την μη μεταλλαξιογόνο κλάση (αριστερά) και την μεταλλαξιογόνο κλάση (δεξιά). . . . 24
1.3.10Σύγκριση μεθόδων δημιουργίας συνόψεων. Το πρώτο σχήμα δείχνει τη σύνοψη που δημιουργήθηκε εξάγοντας μια σύνοψη στο τέλος, χρησιμοποιώντας όλα τα ζεύγη αντιστοίχισης. Το δεύτερο σχήμα δείχνει τη διαδοχική δημιουργία συνόψεων από τα ζεύγη αντιστοίχισης σε κάθε επίπεδο. . . . 24

3.2.1 An illustration of the structure of a biological neuron and an artificial neuron (perceptron). . . . 32
3.2.2 An illustration of the diagrams of the discussed Activation Functions. . . . 34
3.2.3 An illustration of the Transformer - model architecture, as presented in [64]. . . . 39

4.1.1 An example of a directed and an undirected graph. . . . 42
4.2.1 Graph Edit Distance between a pair of graphs. . . . 44
4.2.2 An illustration of the computation of the Weisfeiler-Lehman kernel for two graphs [61]. . . . 45
4.3.1 A visual illustration of two isomorphic graphs. . . . 46
4.4.1 Given the graphs G and H presented in Fig.(a), Fig.(b) reports 4 different possible MCSs. [50] 48

5.1.1 A comparison between a Euclidean and a Non-Euclidean space. . . . 50
5.1.2 Geometric Deep Learning blueprint, exemplified on a graph. A GNN architecture may contain permutation equivariant layers, local pooling, and a permutation invariant global pooling layer [10]. . . . 52

5.3.1 **Left:** The attention mechanism $\alpha(\mathbf{W}\vec{h}_i, \mathbf{W}\vec{h}_j)$ employed by GAT, parametrized by a weight vector $\vec{a} \in \mathbb{R}^{2F'}$, applying a LeakyReLU activation. **Right:** An illustration of multi-head attention ($K = 3$ heads) by node 1 on its neighborhood. Different arrow styles and colors denote independent attention computations. The aggregated features from each head are concatenated or averaged to obtain $\vec{h}'_1$ [65]. 56
5.4.1 Illustration of the graph embedding (left) and matching models (right) [46]. 59
5.4.2 Visualization of cross-graph attention for GMNs after 5 propagation layers. In each pair of graphs the left figure shows the attention from left graph to the right and the right figure shows the opposite [46]. 59

6.2.1 An illustrative example of graph pooling [47]. 63
6.2.2 A deep GNN architecture where message-passing is fol- lowed by the MinCutPool layer [4]. . 65

7.3.1 Overview of Method I. 69
7.3.2 Overview of Method II. 72

8.1.1 The four core geometric shapes in the Geometric Shapes dataset and examples of noisy graphs. 74
8.1.2 Examples of graphs from each of the two classes in the MUTAG dataset (non-mutagenic and mutagenic), along with the prototypes of the dataset. 75
8.3.1 Exact match examples for the Geometric Shapes (8) dataset. 79
8.3.2 Approximate match examples for the Geometric Shapes (8) dataset. 80
8.3.3 Incorrect examples for the Geometric Shapes (8) dataset. 80
8.3.4 Accurate examples for the non-mutagenic class of the MUTAG dataset. 81
8.3.5 Accurate examples for the mutagenic class of the MUTAG dataset. 81
8.3.6 Incorrect examples with more nodes for the non-mutagenic (left) and mutagenic (right) classes. 81
8.3.7 Incorrect examples with less nodes for the non-mutagenic (left) and mutagenic (right) classes. 82
8.3.8 Comparison of summary generation methods. Subfigure (a) shows the summary generated by extracting one summary at the end using all matching pairs. Subfigure (b) demonstrates the sequential generation of summaries from the matching pairs at each layer. 83

Chapter 1

Εκτεταμένη Περίληψη στα Ελληνικά

1.1 Θεωρητικό Υπόβαθρο

Οι πρόσφατες εξελίξεις στον τομέα της Τεχνητής Νοημοσύνης (ΤΝ) έχουν επιφέρει σημαντικές αλλαγές σε διάφορους τομείς, οδηγώντας σε σημαντική πρόοδο σε περιοχές όπως η αναγνώριση εικόνας, η επεξεργασία φυσικής γλώσσας και η βιοπληροφορική. Μεταξύ αυτών των εξελίξεων, τα Νευρωνικά Δίκτυα Γράφων (ΝΔΓ) έχουν αναδειχθεί ως βασικά εργαλεία για την ανάλυση και κατανόηση σύνθετων δεδομένων δομημένων ως γραφήματα, αξιοποιώντας την έμφυτη ιεραρχική πληροφορία που υπάρχει σε αυτά. Έχουν δείξει αξιοσημείωτη επιτυχία σε πολλές εφαρμογές, όπως η ανάλυση κοινωνικών δικτύων, τα βιολογικά δίκτυα, η ανακάλυψη φαρμάκων και τα συστήματα συστάσεων.

Το επίκεντρο αυτής της διπλωματικής εργασίας έγκειται στην αντιμετώπιση του προβλήματος της Σύνοψης Γράφων. Δεδομένου ενός συνόλου δεδομένων γράφων πολλαπλών κλάσεων, στόχος μας είναι να αναπτύξουμε μεθοδολογίες για την εξαγωγή ενός υπογράφου από κάθε γράφο που αντιπροσωπεύει αποτελεσματικά την κλάση στην οποία αυτός ανήκει. Τα βασικά μοντέλα που θα χρησιμοποιήσουμε σε αυτήν την εργασία είναι τα Δίκτυα Αντιστοίχισης Γράφων (ΔΑΓ), ένας εξειδικευμένος τύπος ΝΔΓ που, δεδομένου ενός ζεύγους γράφων ως είσοδο, υπολογίζουν μία μετρική ομοιότητας μεταξύ τους με τη χρήση ενός μηχανισμού αντιστοίχισης βασισμένου στην προσοχή μεταξύ των γράφων. Εκπαιδεύοντας το μοντέλο σε ένα πρόβλημα ομοιότητας γράφων, στοχεύουμε να αναπτύξουμε μεθοδολογίες για την αναγνώριση και εξαγωγή των μοτίβων που έχει μάθει το μοντέλο, προκειμένου να δημιουργήσουμε τις συνοπτικές αναπαραστάσεις που αντιπροσωπεύουν τις κλάσεις.

Επιπλέον, η αυξανόμενη ανάγκη για Ερμηνεύσιμη Τεχνητή Νοημοσύνη (ΧΑΙ) έχει γίνει κρίσιμη λόγω της ενσωμάτωσης των νευρωνικών δικτύων σε πολλούς τομείς, όπως η υγειονομική περίθαλψη και τα αυτόνομα οχήματα. Η φύση των μοντέλων αυτών ως «μαύρα κουτιά» εγείρει ερωτήματα σχετικά με την αξιοπιστία τους, καθιστώντας αναγκαία την ανάπτυξη μεθοδολογιών που ενισχύουν τη διαφάνεια και την κατανόηση των διαδικασιών λήψης αποφάσεων των μοντέλων. Η σύνοψη σύνθετων γράφων και η δημιουργία αναπαραστάσεων που αντιπροσωπεύουν τις κλάσεις, αναγνωρίζοντας τα μοτίβα που έχουν μάθει τα ΝΔΓ, μπορεί να συμβάλει στην καλύτερη κατανόηση των προβλέψεών τους και στην ενίσχυση της ερμηνευσιμότητάς τους.

1.1.1 Θεωρία Γράφων

Στα διακριτά μαθηματικά, ένας γράφος ή ένα γράφημα είναι μια αφηρημένη αναπαράσταση ενός συνόλου στοιχείων, όπου μερικά ζεύγη στοιχείων συνδέονται μεταξύ τους με δεσμούς. Τυπικά, ένας γράφος σημειώνεται ως $G = (V, E)$, όπου V είναι ένα σύνολο κορυφών, και E είναι ένα σύνολο ακμών. Οι κορυφές u και v μιας ακμής {u, v} ονομάζονται άκρα της ακμής. Η σειρά ενός γράφου είναι ο αριθμός των κορυφών $|V|$ και συνήθως συμβολίζεται ως n. Το μέγεθος ενός γράφου είναι ο αριθμός των ακμών $|E|$ και συνήθως συμβολίζεται ως m.

Αναφορικά με την συνδεσιμότητα, οποιεσδήποτε δύο κορυφές σε έναν γράφο μπορεί να συνδέονται με καμία, μία ή πολλαπλές ακμές. Οι γράφοι που επιτρέπουν πολλαπλές ακμές να έχουν τα ίδια άκρα ονομάζονται πολυγραφήματα ή πολύγραφοι. Μερικές φορές, οι γράφοι επιτρέπεται να περιέχουν βρόχους, οι οποίοι είναι ακμές που ενώνουν μια κορυφή με τον εαυτό της. Ο βαθμός μιας κορυφής είναι ο αριθμός των ακμών που συνδέονται με αυτήν,. Σε έναν γράφο τάξης n, ο μέγιστος βαθμός κάθε κορυφής είναι $n - 1$ (ή $n + 1$ αν επιτρέπονται οι βρόχοι, επειδή ένας βρόχος συμβάλλει με 2 στον βαθμό), και ο μέγιστος αριθμός ακμών είναι $n(n - 1)/2$ (ή $n(n + 1)/2$ αν επιτρέπονται οι βρόχοι).

Οι γράφοι μπορούν να κατηγοριοποιηθούν με βάση την κατεύθυνση των ακμών τους:

- **Κατευθυνόμενοι Γράφοι**: Στους κατευθυνόμενους γράφους, κάθε ακμή έχει μια κατεύθυνση συνδεδεμένη με αυτήν και συνήθως αναπαριστάται με ένα βέλος που δείχνει από τη μία κορυφή στην άλλη. Αυτή η κατεύθυνση υποδηλώνει την ασυμμετρία στη σχέση, που σημαίνει ότι η σύνδεση από την κορυφή u στην κορυφή v δεν υποδηλώνει σύνδεση από v σε u.
- **Μη Κατευθυνόμενοι Γράφοι**: Στους μη κατευθυνόμενους γράφους, οι ακμές είναι διπλής κατεύθυνσης, υποδηλώνοντας ότι κάθε σύνδεση είναι συμμετρική. Έτσι, μια ακμή από u σε v συνεπάγεται αυτόματα μια ακμή από v σε u.

Ένας γράφος καθορίζεται πλήρως και μπορεί να αναπαρασταθεί από τον πίνακα γειτνίασης ή τη λίστα γειτνίασής του:

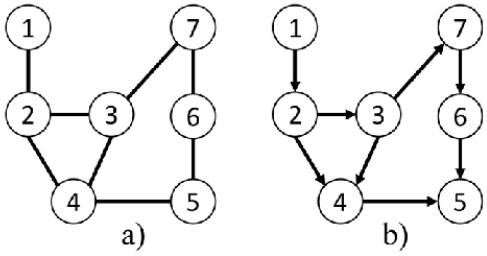


Figure 1.1.1: Ένα παράδειγμα κατευθυνόμενου και μη κατευθυνόμενου γράφου.

- **Πίνακας Γειτνίασης**: Ένας πίνακας γειτνίασης A ενός γράφου σειράς n είναι ένας πίνακας $n \times n$ όπου τα μη μηδενικά στοιχεία υποδεικνύουν την παρουσία μιας ακμής μεταξύ των κορυφών. Στους μη κατευθυνόμενους γράφους, αυτός ο πίνακας είναι συμμετρικός.
- **Λίστα Γειτνίασης**: Η λίστα γειτνίασης είναι ένας πιο αποδοτικός τρόπος αναπαράστασης γράφων, ιδιαίτερα αραιών γράφων. Κάθε κορυφή διατηρεί μια λίστα όλων των κορυφών με τις οποίες είναι άμεσα συνδεδεμένη, διευκολύνοντας την αποδοτικότερη διάσχιση.

Οι κορυφές και οι ακμές μπορούν επίσης να έχουν χαρακτηριστικά ή βάρη συνδεδεμένα με αυτές, που συχνά αναπαριστώνται ως πίνακες ή λίστες που περιέχουν δεδομένα σχετικά με κάθε κορυφή ή ακμή. Αυτά τα βάρη μπορεί να αντιπροσωπεύουν, για παράδειγμα, κόστος, μήκη ή χωρητικότητες, ανάλογα με το πρόβλημα που αντιμετωπίζεται. Τέτοιοι γράφοι εμφανίζονται σε πολλά πλαίσια, για παράδειγμα σε προβλήματα συντομότερης διαδρομής όπως το πρόβλημα του πλανόδιου πωλητή.

- **Γειτονιά**: Η γειτονιά μιας κορυφής v σε έναν γράφο είναι το σύνολο όλων των κορυφών που είναι γειτονικές με την v. Τυπικά, για μια κορυφή v, η γειτονιά της $N(v)$ ορίζεται ως $\{u \in V \mid \{u, v\} \in E\}$.
- **Μονοπάτι**: Ένα μονοπάτι σε έναν γράφο είναι μια ακολουθία ακμών που συνδέει μια ακολουθία διακριτών κορυφών. Ένα μονοπάτι είναι απλό αν όλες οι κορυφές (και επομένως όλες οι ακμές) είναι διακριτές.
- **Περίπατος**: Ένας περίπατος είναι μια ακολουθία ακμών και κορυφών όπου επιτρέπεται η επανάληψη κορυφών και ακμών. Ένας περίπατος μπορεί να είναι ανοιχτός, αν αρχίζει και τελειώνει σε διαφορετικές κορυφές, ή κλειστός, αν αρχίζει και τελειώνει στην ίδια κορυφή.
- **Κύκλος**: Ένας κύκλος είναι ένα κλειστό μονοπάτι χωρίς επαναλαμβανόμενες κορυφές ή ακμές, εκτός από την αρχική και τελική κορυφή.
- **Υπογράφος**: Ένας υπογράφος είναι ένας γράφος που σχηματίζεται από ένα υποσύνολο των κορυφών και ακμών ενός μεγαλύτερου γράφου. Αν ένας υπογράφος περιλαμβάνει όλες τις ακμές μεταξύ των κορυφών που εμφανίζονται στον αρχικό γράφο, ονομάζεται επαγόμενος υπογράφος.

1.1.2 Αντιστοίχιση και Ομοιότητα Γράφων

Η αντιστοίχιση γράφων περιλαμβάνει την εύρεση αντιστοιχιών μεταξύ των κορυφών και των ακμών δύο γράφων, αντικατοπτρίζοντας τη δομική τους ομοιότητα. Είναι ιδιαίτερα σημαντική στην αναγνώριση μοτίβων, την όραση υπολογιστών, τη βιοπληροφορική και την ανάλυση κοινωνικών δικτύων. Η αντιστοίχιση γράφων μπορεί να είναι ακριβής (ισομορφισμός γράφων) ή προσεγγιστική.

Ακριβής Αντιστοίχιση Γράφων (Ισομορφισμός Γράφων)

Η ακριβής αντιστοίχιση γράφων, ή ισομορφισμός γράφων, απαιτεί μια ένα-προς-ένα αντιστοίχιση μεταξύ των κορυφών δύο γράφων που διατηρεί τη γειτονικότητα. Τυπικά, δύο γράφοι $G_1 = (V_1, E_1)$ και $G_2 = (V_2, E_2)$ είναι ισομορφικοί εάν υπάρχει μια 1-1 και επί συνάρτηση $f : V_1 \to V_2$ τέτοια ώστε $(u, v) \in E_1$ αν και μόνο αν $(f(u), f(v)) \in E_2$. Ένα παράδειγμα απεικονίζεται στο Σχήμα 1.1.2.

Το πρόβλημα του ισομορφισμού υπογράφου, το οποίο αποτελεί γενίκευση του προβλήματος ισομορφισμού γράφων, είναι γνωστό ότι είναι NP-πλήρες. Το **Τεστ Weisfeiler-Lehman (WL)** και ο **Αλγόριθμος**

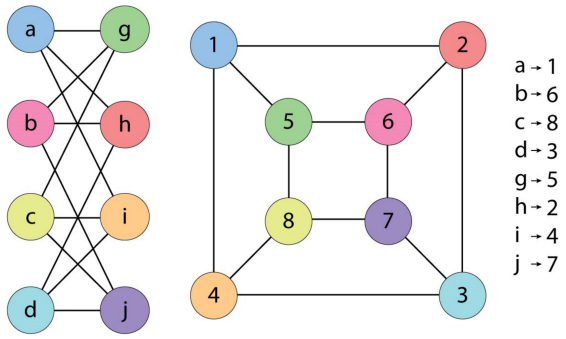


Figure 1.1.2: Μια οπτική απεικόνιση δύο ισομορφικών γράφων.

VF2 είναι αξιοσημείωτες προσεγγίσεις για την ανίχνευση ισομορφισμών, με τον τελευταίο να είναι αποδοτικός στην πράξη παρά την εκθετική του πολυπλοκότητα στην χειρότερη περίπτωση.

Απόσταση Επεξεργασίας Γράφων

Η Απόσταση Επεξεργασίας Γράφων (GED) [59] μετρά την ομοιότητα μεταξύ δύο γράφων ως τον ελάχιστο αριθμό λειτουργιών επεξεργασίας που απαιτούνται για να μετατραπεί ένας αρχικός γράφος σε έναν τελικό. Αυτές οι λειτουργίες περιλαμβάνουν προσθήκες, διαγραφές και αντικαταστάσεις κόμβων και ακμών. Τυπικά, η GED μεταξύ δύο γράφων $G_1 = (V_1, E_1)$ και $G_2 = (V_2, E_2)$ είναι:

$$d(G_1, G_2) = \min_{\gamma \in \Gamma} \sum_{e \in \gamma} c(e)$$

όπου Γ είναι το σύνολο όλων των δυνατών ακολουθιών λειτουργιών επεξεργασίας, και $c(e)$ είναι το κόστος της λειτουργίας επεξεργασίας e. Το Σχήμα 1.1.3 απεικονίζει αυτήν την έννοια.

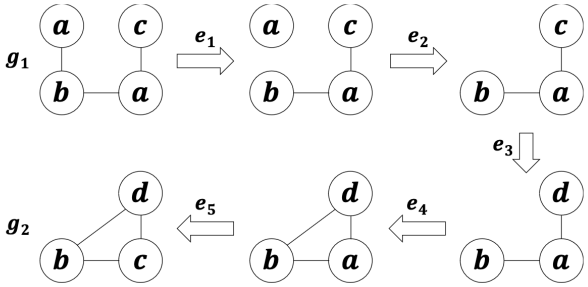


Figure 1.1.3: Απόσταση Επεξεργασίας Γράφων μεταξύ ενός ζεύγους γράφων.

Ο υπολογισμός της GED είναι NP-hard, οδηγώντας στην ανάπτυξη ευρετικών και προσεγγιστικών αλγορίθμων.

Πυρήνες Γράφων

Οι πυρήνες γράφων είναι μια άλλη προσέγγιση για τη μέτρηση της ομοιότητας γράφων, ιδιαίτερα χρήσιμη στη μηχανική μάθηση. Αντιστοιχίζουν τους γράφους σε χώρους χαρακτηριστικών υψηλής διάστασης, επιτρέποντας τη χρήση αλγορίθμων όπως τα Support Vector Machines (SVMs). Οι δημοφιλείς πυρήνες γράφων περιλαμβάνουν:

- **Πυρήνες Τυχαίων Περιπάτων** [28]: Μετρούν την ομοιότητα βάσει του αριθμού των αντίστοιχων τυχαίων περιπάτων σε δύο γράφους.
- **Πυρήνες Συντομότερου Μονοπατιού** [7]: Συγκρίνουν τις κατανομές των συντομότερων μονοπατιών σε δύο γράφους.

- **Πυρήνες Weisfeiler-Lehman (WL)** [61]: Χρησιμοποιούν το τεστ ισομορφισμού Weisfeiler-Lehman για να καταγράψουν δομικές πληροφορίες.
- **Πυρήνες Υπογράφων** [41]: Συγκρίνουν τη συχνότητα διαφόρων μοτίβων υπογράφων μέσα στους γράφους.

Μέγιστος Κοινός Υπογράφος

Το πρόβλημα του Μέγιστου Κοινού Υπογράφου (MCS) περιλαμβάνει την εύρεση του μεγαλύτερου υπογράφου, κοινού σε δύο δεδομένους γράφους. Έχει εφαρμογές στη χημειοπληροφορική, τη βιοπληροφορική και την αναγνώριση μοτίβων.

- **Μέγιστος Κοινός Επαγόμενος Υπογράφος (MCIS)**: Το πρόβλημα MCIS επιδιώκει τον μεγαλύτερο επαγόμενο υπογράφο, κοινό και στους δύο αρχικούς γράφους. Τυπικά, δεδομένων δύο γράφων $G = (V_G, E_G)$ και $H = (V_H, E_H)$, ο στόχος είναι να βρεθεί ένας γράφος $I = (V_I, E_I)$ τέτοιος ώστε I να είναι ένα επαγόμενος υπογράφος τόσο του G όσο και του H και να έχει τον μέγιστο αριθμό κορυφών. Το πρόβλημα αναζήτησης του MCIS είναι NP-hard, και αλγόριθμοι όπως ο αλγόριθμος McSplit [51] χρησιμοποιούνται για την επίλυσή του.
- **Μέγιστος Κοινός Υπογράφος Ακμής (MCES)**: Το πρόβλημα MCES επικεντρώνεται στη μεγιστοποίηση του αριθμού των κοινών ακμών μεταξύ δύο γράφων, ανεξαρτήτως του αριθμού των κορυφών. Είναι χρήσιμο σε σενάρια όπου οι σχέσεις ακμών είναι πιο σημαντικές από τις κορυφές. Όπως και για το MCIS, το πρόβλημα αναζήτησης του MCES είναι NP-hard.

1.1.3 Νευρωνικά Δίκτυα Γράφων

Όπως έχει ήδη αναφερθεί, οι γράφοι είναι διαδεδομένοι σε πολυάριθμους τομείς, συμπεριλαμβανομένων των κοινωνικών δικτύων, των βιολογικών δικτύων, των γράφων γνώσης και των συστημάτων συστάσεων. Τα παραδοσιακά νευρωνικά δίκτυα δεν είναι εγγενώς σχεδιασμένα για τη διαχείριση δεδομένων δομημένων σε γράφους, λόγω της απαίτησής τους για είσοδο με σταθερό μέγεθος. Αυτοί οι περιορισμοί οδήγησαν στην ανάπτυξη των Νευρωνικών Δικτύων Γράφων (ΝΔΓ), τα οποία είναι σχεδιασμένα να επεξεργάζονται άμεσα τις δομές γράφων.

Η κύρια λειτουργία που χρησιμοποιούν τα ΝΔΓ για να επεξεργαστούν και να αναλύσουν δεδομένα σε μορφή γράφων είναι η Συνέλιξη Γράφων (Graph Convolution). Όπως υποδηλώνει και το όνομά της, είναι η αντίστοιχη λειτουργία της συνέλιξης σημάτων (όπως αυτή που εφαρμόζεται στα Συνελικτικά Νευρωνικά Δίκτυα για εικόνες), αλλά με τις απαραίτητες διαφοροποιήσεις, ώστε να μπορεί να εφαρμοστεί σε γράφους. Αυτή η λειτουργία επιτρέπει στα ΝΔΓ να μαθαίνουν και να εξαγάγουν χαρακτηριστικά από κόμβους και τις τοπικές τους γειτονιές, εκμεταλλευόμενα τις σχέσεις που υπάρχουν στον γράφο.

Όπως και με την κλασσική συνέλιξη, η Συνέλιξη Γράφων μπορεί να μελετηθεί τόσο στο χωρικό πεδίο, όσο και στο φασματικό πεδίο. Στο φασματικό πεδίο, η ανάλυση ενός ΝΔΓ συμπεριλαμβάνει τον ορισμό της συνάρτησης που εφαρμόζει, αφού γίνει ο κατάλληλος μετασχηματισμός Fourier (για την συχνοτική μελέτη). Δεδομένου του Μετασχηματισμού Fourier γράφου [54], [19], [6], ενός σήματος γράφου x και ενός φίλτρου $g \in \mathbb{R}^n$, η συνελικτική λειτουργία μπορεί να εκφραστεί ως:

$$x * g = U\left((U^T g) \odot (U^T x)\right)$$

Συμβολίζοντας το φίλτρο με όρους ιδιοτιμών, $g_\theta(\Lambda) = \text{diag}(U^T g)$, η συνελικτική διαδικασία μπορεί να απλοποιηθεί σε:

$$x * g_\theta = U g_\theta U^T x$$

όπου, $g_\theta = \text{diag}(\theta)$ είναι ο διαγώνιος πίνακας που αντιστοιχεί στους συντελεστές του φασματικού φίλτρου.

Για την ανάλυση στο χωρικό πεδίο, τα ΝΔΓ εκμεταλλεύονται τη μέθοδο Μετάβασης Μηνυμάτων [29], δηλαδή διαδίδουν την πληροφορία που έχει ο κάθε κόμβος, σε όλους τους γείτονές του. Η μαθηματική έκφραση μίας τέτοιας συνάρτησης είναι:

$$m_i^{(t+1)} = \sum_{j \in \mathcal{N}(i)} M_t(h_i^{(t)}, h_j^{(t)}, e_{ij})$$

$$h_i^{(t+1)} = U_t(h_i^{(t)}, m_i^{(t+1)})$$

όπου $h_i^{(t)}$ είναι η αναπαράσταση του κόμβου i στο βήμα t, e_{ij} είναι το διάνυσμα της ακμής μεταξύ των κόμβων i και j, και οι M_t και U_t είναι οι μαθήσιμες συναρτήσεις.

Τα περισσότερα χωρικά ΝΔΓ, μπορούν να γραφούν ως μια παραλλαγή της παραπάνω μαθηματικής έκφρασης. Τα πιο διαδεδομένα από αυτά είναι τα ακόλουθα:

- Το **Graph Convolution Network (GCN)** [38] απλοποιεί τη συνέλιξη γράφων στο φασματικό επίπεδο, προσεγγίζοντας τη συνελικτική λειτουργία με μια τοπική προσέγγιση πρώτης τάξης. Ο κανόνας διάδοσης για το GCN δίνεται από τη σχέση:

  $$H^{(l+1)} = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right)$$

  όπου $\tilde{A} = A + I$ είναι ο πίνακας γειτνίασης με προστιθέμενες αυτο-συνδέσεις, $\tilde{D}$ είναι ο πίνακας βαθμού του $\tilde{A}$, $H^{(l)}$ είναι ο πίνακας χαρακτηριστικών κόμβων στο επίπεδο l, $W^{(l)}$ είναι ένας μαθεύσιμος πίνακας βαρών στο επίπεδο l, και σ είναι μια μη γραμμική συνάρτηση ενεργοποίησης.

- Το ΝΔΓ **Graph Attention Network (GAT)**, προτάθηκε το 2017 [65], και η κύρια ιδέα ήταν η χρήση του μηχανισμού προσοχής, και συγκεκριμένα του *multi-head attention* [64]. Ο μηχανισμός προσοχής χρησιμοποιείται για να δημιουργήσει τα βάρη που αποδίδει ο κάθε κόμβος στους γείτονές του. Η ενημερωμένη κατάσταση του κόμβου i προκύπτει τώρα από την ακόλουθη εξίσωση:

  $$h_i^{(l+1)} = \sigma\left(\sum_{j\in\mathcal{N}(i)} \alpha_{ij} W h_j^{(l)}\right)$$

  όπου σ είναι μια μη γραμμικότητα και a_{ij} είναι τα βάρη προσοχής, τα οποία υπολογίζονται ως εξής:

  $$\alpha_{ij} = \frac{\exp(\text{LeakyReLU}(a^T[Wh_i \| Wh_j]))}{\sum_{k\in\mathcal{N}(i)} \exp(\text{LeakyReLU}(a^T[Wh_i \| Wh_k]))}$$

  όπου a και W είναι εκπαιδεύσιμες παράμετροι.

- Το **GATv2** που προτάθηκε στο [8] απέδειξε ότι το αρχικό μοντέλο GAT [65] υπολογίζει στατική προσοχή μεταξύ των κόμβων του γράφου. Το πρόβλημα που παρουσιάζει το μοντέλο GAT είναι ότι η συνάρτηση προσοχής ορίζει μια σταθερή κατάταξη των κόμβων, χωρίς αυτή να εξαρτάται από τον κόμβο i του ερωτήματος κάθε φορά. Στην πράξη, αυτό σημαίνει ότι υπάρχει ένας κόμβος v στον γράφο, στον οποίο όλοι οι υπόλοιποι κόμβοι αποδίδουν το υψηλότερο σκορ προσοχής και αυτό αποδεικνύεται αναλυτικά στο [8]. Η τροποποιημένη εκδοχή για τον υπολογισμό των βαρών προσοχής που λύνει το πρόβλημα αυτό είναι η εξής:

  $$\alpha_{ij} = \frac{\exp(a^T\text{LeakyReLU}([Wh_i \| Wh_j]))}{\sum_{k\in\mathcal{N}(i)} \exp(a^T\text{LeakyReLU}([Wh_i \| Wh_k]))}$$

- Τέλος, το **Graph Isomorphism Network (GIN)** [66] είναι η πρώτη χωρική προσέγγιση που αντιμετωπίζει την αδυναμία προηγούμενων χωρικών μοντέλων να κάνουν διάκριση μεταξύ διαφορετικών δομών γράφων με βάση τις ενσωματώσεις που παράγονται. Για να γίνει αυτό, το GIN χρησιμοποιεί μια απλή τεχνική, προσθέτοντας μια παράμετρο βάρους για τον κεντρικό κόμβο της συνέλιξης. Η λειτουργία ορίζεται παρακάτω, όπου ϵ είναι το βάρος:

  $$h_i^{(l+1)} = \text{MLP}\left((1+\epsilon)\cdot h_i^{(l)} + \sum_{j\in\mathcal{N}(i)} h_j^{(l)}\right)$$

  Το GIN αποδεικνύεται ότι είναι εξίσου ισχυρό με το τεστ ισομορφισμού γράφων Weisfeiler-Lehman, δηλαδή παράγει διαφορετικές ενσωματώσεις κόμβων όταν ασχολούμαστε με μη ισομορφικούς γράφους.

Δίκτυα Αντιστοίχισης Γράφων

Τα Δίκτυα Αντιστοίχισης Γράφων (ΔΑΓ), που προτάθηκαν από τους Li et al. [46], αντιπροσωπεύουν μια εξειδικευμένη προσέγγιση στα ΝΔΓ, σχεδιασμένη για την αντιμετώπιση του προβλήματος της μάθησης ομοιότητας γράφων. Σε αντίθεση με τα παραδοσιακά ΝΔΓ, που αντιστοιχίζουν κάθε γράφο ανεξάρτητα σε έναν διανυσματικό χώρο, τα ΔΑΓ αξιοποιούν έναν μηχανισμό προσοχής μεταξύ γράφων. Αυτό καθιστά τα ΔΑΓ ιδιαίτερα αποτελεσματικά για εργασίες όπου η σχέση μεταξύ των δομών των γράφων παίζει καθοριστικό ρόλο στον προσδιορισμό της ομοιότητας. Τα ΔΑΓ είναι το κύριο μοντέλο ΝΔΓ που θα χρησιμοποιήσουμε στα πειράματά μας στα επόμενα κεφάλαια.

Αρχιτεκτονική Δικτύων Αντιστοίχισης Γράφων: Τα ΔΑΓ ενισχύουν το βασικό μοντέλο των παραδοσιακών ΝΔΓ, ενσωματώνοντας έναν μηχανισμό προσοχής μεταξύ γράφων, που αντιστοιχίζει τους κόμβους ενός γράφου με τους κόμβους του άλλου γράφου.

1. **Προσοχή μεταξύ Γράφων**: Για κάθε κόμβο σε καθέναν από τους δύο γράφους, ο μηχανισμός προσοχής μεταξύ γράφων υπολογίζει έναν συντελεστή προσοχής με κάθε κόμβο στον άλλο γράφο.

$$\alpha_{j\to i} = \frac{\exp(\text{sim}(h_i^{(t)}, h_j^{(t)}))}{\sum_{j'} \exp(\text{sim}(h_i^{(t)}, h_{j'}^{(t)}))}$$

   όπου sim είναι μια συνάρτηση ομοιότητας, όπως η ομοιότητα συνημιτόνου, και $h_i^{(t)}, h_j^{(t)}$ είναι οι αναπαραστάσεις των κόμβων i και j στο επίπεδο t.

2. **Διάνυσμα Αντιστοίχισης**: Το διάνυσμα αντιστοίχισης $\mu_{j\to i}$ αντικατοπτρίζει σε ποιον βαθμό ταιριάζει ένας κόμβος στον ένα γράφο με έναν ή περισσότερους κόμβους στον άλλο γράφο.

$$\mu_{j\to i} = a_{j\to i} \cdot (h_i^{(t)} - h_j^{(t)})$$

3. **Ενημέρωση Διανύσματος Κόμβων**: Τα διανύσματα των κόμβων ενημερώνονται λαμβάνοντας υπόψη τόσο τα συγκεντρωμένα μηνύματα από τη γειτονιά του κόμβου, όσο και το διάνυσμα αντιστοίχισης:

$$h_i^{(t+1)} = f_{\text{node}}\left(h_i^{(t)}, \sum_j m_{j\to i}, \sum_{j'} \mu_{j'\to i}\right)$$

4. **Συσσωρευτής:** Μετά από έναν συγκεκριμένο αριθμό T επιπέδων διάδοσης, ένας συσσωρευτής λαμβάνει το σύνολο των αναπαραστάσεων κόμβων $\{\mathbf{h}_i^{(T)}\}$ ως είσοδο, και υπολογίζει μια αναπαράσταση σε επίπεδο γράφου $\mathbf{h}_G = f_G(\{\mathbf{h}_i^{(T)}\})$. Χρησιμοποιείται ο ακόλουθος συσσωρευτής [45]:

$$\mathbf{h}_G = \text{MLP}_G\left(\sum_{i\in V} \sigma(\text{MLP}_{\text{gate}}(\mathbf{h}_i^{(T)})) \odot \text{MLP}(\mathbf{h}_i^{(T)})\right),$$

   ο οποίος μετασχηματίζει τις αναπαραστάσεις κόμβων και στη συνέχεια χρησιμοποιεί ένα σταθμισμένο άθροισμα με διανύσματα πύλης για να συγκεντρώσει τις πληροφορίες από τους κόμβους. Το σταθμισμένο άθροισμα μπορεί να βοηθήσει στο φιλτράρισμα των άσχετων πληροφοριών, είναι πιο ισχυρό από ένα απλό άθροισμα και λειτουργεί επίσης σημαντικά καλύτερα εμπειρικά.

5. **Αναπαράσταση και Υπολογισμός Ομοιότητας Γράφων**: Οι αναπαραστάσεις σε επίπεδο γράφου h_{G_1} και h_{G_2}, χρησιμοποιούνται για να υπολογιστεί η μετρική ομοιότητας.

$$s(G_1, G_2) = f_s(h_{G_1}, h_{G_2})$$

   όπου f_s είναι μια συνάρτηση ομοιότητας.

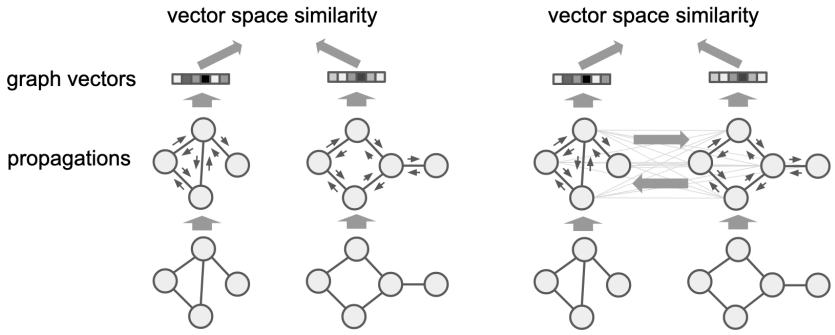


Figure 1.1.4: Απεικόνιση ενός Νευρωνικού Δικτύου Γράφων (αριστερά) και ενός Δικτύου Αντιστοίχισης Γράφων (δεξιά) [46].

Εκπαίδευση Δικτύων Αντιστοίχισης Γράφων: Η εκπαίδευση των ΔΑΓ περιλαμβάνει τη βελτιστοποίηση μιας συνάρτησης απώλειας που ενθαρρύνει παρόμοιους γράφους να έχουν υψηλές βαθμολογίες ομοιότητας και ανόμοιους γράφους να έχουν χαμηλές βαθμολογίες ομοιότητας. Δύο κοινές συναρτήσεις απώλειας είναι:

1. **Απώλεια Ζευγών**: Δεδομένων ζευγών γράφων με ετικέτα $t = 1$ αν είναι όμοιοι ή $t = -1$ αν είναι ανόμοιοι:

$$L_{\text{pair}} = \mathbb{E}_{(G_1,G_2,t)} \left[\max\{0, \gamma - t(1 - d(G_1, G_2))\}\right]$$

   όπου $d(G_1, G_2)$ είναι η απόσταση μεταξύ των διανυσμάτων γράφων και γ είναι μια παράμετρος περιθωρίου.

2. **Απώλεια Τριάδων**: Δεδομένων τριών γράφων (G_1, G_2, G_3) όπου ο G_1 είναι πιο όμοιος με τον G_2 από ό,τι με τον G_3:

$$L_{\text{triplet}} = \mathbb{E}_{(G_1,G_2,G_3)} \left[\max\{0, d(G_1, G_2) - d(G_1, G_3) + \gamma\}\right]$$

1.1.4 Σύνοψη Γράφων

Η σύνοψη γράφων αναφέρεται στη διαδικασία δημιουργίας μιας συνοπτικής αναπαράστασης ενός μεγάλου γράφου, διατηρώντας παράλληλα βασικές δομικές ιδιότητες και πληροφορίες. Έχει ευρείες εφαρμογές, όπως η ομαδοποίηση, η ταξινόμηση και η ανίχνευση κοινοτήτων σε τομείς όπως η ανάλυση κοινωνικών δικτύων, τα βιολογικά δίκτυα, οι γράφοι γνώσης, τα συστήματα συστάσεων και η βιοπληροφορική.

Οι αλγόριθμοι σύνοψης γράφων συνήθως παράγουν είτε συνοπτικούς γράφους με τη μορφή υπερ-γράφων ή αραιωμένων γράφων, είτε μια λίστα ανεξάρτητων δομών. Οι ευρύτερα χρησιμοποιούμενες μέθοδοι είναι οι ακόλουθες:

- **Graph Pooling:** Το graph pooling είναι μια τεχνική που χρησιμοποιείται κυρίως στο πλαίσιο αρχιτεκτονικών ΝΔΓ με στόχο τη δημιουργία μιας συνοπτικότερης αναπαράστασης ενός γράφου, ομαδοποιώντας ή επιλέγοντας κόμβους ή χαρακτηριστικά. Το graph pooling συγκεντρώνει πληροφορίες από ένα σύνολο κόμβων σε μια μοναδική αναπαράσταση, δημιουργώντας μια ιεραρχία προοδευτικά μικρότερων γράφων.

- **Graph Clustering:** Το graph clustering στοχεύει στη διαίρεση των κόμβων ενός γράφου σε ομάδες, έτσι ώστε οι κόμβοι εντός της ίδιας ομάδας να είναι περισσότερο όμοιοι μεταξύ τους από ό,τι με αυτούς σε άλλες ομάδες. Η ομοιότητα μπορεί να βασίζεται σε διάφορα κριτήρια, όπως η συνδεσιμότητα, τα κοινά χαρακτηριστικά ή οι κοινοί ρόλοι εντός του γράφου.

- **Graph Prototyping:** Το graph prototyping στοχεύει στον εντοπισμό αντιπροσωπευτικών υπογράφων, που ονομάζονται πρωτότυπα, οι οποία αποτυπώνουν τα βασικά χαρακτηριστικά ενός μεγάλου γράφου.

Το πεδίο της σύνοψης γράφων έχει μελετηθεί εκτενώς, με πολλές προσεγγίσεις να προτείνονται για συγκέντρωση, ομαδοποίηση και δημιουργία πρωτοτύπων.

Παραδοσιακές Μέθοδοι

Οι παραδοσιακές μέθοδοι έχουν προσφέρει διάφορες τεχνικές για την απλοποίηση και ερμηνεία των δομών γράφων. Ορισμένες παραδοσιακές μέθοδοι για την ομαδοποίηση γράφων περιλαμβάνουν:

- **Ομαδοποίηση Με Βάση Το Modularity**: Αλγόριθμοι όπως η μέθοδος Louvain [5] μεγιστοποιούν το modularity, ένα μέτρο της πυκνότητας των συνδέσεων εντός των ομάδων σε σύγκριση με τις συνδέσεις μεταξύ των ομάδων.
- **Φασματική Ομαδοποίηση**: Αυτή η τεχνική [49] χρησιμοποιεί τις ιδιοτιμές του Λαπλασιανού πίνακα του γράφου για τη μείωση των διαστάσεων πριν την εφαρμογή παραδοσιακών μεθόδων ομαδοποίησης, όπως το k-means.
- **Ιεραρχική Ομαδοποίηση**: Στην ιεραρχική ομαδοποίηση [69], οι κόμβοι ομαδοποιούνται ή διαχωρίζονται διαδοχικά με βάση τη συνδεσιμότητά τους, σχηματίζοντας ένα δέντρο ομάδων (δενδρόγραμμα).
- **Ομαδοποίηση Με Βάση Την Πυκνότητα**: Μέθοδοι όπως η DBSCAN [24] εντοπίζουν ομάδες με βάση την πυκνότητα των κόμβων στον γράφο.

Μέθοδοι Βασισμένες Σε ΝΔΓ

Οι πρόσφατες εξελίξεις στη βαθιά μάθηση έχουν οδηγήσει στην ανάπτυξη μεθόδων βασισμένων σε ΝΔΓ για σύνοψη γράφων [60], [48]. Αυτές οι τεχνικές αξιοποιούν την ικανότητα των ΝΔΓ να συλλαμβάνουν σύνθετα μοτίβα και σχέσεις εντός των γράφων.

Graph Pooling και Clustering

Ορισμένες από τις ευρύτερα χρησιμοποιούμενες μεθόδους για Graph Pooling και Clustering είναι οι ακόλουθες:

- **MinCutPool:** Η μέθοδος MinCutPool [4] διατυπώνει μια γενίκευση του κανονικοποιημένου προβλήματος της ελάχιστης τομής και εκπαιδεύει ένα ΝΔΓ για να διαχωρίσει τους κόμβους σε ομάδες. Η συνάρτηση απώλειας της MinCutPool είναι:

$$\mathcal{L}_{MC} = -\frac{\mathrm{Tr}\left(S^T \tilde{A} S\right)}{\mathrm{Tr}\left(S^T \tilde{D} S\right)} + \left\| \frac{S^T S}{\|S^T S\|_F} - \frac{I_K}{\sqrt{K}} \right\|_F,$$

  όπου $\|\cdot\|$ είναι η νόρμα Frobenius, $\tilde{A} = D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$ και $\tilde{D}$ είναι ο βαθμός πίνακα του $\tilde{A}$.

- **DMoN:** Η μέθοδος DMoN [63] βελτιστοποιεί μια συνάρτηση απώλειας βασισμένη στο modularity:

$$\mathcal{L}_{DMoN} = -\frac{\mathrm{Tr}\left(S^T \tilde{A} S\right)}{2E} + \frac{\sqrt{K}}{N} \left\| \sum_i {S_i}^T \right\|_F - 1,$$

  όπου $\|\cdot\|$ είναι η νόρμα Frobenius, $\tilde{A} = A - D^T D$ και D είναι ο degree matrix του A.

- **JustBalance:** Η μέθοδος JustBalance [3] προτείνει μια απλοποίηση στον υπολογισμό της συνάρτησης απώλειας σε σύγκριση με το MinCutPool:

$$\mathcal{L}_{JB} = -\mathrm{Tr}(\sqrt{S^T S})$$

- **TopK:** Η μέθοδος TopK [39], [12], [26] επιλέγει τους σημαντικότερους κόμβους ενός γράφου βάσει των χαρακτηριστικών τους. Αυτή η μέθοδος μειώνει το μέγεθος του γράφου διατηρώντας σημαντικές δομικές πληροφορίες, διατηρώντας κόμβους με τις υψηλότερες βαθμολογίες με βάση έναν πίνακα προβολής.

Graph Prototyping

Πρόσφατες εργασίες έχουν εξερευνήσει διάφορες μεθοδολογίες για την επίτευξη αποτελεσματικής δημιουργίας πρωτοτύπων εντός του πλαισίου των ΝΔΓ. Σε αυτό το πλαίσιο, σημαντικές συνεισφορές έχουν γίνει χρησιμοποιώντας τεχνικές βασισμένες σε γράφους για Εξηγήσεις με Αντιπαραδείγματα (Counterfactual Explanations

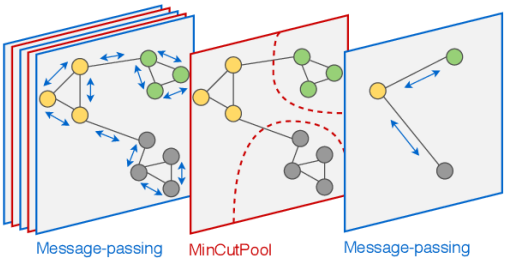


Figure 1.1.5: Μια αρχιτεκτονική ΝΔΓ όπου τα επίπεδα μεταφοράς μηνυμάτων ακολουθούνται από ένα επίπεδο MinCutPool [4].

- CE). Οι Dimitriou et al. (2024) [18] προτείνουν μια προσέγγιση που χρησιμοποιεί σημασιολογικούς γράφους για να αναπαραστήσει εικόνες και αξιοποιεί τα ΝΔΓ για αποτελεσματικό υπολογισμό της Απόστασης Επεξεργασίας Γράφων (GED) για την ανάκτηση αντιπαραδειγμάτων μέσω ελαχίστων επεξεργασιών γράφων. Ομοίως, οι Dimitriou και Chaidos et al. (2024) [17] πραγματοποιούν μια συγκριτική μελέτη διαφόρων αλγορίθμων μηχανικής μάθησης για γράφους για να καθορίσουν την πιο αποτελεσματική προσέγγιση για τη δημιουργία εξηγήσεων με αντιπαραδείγματα μέσω επεξεργασιών γράφων.

Επιπλέον, έχουν προταθεί διάφορες μέθοδοι που αξιοποιούν τις δυνατότητες των ΝΔΓ για τη δημιουργία αντιπροσωπευτικών υπογράφων που συλλαμβάνουν τα βασικά μοτίβα των δεδομένων, βελτιώνοντας την ερμηνευσιμότητά τους. Σημαντικές προσεγγίσεις περιλαμβάνουν:

- **ProtGNN:** Το ProtGNN [70] χρησιμοποιεί πρωτότυπα σε έναν διανυσματικό χώρο προκειμένου να κάνει τα ΝΔΓ πιο ερμηνεύσιμα.
- **PxGNN:** Το PxGNN [14] επικεντρώνεται στη μάθηση πρωτοτύπων που αποτυπώνουν αντιπροσωπευτικά χαρακτηριστικά κάθε κλάσης.
- **PAGE:** Το PAGE [62] ανακαλύπτει ερμηνεύσιμα από τον άνθρωπο πρωτότυπα για την εξήγηση της συμπεριφοράς των ΝΔΓ.
- **CPCA:** Η μέθοδος CPCA [55] κατασκευάζει πρωτότυπα κλάσεων και χρησιμοποιεί ενίσχυση πρωτοτύπων για τη δημιουργία εικονικών κλάσεων.

1.2 Προτεινόμενες Προσεγγίσεις

1.2.1 Συνεισφορά

Οι κύριες συνεισφορές αυτής της διατριβής συνοψίζονται παρακάτω:

- Χρησιμοποιούμε τα Δίκτυα Αντιστοίχισης Γράφων (ΔΑΓ) για την αντιμετώπιση του προβλήματος της σύνοψης γράφων, παρέχοντας μία λεπτομερή επισκόπηση του προβλήματος και των μεθοδολογιών που εφαρμόζουμε για την επίλυσή του.
- Εκπαιδεύουμε το ΔΑΓ σε ένα πρόβλημα ομοιότητας γράφων για να αξιοποιήσουμε την ικανότητά του να μαθαίνει σημαντικά μοτίβα και χαρακτηριστικά, τα οποία στη συνέχεια χρησιμοποιούνται για την εξαγωγή σημαντικών υπογράφων.
- Προτείνουμε και αξιολογούμε δύο μεθόδους εξαγωγής συνόψεων γράφων από τα διανύσματα των γράφων έπειτα από πέρασμα από το μοντέλο. Η πρώτη μέθοδος περιλαμβάνει τη δημιουργία υποψηφίων συνόψεων χρησιμοποιώντας TopK pooling και την επιλογή της καλύτερης. Η δεύτερη μέθοδος επεκτείνει έναν υπάρχοντα αλγόριθμο Μέγιστου Κοινού Υπογράφου (MCS) με την επιβολή πρόσθετων περιορισμών που βασίζονται στην προσοχή μεταξύ των γράφων.
- Δημιουργούμε ένα σύνολο δεδομένων με Γεωμετρικά Σχήματα και τυχαίο θόρυβο, όπου είναι γνωστό το ground truth, για να αξιολογήσουμε τις προτεινόμενες μεθόδους.

1.2.2 Ορισμός Προβλήματος

Το πρόβλημα που επιδιώκουμε να αντιμετωπίσουμε είναι η αναγνώριση υπογράφων που είναι αντιπροσωπευτικά για μία κλάση σε ένα σύνολο δεδομένων πολλαπλών κλάσεων, όπου κάθε γράφος ανήκει σε μια συγκεκριμένη κλάση. Ο κύριος στόχος είναι η εξαγωγή ενός υπογράφου από κάθε γράφο, που να αντιπροσωπεύει αποτελεσματικά την κλάση στην οποία ανήκει ο γράφος.

Τυπικά, δεδομένου ενός συνόλου γράφων $G = \{G_1, G_2, \ldots, G_N\}$, όπου κάθε γράφος G_i είναι επισημασμένος με μια κλάση $c_i \in C$ (με C να είναι το σύνολο όλων των κλάσεων), ο στόχος είναι να αναπτύξουμε μεθόδους για να εξαγάγουμε έναν υπογράφο $S_i \subseteq G_i$ για κάθε γράφο G_i. Ο εξαγόμενος υπογράφος S_i θα πρέπει να είναι ιδιαίτερα αντιπροσωπευτικό της κλάσης c_i.

1.2.3 Προτεινόμενο Μοντέλο

Το ΝΔΓ που χρησιμοποιούμε βασίζεται στην αρχιτεκτονική του Δικτύου Αντιστοίχισης Γράφων [46] και αποτελείται από πολλαπλά επίπεδα **Graph Matching Convolution**, τα οποία περιλαμβάνουν έναν μηχανισμό προσοχής μεταξύ των γράφων, ακολουθούμενα από έναν **Graph Aggregator**.

Εκπαίδευση

Σύμφωνα με την προσέγγιση που περιγράψαμε προηγουμένως, το ΔΑΓ εκπαιδεύεται σε ένα πρόβλημα ομοιότητας γράφων. Το μοντέλο λαμβάνει ζεύγη γράφων με θετικές (ίδια κλάση) ή αρνητικές (διαφορετική κλάση) ετικέτες. Ο στόχος είναι να προβλέψει υψηλή ομοιότητα για τα θετικά ζεύγη και χαμηλή για τα αρνητικά ζεύγη, ελαχιστοποιώντας την παρακάτω συνάρτηση απώλειας:

$$L_{\text{pair}} = \mathbb{E}_{(G_1,G_2,t)} \left[\max\{0, \gamma - t(1 - d(G_1, G_2))\}\right]$$

όπου $d(G_1, G_2)$ είναι η απόσταση μεταξύ των γράφων στην είσοδο, γ είναι μια παράμετρος περιθωρίου, και t είναι 1 για θετικά ζεύγη και -1 για αρνητικά ζεύγη.

Πρόβλεψη

Κατά την πρόβλεψη, το μοντέλο λαμβάνει ένα ζεύγος γράφων που μπορεί να ανήκουν στην ίδια ή σε διαφορετικές κλάσεις. Το ζεύγος περνά μέσα από το μοντέλο και μετά από κάθε επίπεδο, τα ενημερωμένα διανύσματα των κόμβων και η προσοχή μεταξύ των γράφων υπολογίζονται και χρησιμοποιούνται για τη δημιουργία των συνόψεων γράφων. Οι δύο προτεινόμενες μέθοδοι για τη δημιουργία συνόψεων γράφων περιγράφονται ακολούθως.

Μέθοδος I: Στην πρώτη μέθοδο, επιδιώκουμε να εξαγάγουμε μια σύνοψη γράφου αντιπροσωπευτική της κλάσης από ένα ζεύγος γράφων στην είσοδο, χρησιμοποιώντας τα διανύσματα κόμβων που υπολογίζονται σε κάθε επίπεδο κατά τη διάρκεια του forward pass.

1. **Forward Pass:** Το ζεύγος γράφων (G_1, G_2) περνά μέσα από ένα μοντέλο με L επίπεδα. Μετά από κάθε επίπεδο, εξάγονται τα διανύσματα κόμβων.
2. **Πρόβλεψη k:** Ο αριθμός των σημαντικών κόμβων k προβλέπεται χρησιμοποιώντας τα διανύσματα κόμβων X_1^l και X_2^l. Για κάθε επίπεδο l:
   - Για κάθε κόμβο στους γράφους G_1 και G_2, υπολογίζονται βαθμολογίες, με βάση τη σημασία τους, χρησιμοποιώντας έναν διάνυσμα προβολής w, που είναι ένα διάνυσμα βαρών που αρχικοποιείται με βάση μια ομοιόμορφη κατανομή. Το διάνυσμα προβολής w αρχικοποιείται αντλώντας τιμές από μια ομοιόμορφη κατανομή εντός του εύρους $[-\frac{1}{\sqrt{d}}, \frac{1}{\sqrt{d}}]$, όπου d είναι η διάσταση των διανυσμάτων κόμβων. Οι βαθμολογίες σημασίας δίνονται από:

$$s_{1,i}^l = \frac{X_{1,i}^l \cdot w}{\|w\|}, \quad s_{2,i}^l = \frac{X_{2,i}^l \cdot w}{\|w\|}$$

   - Αυτές οι βαθμολογίες κανονικοποιούνται στη συνέχεια χρησιμοποιώντας τη συνάρτηση softmax:

$$\hat{s}_{1,i}^l = \frac{e^{s_{1,i}^l}}{\sum_{j=1}^{N_1} e^{s_{1,j}^l}}, \quad \hat{s}_{2,i}^l = \frac{e^{s_{2,i}^l}}{\sum_{j=1}^{N_2} e^{s_{2,j}^l}}$$

- Οι κόμβοι ταξινομούνται στη συνέχεια βάσει των κανονικοποιημένων βαθμολογιών με φθίνουσα σειρά. Έστω $\hat{s}_{1,i}^{l,\text{sorted}}$ και $\hat{s}_{2,i}^{l,\text{sorted}}$ οι ταξινομημένες κανονικοποιημένες βαθμολογίες για G_1 και G_2 αντίστοιχα.
- Το σωρευτικό άθροισμα των ταξινομημένων και κανονικοποιημένων βαθμολογιών υπολογίζεται για τον προσδιορισμό του k ως τον μικρότερο αριθμό κόμβων που απαιτούνται για να φτάσουν ένα κατώφλι τ:

$$k_1^l = \min\left\{ n \,\middle|\, \sum_{i=1}^{n} \hat{s}_{1,i}^{l,\text{sorted}} \geq \tau \right\}$$

$$k_2^l = \min\left\{ n \,\middle|\, \sum_{i=1}^{n} \hat{s}_{2,i}^{l,\text{sorted}} \geq \tau \right\}$$

Το κατώφλι τ είναι μια υπερπαράμετρος του μοντέλου.

Μετά την απόκτηση τιμών k για κάθε επίπεδο και για τους δύο γράφους, το τελικό k υπολογίζεται ως ο μέσος όρος όλων των προβλεπόμενων τιμών k:

$$k = \left\lceil \frac{1}{2L} \sum_{l=1}^{L} (k_1^l + k_2^l) \right\rceil$$

3. **Δημιουργία Υπογράφων:** Για κάθε στρώμα l, δημιουργούνται οι υπογράφοι S_1^l και S_2^l για τους γράφους G_1 και G_2 χρησιμοποιώντας ένα στρώμα TopK pooling με το k που προβλέφθηκε:

$$S_1^l = \text{TopK}(X_1^l, k),$$
$$S_2^l = \text{TopK}(X_2^l, k)$$

Αυτοί οι υπογράφοι λειτουργούν ως υποψήφιοι υπογράφοι από τους οποίους θα παραχθεί η τελική σύνοψη.

4. **Έλεγχος Ισομορφισμού:** Σε κάθε επίπεδο l, οι υπογράφοι S_1^l και S_2^l ελέγχονται για ισομορφισμό. Βάσει εμπειρικών παρατηρήσεων, εάν οι υπογράφοι σε ένα επίπεδο είναι ισομορφικοί, συνήθως ταυτίζονται με το ground truth. Επομένως, εάν $S_1^l \simeq S_2^l$, ο υπογράφος S^l επιστρέφεται. Εάν δεν βρεθούν ισομορφικοί υπογράφοι στο ίδιο επίπεδο, εξετάζουμε όλα τα επίπεδα, προσπαθώντας να βρούμε οποιουσδήποτε υπογράφους από τον γράφο 1 που είναι ισομορφικοί με οποιονδήποτε υπογράφο από τον γράφο 2: $\exists l_1, l_2$ όπως $S_1^{l_1} \simeq S_2^{l_2}$. Σε αυτήν την περίπτωση, επιστρέφεται ο πιο συχνά εμφανιζόμενος ισομορφικός υπογράφος.

5. **Επιλογή Τελικής Σύνοψης:** Εάν δεν βρεθεί ζεύγος ισομορφικών υπογράφων σε όλα τα επίπεδα, επιστρέφεται ο πιο συχνά εμφανιζόμενος ισομορφικός υπογράφος στη λίστα των υποψήφιων συνόψεων.

$$S^* = \arg \max_{S \in \{P^1, P^2, \ldots, P^L\}} \text{count}(S).$$

Μέθοδος II: Στη δεύτερη μέθοδο, χρησιμοποιούμε την προσοχή μεταξύ των γράφων που υπολογίζονται από το μοντέλο μετά από κάθε επίπεδο για να κατασκευάσουμε τις συνόψεις γράφων.

Αλγόριθμος MCS: Ο αλγόριθμος MCS που υλοποιήσαμε είναι μια επέκταση του αλγορίθμου McSplit [51], σχεδιασμένος να βρει τον μέγιστο κοινό επαγόμενο υπογράφο με επιπλέον περιορισμούς. Αυτή η επέκταση λαμβάνει υπόψη όχι μόνο τον αριθμό των κόμβων αλλά και τον αριθμό των ακμών, δίνοντας προτεραιότητα στους πιο πυκνούς υπογράφους όταν βρεθούν πολλαπλοί υπογράφοι του ίδιου μεγέθους, καθώς αυτή η προσέγγιση αποτυπώνει καλύτερα τη δομική σημασία σε ορισμένες εφαρμογές [56], [22], [13]. Η διαδικασία περιλαμβάνει δύο κύρια βήματα: τον προσδιορισμό ζευγών αντιστοίχισης μεταξύ των δύο γράφων και την εκτέλεση του αλγόριθμου χρησιμοποιώντας αυτά τα ζεύγη ως περιορισμούς.

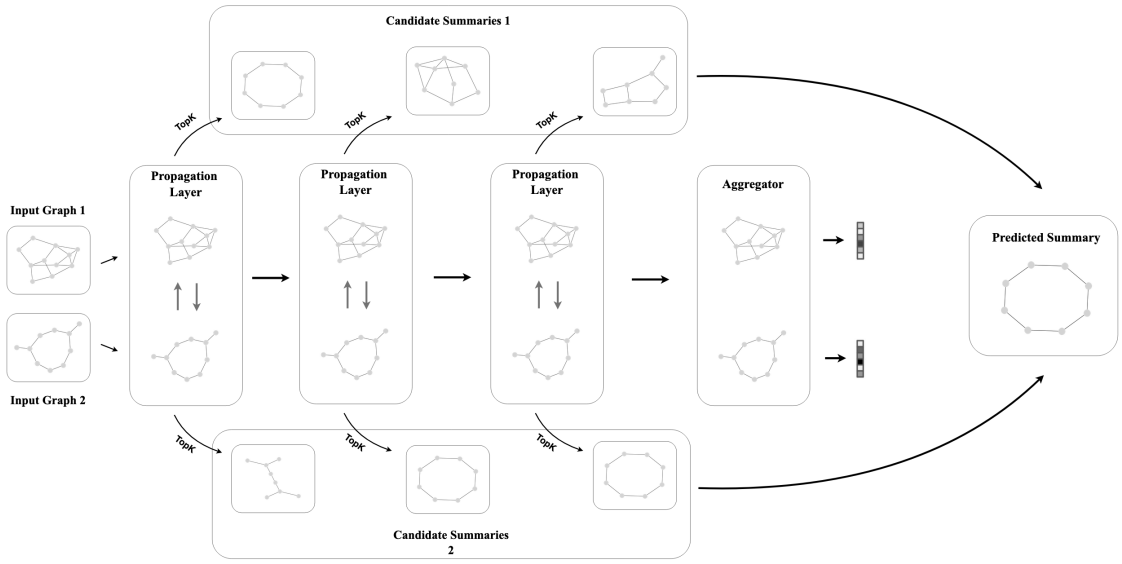


Figure 1.2.1: Επισκόπηση της Μεθόδου Ι.

- **Προσδιορισμός Ζευγών Αντιστοίχισης:** Ένα ζεύγος κόμβων, ένας από κάθε γράφο, θεωρείται ζεύγος αντιστοίχισης εάν οι κόμβοι παρουσιάζουν υψηλές βαθμολογίες προσοχής ο ένας προς τον άλλον και στις δύο κατευθύνσεις. Αυτά τα ζεύγη αντιστοίχισης σχηματίζουν το σύνολο MP.

  Τυπικά, έστω α_{ij} η βαθμολογία προσοχής από τον κόμβο $i \in V_1$ στον κόμβο $j \in V_2$. Τα ζεύγη αντιστοίχισης προσδιορίζονται ως εξής:

  – Για κάθε κόμβο $v_i \in V_1$, προσδιορίζονται οι κόμβοι στο V_2 προς τους οποίους δίνει προσοχή με βαθμολογία πάνω από ένα κατώφλι θ.

  – Για κάθε κόμβο $v_j \in V_2$, προσδιορίζονται οι κόμβοι στο V_1 προς τους οποίους δίνει προσοχή με βαθμολογία πάνω από ένα κατώφλι θ.

  – Ένα ζεύγος (v_i, v_j) θεωρείται ταιριαστό ζεύγος εάν $\alpha_{ij} > \theta$ και $\alpha_{ji} > \theta$.

- **Αλγόριθμος MCS:** Χρησιμοποιώντας το σύνολο των ζευγών αντιστοίχισης MP από όλα τα επίπεδα ως περιορισμούς, ο αλγόριθμος MCS βρίσκει τον μέγιστο κοινό επαγόμενο υπογράφο μεταξύ δύο γράφων G_1 και G_2. Ο αλγόριθμος παρουσιάζεται παρακάτω:

Algorithm Μέγιστος Κοινός Υπογράφος με περιορισμούς

Procedure Search(future, M, E, mp)
begin
if $|M| >$ |best_mapping| **or** ($|M| ==$ |best_mapping| **and** $E >$best_edge_count) **then**
  best_mapping $\leftarrow$ M
  best_edge_count $\leftarrow$ E
end if
bound $\leftarrow |M| + \sum_{\langle G,H\rangle \in \text{future}} \min(|G|, |H|)$
if bound $\leq$ |best_mapping| **then**
  return
end if
$\langle G, H\rangle \leftarrow$ SelectLabelClass(future)
$v \leftarrow$ SelectVertex(G)
for $w \in H$ **do**
  if $(v, w) \notin mp$ **then**
    continue

```
    end if
    future' ← ∅
    for ⟨G′, H′⟩ ∈ future do
      G″ ← G′ ∩ N(G, v) \ {v}
      H″ ← H′ ∩ N(H, w) \ {w}
      if G″ ≠ ∅ and H″ ≠ ∅ then
        future' ← future' ∪{⟨G″, H″⟩}
      end if
      G″ ← G′ ∩ N̄(G, v) \ {v}
      H″ ← H′ ∩ N̄(H, w) \ {w}
      if G″ ≠ ∅ and H″ ≠ ∅ then
        future' ← future' ∪{⟨G″, H″⟩}
      end if
    end for
    Search(future', M ∪{(v, w)}, E + new_edges(M, (v, w)), mp)
  end for
  G′ ← G \ {v}
  future ← future \{⟨G, H⟩}
  if G′ ≠ ∅ then
    future ← future ∪{⟨G′, H⟩}
  end if
  Search(future, M, E, mp)
  end

  Procedure McSplit(G, H)
  begin
  best_mapping ← ∅
  best_edge_count ← 0
  Search({⟨V(G), V(H)⟩}, ∅, 0, mp)
  return best_mapping
  end
```

Για να αποφύγουμε την εξερεύνηση υπογράφων που έχουν ήδη εξερευνηθεί, ορίζεται μια κανονική μορφή για κάθε αντιστοίχιση. Αυτή η κανονική μορφή δημιουργείται με την αναδιάταξη των κόμβων του υπογράφου που προκαλείται από την αντιστοίχιση M με συνεπή τρόπο, έτσι ώστε ισομορφικοί υπογράφοι να έχουν την ίδια αναπαράσταση.

Εξαγωγή Βασικού Υπογράφου: Για να βελτιώσουμε περαιτέρω την αποτελεσματικότητα της μεθόδου, πειραματιζόμαστε με μια προσέγγιση για να παράγουμε μια τελική σύνοψη αναγνωρίζοντας τον μέγιστο βασικό υπογράφο από μια ομάδα συνόψεων. Ο μέγιστος βασικός υπογράφος ορίζεται ως ο γράφος από την ομάδα συνόψεων που εμφανίζεται πιο συχνά ως υπογράφος των άλλων συνόψεων. Σε περίπτωση ισοπαλιών, επιλέγεται ο μεγαλύτερος γράφος μεταξύ των υπογράφων που είναι ισόπαλοι. Τυπικά:

$$S^* = \arg \max_{S \in \{S^1, S^2, \ldots, S^n\}} \left(\sum_{i=1}^{n} \mathbb{I}(S \subseteq S^i) \right)$$

όπου $\mathbb{I}$ είναι η συνάρτηση δείκτη που επιστρέφει 1 αν S είναι υπογράφος του S^i και 0 διαφορετικά.

Εάν υπάρχουν πολλαπλοί γράφοι με τον ίδιο μέγιστο αριθμό υπογράφων, ο μέγιστος βασικός υπογράφος S^{**} ορίζεται ως:

$$S^{**} = \arg \max_{S \in \{S | \sum_{i=1}^{n} \mathbb{I}(S \subseteq S^i) = \sum_{i=1}^{n} \mathbb{I}(S^* \subseteq S^i)\}} |V(S)|$$

όπου $|V(S)|$ συμβολίζει τον αριθμό κορυφών στον γράφο S.

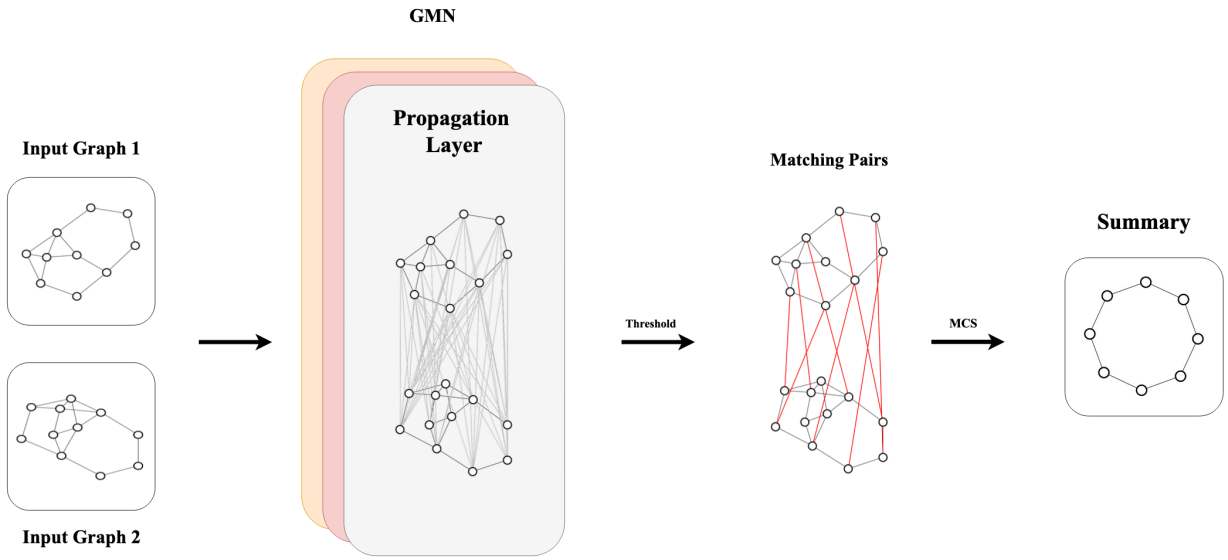


Figure 1.2.2: Επισκόπηση της Μεθόδου ΙΙ.

1.3 Πειραματικό μέρος

1.3.1 Σύνολα Δεδομένων

Τα δύο σύνολα δεδομένων που θα χρησιμοποιήσουμε στα πειράματά μας είναι τα MUTAG [15] και Geometric Shapes.

Geometric Shapes

Για να προσεγγίσουμε καλύτερα τη συγκεκριμένη φύση του προβλήματός μας, δημιουργήσαμε ένα συνθετικό σύνολο δεδομένων με Γεωμετρικά Σχήματα. Αυτό το σύνολο δεδομένων αποτελείται από γράφους που αναπαριστούν τέσσερα διαφορετικά γεωμετρικά σχήματα: κύκλους, γραμμές, αστέρια και πλήρεις γράφους. Για να προσθέσουμε ποικιλομορφία, προσθέτουμε θόρυβο με τη μορφή τυχαίων κόμβων και ακμών σε κάθε γράφο. Αυτές οι ακμές μπορούν να συνδέουν δύο κόμβους των βασικών σχημάτων, δύο κόμβους θορύβου ή έναν κόμβο από κάθε κλάση. Πειραματιστήκαμε με δύο προσεγγίσεις για τα χαρακτηριστικά των κόμβων. Στην πρώτη προσέγγιση, όλοι οι κόμβοι έχουν χαρακτηριστικά που αποτελούνται από 1. Στη δεύτερη προσέγγιση, χρησιμοποιούμε το Node2Vec [31] για να δημιουργήσουμε χαρακτηριστικά κόμβων. Η πρώτη από τις δύο προσεγγίσεις είχε καλύτερα αποτελέσματα, συνεπώς ακολουθήσαμε αυτή στα πειράματά μας.

Αυτό μας παρέχει ένα ελεγχόμενο περιβάλλον όπου γνωρίζουμε το ground truth κάθε γράφου, και μας επιτρέπει να αξιολογήσουμε με μεγαλύτερη ακρίβεια την αποτελεσματικότητα του προτεινόμενου μοντέλου και των μεθόδων μας. Δημιουργήσαμε τρεις εκδόσεις του συνόλου δεδομένων Geometric Shapes, με βασικά γεωμετρικά σχήματα που αποτελούνται από 8, 15 και 25 κόμβους, αντίστοιχα. Κάθε έκδοση περιέχει 360 γράφους, με 90 γράφους ανά κλάση. Για το σύνολο δεδομένων με σχήματα 8 κόμβων, προσθέσαμε 2-4 κόμβους θορύβου και 1-3 ακμές θορύβου ανά προστιθέμενο κόμβο. Καθώς το μέγεθος των βασικών σχημάτων αυξάνεται σε 15 και 25 κόμβους, αυξήσαμε τον αριθμό των προστιθέμενων κόμβων και ακμών θορύβου για να διατηρήσουμε μια σχετικά σταθερή αναλογία θορύβου σε σχέση με το βασικό σχήμα.

MUTAG

Το MUTAG [15] είναι μια συλλογή νιτροαρωματικών ενώσεων και ο στόχος είναι να προβλεφθεί η μεταλλαξιογόνος δράση τους στη Salmonella typhimurium. Οι γράφοι χρησιμοποιούνται για να αναπαραστήσουν

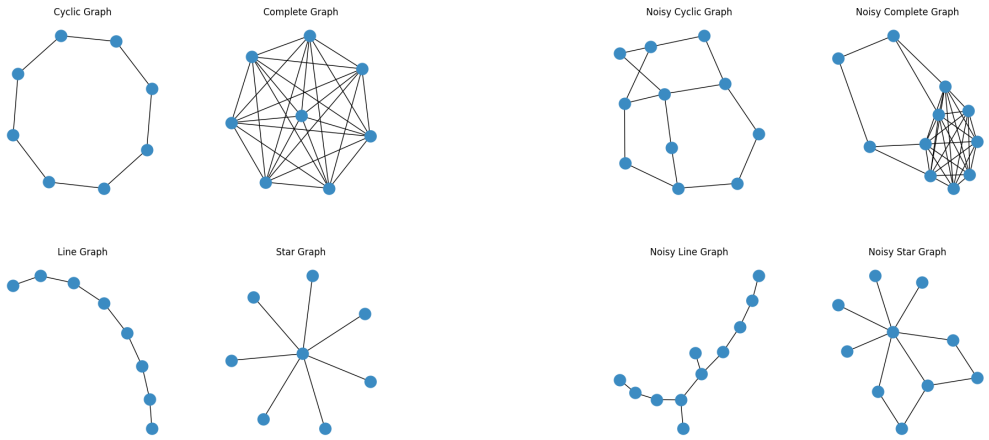


Figure 1.3.1: Τα τέσσερα βασικά γεωμετρικά σχήματα στο συνόλου δεδομένων Geometric Shapes και παραδείγματα γράφων με θόρυβο.

χημικές ενώσεις, όπου οι κορυφές αντιπροσωπεύουν άτομα και είναι επισημασμένες με τον τύπο του ατόμου (που προκύπτει με one-hot encoding), ενώ οι ακμές μεταξύ των κορυφών αναπαριστούν δεσμούς μεταξύ των αντίστοιχων ατόμων. Περιλαμβάνει 188 δείγματα χημικών ενώσεων με 7 διακριτές ετικέτες κόμβων. Κάθε γράφος επισημαίνεται ως είτε μεταλλαξιογόνος (θετική κλάση) είτε μη μεταλλαξιογόνος (αρνητική κλάση), υποδεικνύοντας εάν η ένωση έχει μεταλλαξιογόνο επίδραση σε έναν συγκεκριμένο οργανισμό.

Το dataset MUTAG δεν περιλαμβάνει ground truth υπογράφους που αντιπροσωπεύουν την κλάση, αλλά παρέχει ground truth πρωτότυπα, καθιστώντας το ιδιαίτερα κατάλληλο για πειράματα με τη Μέθοδο ΙΙ.

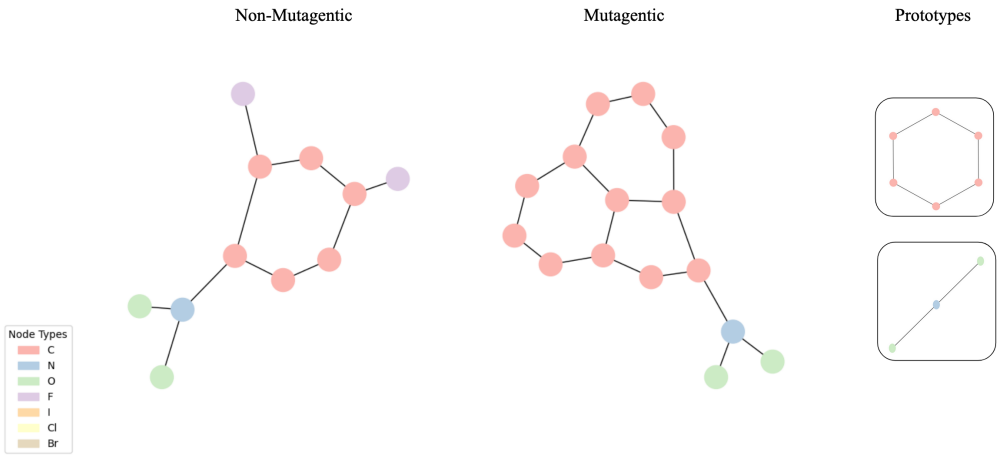


Figure 1.3.2: Παραδείγματα γράφων από κάθε μία από τις δύο κλάσεις στο σύνολο δεδομένων MUTAG (μη μεταλλαξιογόνος και μεταλλαξιογόνος), μαζί με τα πρωτότυπα του συνόλου δεδομένων.

1.3.2 Μετρικές Αξιολόγησης

Ακρίβεια

Για την αξιολόγηση της αποτελεσματικότητας των προτεινόμενων μεθόδων, χρησιμοποιούμε μία μετρική ακρίβειας προσαρμοσμένη στο πρόβλημά μας. Για κάθε κλάση στο σύνολο δεδομένων, παρέχεται ένας ground truth υπογράφος. Η ακρίβεια του μοντέλου καθορίζεται με τη σύγκριση των κόμβων που επιλέγονται από το μοντέλο από τον αρχικό γράφο με τους κόμβους στον ground truth υπογράφο. Εάν οι επιλεγμένοι κόμβοι ταιριάζουν με τον ground truth υπογράφο, η ακρίβεια για αυτό τον γράφο θεωρείται ότι είναι 1· διαφορετικά, είναι 0. Η

συνολική ακρίβεια υπολογίζεται στη συνέχεια ως ο μέσος όρος αυτών των τιμών σε όλους τους γράφους του συνόλου αξιολόγησης. Θεωρούμε δύο τύπους ακρίβειας, Ακριβή και Προσεγγιστική. Η Ακριβής αναφέρεται στις περιπτώσεις όπου η σύνοψη που παράγεται από το μοντέλο είναι το σωστό σχήμα και έχει τον σωστό αριθμό κόμβων, ταιριάζοντας ακριβώς με το ground truth, ενώ η Προσεγγιστική αναφέρεται στις περιπτώσεις όπου η σύνοψη είναι το σωστό σχήμα, αλλά επιτρέπεται μια μικρή απόκλιση στον αριθμό των κόμβων (± 1 κόμβοι σε σχέση με το ground truth).

Για να συγκρίνουμε το μοντέλο μας με άλλες μεθόδους σύνοψης γράφων που παρουσιάστηκαν προηγουμένως, οι οποίες παράγουν συνόψεις αναθέτοντας ετικέτες στους κόμβους και ομαδοποιώντας κόμβους με την ίδια ετικέτα, προσθέσαμε ετικέτες στους κόμβους των γράφων στο σύνολο δεδομένων. Συγκεκριμένα, σε κάθε κόμβο στο βασικό γεωμετρικό σχήμα του γράφου ανατίθεται μια μοναδική ετικέτα. Σε κάθε κόμβο θορύβου ανατίθενται ετικέτες που προκύπτουν από την ένωση των ετικετών των κόμβων με τους οποίους συνδέεται. Με αυτόν τον τρόπο, το μοντέλο πρέπει μόνο να προβλέψει για έναν κόμβο θορύβου την ίδια ετικέτα με έναν από τους κόμβους του βασικού σχήματος στον οποίο έχει μια διαδρομή μέσω άλλων κόμβων θορύβου. Με βάση αυτές τις ετικέτες, υπολογίζουμε την ακρίβεια του μοντέλου. Αυτή η μετρική ακρίβειας είναι μια χαλάρωση της παραδοσιακής ακρίβειας που χρησιμοποιείται στην ταξινόμηση κόμβων, καθώς ένας κόμβος θεωρείται σωστά ταξινομημένος εάν το μοντέλο προβλέψει σωστά οποιαδήποτε από τις ετικέτες που του έχουν ανατεθεί.

1.3.3 Βασικά Μοντέλα

Για να αξιολογήσουμε την απόδοση του προτεινόμενου μοντέλου ΔΑΓ, το συγκρίνουμε με ορισμένες καθιερωμένες μεθόδους σύνοψης γράφων, τις οποίες έχουμε αναλύσει προηγουμένως. Συγκεκριμένα, εφαρμόσαμε τις αρχιτεκτονικές ΝΔΓ που προτάθηκαν στις αντίστοιχες εργασίες που εισήγαγαν τα επίπεδα pooling MinCut [4], DMoN [63], και JustBalance [3] για την ομαδοποίηση γράφων, κάνοντας μικρές τροποποιήσεις για να τις προσαρμόσουμε στο σύνολο δεδομένων μας.

Επιπλέον, συγκρίνουμε το μοντέλο μας με ένα παραδοσιακό μοντέλο GAT [65]. Δεδομένου ότι και τα δύο μοντέλα χρησιμοποιούν μηχανισμούς προσοχής, αυτή η σύγκριση μας επιτρέπει να αξιολογήσουμε εάν ο μηχανισμός αντιστοίχισης με βάση την προσοχή μεταξύ γράφων που προτάθηκε από τους Li et al. [46] παρέχει καλύτερα αποτελέσματα σε σύγκριση με τον μηχανισμό αυτο-προσοχής που χρησιμοποιείται στο GAT. Δεδομένου ότι το μοντέλο GAT επεξεργάζεται έναν μόνο γράφο ως είσοδο και δεν μπορεί να εκπαιδευτεί σε ένα πρόβλημα ομοιότητας γράφων, το εκπαιδεύσαμε σε ένα πρόβλημα ταξινόμησης γράφων και αξιολογήσαμε την απόδοσή του στην σύνοψη γράφων χρησιμοποιώντας την προτεινόμενη προσέγγισή μας που περιγράφεται στην Μέθοδο Ι με μικρές τροποποιήσεις για να προσαρμοστεί από ζεύγη γράφων σε μεμονωμένους γράφους.

1.3.4 Λεπτομέρειες Μοντέλων

Δίκτυα Αντιστοίχισης Γράφων

Εκπαίδευση: Εκπαιδεύουμε το ΔΑΓ [46] σε ένα πρόβλημα ομοιότητας γράφων. Για τα δύο σύνολα δεδομένων, σχηματίζουμε παρτίδες από τυχαία επιλεγμένα ζεύγη και τις αντίστοιχες ετικέτες τους, οι οποίες είναι 1 αν οι δύο γράφοι ανήκουν στην ίδια κλάση ή -1 διαφορετικά.

Οι υπερπαράμετροι που χρειάστηκαν ρύθμιση περιλαμβάνουν την παράμετρο περιθωρίου γ, τη διάσταση των κρυφών επιπέδων του μοντέλου, τον αριθμό των επιπέδων, τον ρυθμό μάθησης, το μέγεθος της παρτίδας και τον συνολικό αριθμό ζευγών. Για τη βελτιστοποίηση αυτών των υπερπαραμέτρων, λάβαμε υπόψη την απόδοση του μοντέλου τόσο στο πρόβλημα εκπαίδευσης ομοιότητας γράφων όσο και στο πρόβλημα εξαγωγής συνόψεων.

Για την παράμετρο περιθωρίου γ, το μοντέλο προσαρμόστηκε καλά σε διάφορες τιμές κατά τη διάρκεια της εκπαίδευσης αλλά απέδωσε καλύτερα στο πρόβλημα εξαγωγής συνόψεων με χαμηλότερη τιμή, συγκεκριμένα 0.2. Η διάσταση των κρυφών επιπέδων του μοντέλου ορίστηκε σε υψηλότερη τιμή, συγκεκριμένα 32, από τη διάσταση των χαρακτηριστικών των κόμβων του συνόλου δεδομένων για να επιτρέψει μεγαλύτερη εκφραστικότητα. Δεδομένου του μικρού μεγέθους του συνόλου δεδομένων, επιλέξαμε έναν υψηλότερο ρυθμό μάθησης (0.01), και μικρότερες τιμές για το μέγεθος της παρτίδας (64). Επιπλέον, επιλέξαμε μικρότερες τιμές για τον αριθμό των ζευγών, συγκεκριμένα 400, στο σύνολο εκπαίδευσης για να αποφύγουμε την υπερπροσαρμογή. Παρατηρήσαμε, επίσης, ότι οι αρχιτεκτονικές μοντέλων με περισσότερα επίπεδα (5-7) απέδωσαν καλύτερα στο πρόβλημα εξαγωγής συνόψεων, και επιλέξαμε ως τελική τιμή το 7. Τέλος, χρησιμοποιήσαμε τον βελτιστοποιητή Adam [37], θέτοντας την τιμή της αποσύνθεσης των βαρών σε 10^{-5}.

Πρόβλεψη: Κατά την πρόβλεψη, ένα ζεύγος γράφων παρέχεται ως είσοδος στο μοντέλο. Καθώς το ζεύγος μεταβαίνει από ένα επίπεδο στο επόμενο, τα διανύσματα και οι προσοχές μεταξύ γράφων υπολογίζονται μετά από κάθε επίπεδο και στη συνέχεια χρησιμοποιούνται για την εξαγωγή των συνόψεων σύμφωνα με τις Μεθόδους Ι και ΙΙ. Οι δύο γράφοι στο ζεύγος ανήκουν στην ίδια κλάση, επιτρέποντάς μας να αξιολογήσουμε την ικανότητα του μοντέλου να αναγνωρίζει υπογράφους που αντιπροσωπεύουν την κλάση.

Για να κατανοήσουμε περαιτέρω τη συμπεριφορά του μοντέλου, εξετάζουμε επίσης τις συνόψεις που παράγονται σε κάθε επίπεδο όταν το ζεύγος εισόδου αποτελείται από γράφους διαφορετικών κλάσεων. Αυτή η ανάλυση παρέχει πληροφορίες για το πώς το μοντέλο ενημερώνει τα διανύσματα των κόμβων όταν η προβλεπόμενη ομοιότητα μεταξύ των γράφων στην είσοδο είναι χαμηλή και εάν αυτές οι ενημερώσεις οδηγούν στη δημιουργία ουσιαστικών συνόψεων σε κάθε επίπεδο. Τα αποτελέσματα αυτών των πειραμάτων, συμπεριλαμβανομένων παραδειγμάτων των συνόψεων που παράγονται για ζεύγη γράφων της ίδιας κλάσης, παρουσιάζονται και συζητούνται ακολούθως.

Βασικά Μοντέλα

Εκπαίδευση Για τις αρχιτεκτονικές που προτάθηκαν στις αντίστοιχες εργασίες που εισήγαγαν τα επίπεδα pooling MinCut [4], DMoN [63] και JustBalance [3], εκπαιδεύσαμε τα μοντέλα χρησιμοποιώντας είτε την απώλεια των επιπέδων pooling μόνη της, όπως προτάθηκε αρχικά, είτε έναν συνδυασμό της απώλειας των επιπέδων pooling και μιας απώλειας ταξινόμησης γράφων. Το μοντέλο GAT [65] εκπαιδεύτηκε σε ένα πρόβλημα ταξινόμησης γράφων. Δεδομένου ότι το μοντέλο GAT επεξεργάζεται έναν μόνο γράφο ως είσοδο και δεν μπορεί να εκπαιδευτεί άμεσα σε ένα πρόβλημα ομοιότητας γράφων, αυτή η προσέγγιση μας επέτρεψε να αξιολογήσουμε την απόδοσή του με συγκρίσιμο τρόπο με το προτεινόμενο ΔΑΓ.

Πρόβλεψη: Κατά την πρόβλεψη, οι αρχιτεκτονικές που χρησιμοποιούν τα επίπεδα pooling MinCut, DMoN και JustBalance έλαβαν έναν γράφο, ο οποίος διέρχεται από το δίκτυο. Στο τέλος της διαδικασίας, η σύνοψη που παράγεται από το επίπεδο pooling λαμβάνεται ως η προβλεπόμενη σύνοψη γράφου. Για το μοντέλο GAT, ένας γράφος διέρχεται από το δίκτυο και, όμοια με το ΔΑΓ, μια σύνοψη δημιουργείται σε κάθε επίπεδο χρησιμοποιώντας έναν μηχανισμό TopK pooling. Η πιο συχνή σύνοψη σε όλα τα επίπεδα επιλέγεται στη συνέχεια, ακολουθώντας μια μεθοδολογία παρόμοια με την Μέθοδο Ι που χρησιμοποιείται για το ΔΑΓ. Αυτή η προσέγγιση μας επέτρεψε να διατηρήσουμε τη συνέπεια στη διαδικασία αξιολόγησης και να εξασφαλίσουμε δίκαιη σύγκριση της απόδοσης των μοντέλων.

1.3.5 Αποτελέσματα

Ποσοτικά Αποτελέσματα

Αρχικά, θα παρουσιάσουμε τις ακριβείς και τις προσεγγιστικές βαθμολογίες ακρίβειας που επιτεύχθηκαν από το ΔΑΓ, χρησιμοποιώντας τις Μεθόδους Ι και ΙΙ, και τα βασικά μοντέλα στο σύνολο δεδομένων Geometric Shapes. Κάθε μοντέλο αντιστοιχεί στο βέλτιστο μοντέλο που επιτεύχθηκε μετά την ρύθμιση των υπερπαραμέτρων. Τα αποτελέσματα παρουσιάζονται στον Πίνακα 1.1.

Model	Geometric Shapes (8)	Geometric Shapes (15)	Geometric Shapes (25)
ΔΑΓ (Method I)	0.622	0.539	0.452
ΔΑΓ (Method II)	**0.695**	**0.687**	**0.641**
GAT	0.488	0.436	0.388
GNN with MinCut	0.194	0.092	0.049
GNN with DMoN	0.203	0.097	0.050
GNN with JustBalance	0.231	0.103	0.052

Table 1.1: Συνολικές βαθμολογίες ακριβούς και προσεγγιστικής ακρίβειας σε όλες τις κλάσεις για όλα τα μοντέλα στο dataset Geometric Shapes.

Η πρώτη παρατήρηση είναι ότι τόσο το ΔΑΓ όσο και το GAT ξεπερνούν σημαντικά τα μοντέλα που βασίζονται στο clustering και στις τρεις εκδόσεις του συνόλου δεδομένων. Αυτό υποδηλώνει ότι η εκπαίδευση των μοντέλων ΝΔΓ βελτιστοποιώντας το εκάστοτε πρόβλημα που μοντελοποιούν τα αντίστοιχα επίπεδα pooling δεν οδηγεί άμεσα σε συνόψεις που να προσεγγίζουν το ground truth.

Συγκρίνοντας τα μοντέλα ΔΑΓ και GAT, το ΔΑΓ με την Μέθοδο ΙΙ επιτυγχάνει σταθερά υψηλότερη ακρίβεια και με τις δύο μεθόδους σε όλες τις τρεις εκδόσεις του συνόλου δεδομένων. Για να αποκτήσουμε μια βαθύτερη κατανόηση της απόδοσης αυτών των μοντέλων, οι μετρικές ακρίβειας για κάθε κλάση παρουσιάζονται στον Πίνακα 1.2 (Προσεγγιστική) και στον Πίνακα 1.3 (Ακριβής).

(a) Geometric Shapes (8)

Model	Cycle	Complete	Line	Star	Total
GMN (Method I)	0.607 (0.806)	**0.977** (0.977)	0.491 (0.751)	0.416 (0.816)	0.622 (0.838)
GMN (Method II)	**0.713**	0.764	**0.763**	**0.541**	**0.695**
GAT	0.573 (0.722)	0.418 (0.567)	0.457 (0.713)	0.502 (0.688)	0.488 (0.672)

(b) Geometric Shapes (15)

Model	Cycle	Complete	Line	Star	Total
GMN (Method I)	0.407 (0.452)	**0.971** (0.971)	0.285 (0.454)	0.493 (0.753)	0.539 (0.658)
GMN (Method II)	**0.682**	0.764	**0.753**	**0.516**	**0.687**
GAT	0.483 (0.612)	0.344 (0.578)	0.465 (0.632)	0.453 (0.657)	0.436 (0.620)

(c) Geometric Shapes (25)

Model	Cycle	Complete	Line	Star	Total
GMN (Method I)	0.235 (0.274)	**0.948** (0.948)	0.220 (0.293)	**0.477** (0.613)	0.452 (0.532)
GMN (Method II)	**0.662**	0.738	**0.727**	0.438	**0.641**
GAT	0.451 (0.607)	0.268 (0.564)	0.406 (0.491)	0.425 (0.694)	0.388 (0.587)

Table 1.2: Προσεγγιστικές βαθμολογίες ακρίβειας ανά κλάση για ΔΑΓ και GAT στο σύνολο δεδομένων Geometric Shapes. Οι τιμές σε παρενθέσεις αντιπροσωπεύουν το ποσοστό των ζευγών όπου το ground truth εμφανίστηκε τουλάχιστον μία φορά στους υπογράφους που δημιουργήθηκαν μετά από κάθε επίπεδο.

Συνολικά, τα αποτελέσματα δείχνουν ότι τα διαφορετικά μοντέλα και οι μέθοδοι παρουσιάζουν διαφορετικά επίπεδα απόδοσης ανάλογα με τις κλάσεις και τα σύνολα δεδομένων. Από τα τρία, το ΔΑΓ με τη Μέθοδο ΙΙ ήταν το πιο συνεπές σε όλα τα σύνολα δεδομένων, επιτυγχάνοντας τις υψηλότερες βαθμολογίες ακριβούς ακρίβειας σε όλες τις κλάσεις εκτός από την κλάση "αστέρι", όπου το μοντέλο GAT είχε λίγο καλύτερη επίδοση. Επιπλέον, οι βαθμολογίες ακριβούς και προσεγγιστικής ακρίβειας για κάθε κλάση είναι παρόμοιες, το οποίο οφείλεται στο γεγονός ότι όταν οι προβλεπόμενες συνόψεις δεν είναι απόλυτα σωστές, ιδίως για τα αστέρια, περιλαμβάνουν συχνά κόμβους θορύβου που ταιριάζουν μεταξύ των δύο γράφων, όπως θα δείξουμε με παραδείγματα στην επόμενη ενότητα. Συνεπώς, όταν η προβλεπόμενη σύνοψη δεν έχει τον ακριβή αριθμό κόμβων, συνήθως αποτυγχάνει να διατηρήσει και το σωστό σχήμα.

Η Μέθοδος Ι, από την άλλη πλευρά, αποδίδει καλά στην προσεγγιστική ακρίβεια, ιδίως για πλήρεις γράφους, αλλά έχει χειρότερα αποτελέσματα στην ακριβή ακρίβεια. Παρόλο που συχνά προβλέπει τον σωστό αριθμό κόμβων (k), η ακριβής ακρίβεια της δεν είναι τόσο υψηλή όσο αναμενόταν. Ωστόσο, η Μέθοδος Ι εξακολουθεί να αποδίδει καλά όταν ο προβλεπόμενος αριθμός κόμβων είναι εντός του ± 1 του σωστού αριθμού, υποδεικνύοντας ότι αποτυπώνει αποτελεσματικά τη γενική δομή.

Τέλος, το μοντέλο GAT παρουσιάζει συνεπή απόδοση σε διαφορετικές κλάσεις, αποδίδοντας καλά στην ακριβή ακρίβεια, όταν προβλέπεται σωστά ο αριθμός των κόμβων (k), συχνά υπερβαίνοντας τη Μέθοδο Ι. Ωστόσο, δεν αποδίδει καλά όταν ο αριθμός των κόμβων δεν προβλέπεται σωστά, αποτυγχάνοντας να δημιουργήσει ουσιαστικές συνόψεις, όπως υποδηλώνεται από τις πολύ παρόμοιες βαθμολογίες ακριβούς και προσεγγιστικής ακρίβειας, σε αντίθεση με το ΔΑΓ με τη Μέθοδο Ι, που μπορεί ακόμη να παράξει σχετικά ακριβείς συνόψεις ακόμη και με μικρές αποκλίσεις στο k.

Στη συνέχεια, εξετάζουμε τη συμπεριφορά του μοντέλου ΔΑΓ όταν το ζεύγος εισόδου αποτελείται από γράφους που ανήκουν σε διαφορετικές κλάσεις. Δεδομένου ότι αυτοί οι γράφοι ανήκουν σε διαφορετικές κλάσεις, οι Μέθοδοι Ι και ΙΙ δεν μπορούν να χρησιμοποιηθούν άμεσα για την αξιολόγηση της ακρίβειας του μοντέλου. Συνεπώς, ακολουθούμε μια απλοποιημένη προσέγγιση παρόμοια με τα αρχικά βήματα της Μεθόδου Ι, δημιουργώντας υπογράφους για τους δύο γράφους μετά από κάθε επίπεδο l. Στη συνέχεια, μετράμε τον αριθμό των ζευγών όπου οι παραγόμενοι υπογράφοι ταιριάζουν με το ground truth. Συγκεκριμένα, μετράμε πόσο συχνά εμφανίζε-

(a) Geometric Shapes (8)

Model	Cycle	Complete	Line	Star	Total
GMN (Method I)	0.607 (0.806)	0.760 (0.773)	0.474 (0.713)	0.331 (0.708)	0.543 (0.750)
GMN (Method II)	**0.713**	**0.764**	**0.721**	0.486	**0.671**
GAT	0.573 (0.722)	0.418 (0.567)	0.446 (0.656)	**0.493** (0.615)	0.482 (0.640)

(b) Geometric Shapes (15)

Model	Cycle	Complete	Line	Star	Total
GMN (Method I)	0.407 (0.452)	0.286 (0.295)	0.239 (0.425)	0.279 (0.448)	0.302 (0.405)
GMN (Method II)	**0.682**	**0.764**	**0.746**	0.421	**0.653**
GAT	0.483 (0.612)	0.344 (0.578)	0.451 (0.586)	**0.436** (0.641)	0.429 (0.604)

(c) Geometric Shapes (25)

Model	Cycle	Complete	Line	Star	Total
GMN (Method I)	0.235 (0.274)	0.583 (0.583)	0.207 (0.276)	0.237 (0.335)	0.316 (0.367)
GMN (Method II)	**0.662**	**0.738**	**0.704**	0.396	**0.625**
GAT	0.451 (0.607)	0.268 (0.564)	0.393 (0.477)	**0.418** (0.668)	0.382 (0.579)

Table 1.3: Ακριβείς βαθμολογίες ακρίβειας ανά κλάση για ΔΑΓ και GAT στο σύνολο δεδομένων Geometric Shapes. Οι τιμές σε παρενθέσεις αντιπροσωπεύουν το ποσοστό των ζευγών όπου το ground truth εμφανίστηκε τουλάχιστον μία φορά στους υπογράφους που δημιουργήθηκαν μετά από κάθε επίπεδο.

ται το ground truth για κάθε γράφο, καθώς και πόσο συχνά εμφανίζονται τα ground truths και για τους δύο γράφους. Τα αποτελέσματα παρουσιάζονται στον Πίνακα 1.4

Classes	**Geometric Shapes (8)**			**Geometric Shapes (15)**			**Geometric Shapes (25)**		
	1	2	Both	1	2	Both	1	2	Both
Cycle (1) και Complete (2)	0.62	0.97	0.61	0.51	0.89	0.49	0.29	0.83	0.27
Cycle (1) και Line (2)	0.66	0.46	0.34	0.38	0.32	0.13	0.31	0.27	0.08
Cycle (1) και Star (2)	0.53	0.59	0.33	0.41	0.55	0.27	0.36	0.57	0.27
Complete (1) και Line (2)	0.97	0.41	0.40	0.83	0.35	0.32	0.85	0.33	0.31
Complete (1) και Star (2)	0.96	0.53	0.52	0.86	0.47	0.42	0.80	0.64	0.56
Line (1) και Star (2)	0.47	0.54	0.23	0.32	0.46	0.18	0.24	0.60	0.17

Table 1.4: Αποτελέσματα του ΔΑΓ με ζεύγη γράφων από διαφορετικές κλάσεις στο συνόλου δεδομένων Geometric Shapes. Οι στήλες "1" και "2" δείχνουν το ποσοστό των ζευγών όπου το ground truth για τον πρώτο και τον δεύτερο γράφο, αντίστοιχα, εμφανίστηκε τουλάχιστον μία φορά στους παραγόμενα υπογράφους. Η στήλη "Both" υποδεικνύει το ποσοστό των ζευγών όπου τα ground truth και για τους δύο γράφους ήταν παρόντα.

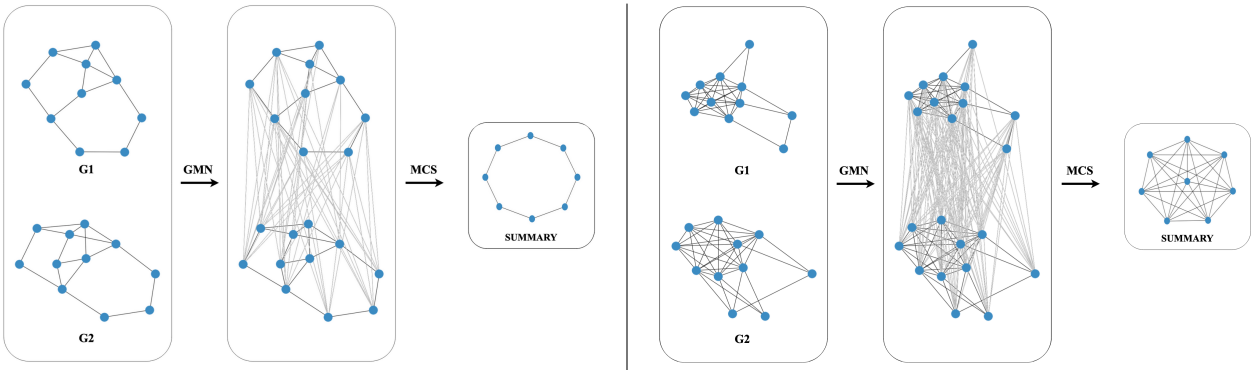
Μια γενική παρατήρηση που προκύπτει από τον Πίνακα 1.4 είναι ότι οι μετρικές ακρίβειας παραμένουν σχετικά σταθερές για κάθε κλάση, ανεξάρτητα από την κλάση του άλλου γράφου στο ζεύγος. Επιπλέον, αυτές οι μετρικές ακρίβειας είναι συγκρίσιμες με εκείνες που παρατηρήθηκαν όταν και οι δύο γράφοι στο ζεύγος ανήκουν στην ίδια κλάση, όπως φαίνεται στον Πίνακα 1.2. Αυτή η συνέπεια υποδεικνύει ότι η ικανότητα του μοντέλου να αναγνωρίζει υπογράφους που αντιπροσωπεύουν την κλάση είναι συνεπής και δεν επηρεάζεται σημαντικά από την παρουσία γράφων από διαφορετική κλάση στο ζεύγος. Το μοντέλο αναγνωρίζει αποτελεσματικά τα σχετικά χαρακτηριστικά υπογράφων που αντιστοιχούν σε κάθε κλάση, ακόμη και παρουσία δυνητικά παραπλανητικών πληροφοριών από διαφορετική κλάση.

Ποιοτικά Αποτελέσματα

Τέλος, για να αξιολογήσουμε ποιοτικά τη Μέθοδο ΙΙ, η οποία έχει την καλύτερη απόδοση στο σύνολο δεδομένων Geometric Shapes, παρουσιάζουμε παραδείγματα από τις παραγόμενες συνόψεις από το αυτό το σύνολο δεδομένων και ορισμένα αποτελέσματα από τα πειράματά μας στο σύνολο δεδομένων MUTAG.

Geometric Shapes

Ξεκινώντας με το σύνολο δεδομένων Geometric Shapes, επισημαίνουμε περιπτώσεις όπου η μέθοδος αποδίδει καλά, αναγνωρίζοντας το ground truth, περιπτώσεις όπου ταιριάζει επίσης κόμβους θορύβου μεταξύ των δύο γράφων, οδηγώντας είτε σε προσεγγιστικές είτε σε λανθασμένες λύσεις, και περιπτώσεις όπου βρίσκει προσεγγιστικές λύσεις αναγνωρίζοντας ένα υποσύνολο του ground truth που διατηρεί το σωστό σχήμα. Παρουσιάζουμε επίσης περιπτώσεις όπου η παραγόμενη σύνοψη είναι λανθασμένη. Για καλύτερη ευκρίνεια, παρουσιάζουμε κυρίως παραδείγματα από το σύνολο δεδομένων με βασικά σχήματα 8 κόμβων, αλλά παρόμοια μοτίβα και παρατηρήσεις βρέθηκαν και στις εκδόσεις του συνόλου δεδομένων με 15 και 25 κόμβους.

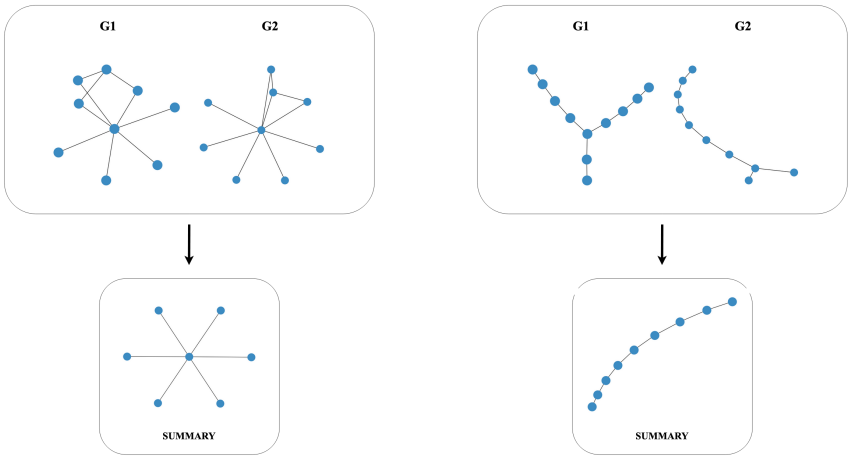


(a) Exact match example for the Cycle class.

(b) Exact match example for the Complete class.

Figure 1.3.3: Παραδείγματα ακριβούς ταιριάσματος για το σύνολο δεδομένων Geometric Shapes (8).

Στο Σχήμα 1.3.3, παρουσιάζουμε δύο παραδείγματα όπου το μοντέλο προβλέπει με ακρίβεια το ground truth. Στο μεσαίο τμήμα, όπου απεικονίζουμε τις προσοχές μεταξύ των γράφων, παρατηρούμε ότι οι κόμβοι θορύβου, που δεν είναι μέρος του ground truth, είτε δεν "προσέχουν" κανέναν κόμβο από τον άλλο γράφο είτε "προσέχουν" κόμβους με τους οποίους δεν ταιριάζουν. Επομένως, αν και ο μέγιστος κοινός υπογράφος των γράφων G_1 και G_2 είναι ένα υπερσύνολο του ground truth, τα ταιριάσματα που προκύπτουν από τη διέλευσή τους μέσω του ΝΔΓ μας επιτρέπουν να προβλέψουμε με ακρίβεια τη σωστή σύνοψη.

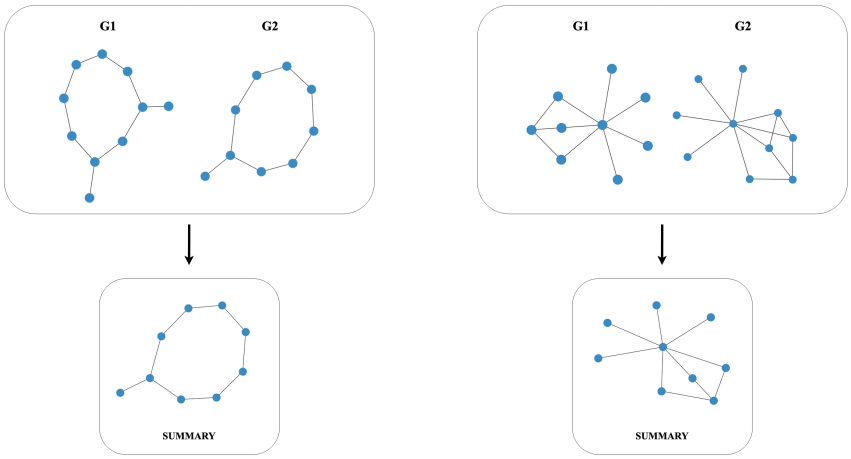


(a) Παράδειγμα προσεγγιστικού ταιριάσματος για αστεροειδή γράφο.

(b) Παράδειγμα προσεγγιστικού ταιριάσματος για γράφο γραμμή.

Figure 1.3.4: Παραδείγματα προσεγγιστικού ταιριάσματος για το σύνολο δεδομένων Geometric Shapes (8).

Στη συνέχεια, στο Σχήμα 1.3.4 παρουσιάζουμε δύο παραδείγματα όπου οι παραγόμενες συνόψεις ταιριάζουν προσεγγιστικά με το ground truth, δηλαδή διατηρούν το σωστό σχήμα αλλά διαφέρουν στον αριθμό κόμβων κατά ± 1. Πιο συγκεκριμένα, στο Σχήμα 1.3.4a, η παραγόμενη αστεροειδής σύνοψη έχει 7 κόμβους, σε σύγκριση με τους 8 του ground truth. Αυτή η διαφορά οφείλεται στο γεγονός ότι οι προσοχές μεταξύ των κόμβων θορύβου



(a) Λανθασμένο παράδειγμα για κυκλικό γράφο.

(b) Λανθασμένο παράδειγμα για αστεροειδή γράφο.

Figure 1.3.5: Λανθασμένα παραδείγματα για το σύνολο δεδομένων Geometric Shapes (8).

των δύο γράφων φιλτραρίστηκαν από το όριο που θέσαμε, αλλά το ίδιο φιλτράρισμα επηρέασε και έναν από τους κόμβους που συνδέονται με αυτούς σε κάθε γράφο, με αποτέλεσμα μια μικρότερη αστεροειδή σύνοψη. Αντίθετα, στο Σχήμα 1.3.4b, η παραγόμενη σύνοψη έχει 9 κόμβους αντί για 8, καθώς, εκτός από τις προσοχές μεταξύ των κόμβων του βασικού σχήματος, οι προσοχές μεταξύ ενός ζεύγους κόμβων θορύβου που επέκτειναν τη γραμμή ήταν επίσης πάνω από το όριο που θέσαμε, οδηγώντας σε μια μεγαλύτερη παραγόμενη σύνοψη.

Τέλος, στο Σχήμα 1.3.5, παρουσιάζουμε δύο παραδείγματα όπου οι παραγόμενες συνόψεις είναι λανθασμένες. Όμοια με το Σχήμα 1.3.4b, οι προσοχές μεταξύ των κόμβων θορύβου που ταιριάζουν στους δύο γράφους είναι πάνω από το όριο που θέσαμε. Ωστόσο, σε αυτές τις περιπτώσεις, οι κόμβοι θορύβου δεν επεκτείνουν το βασικό σχήμα αλλά το διαταράσσουν, με αποτέλεσμα μια θορυβώδη, λανθασμένη παραγόμενη σύνοψη.

MUTAG:

Λόγω της καλής απόδοσης της Μεθόδου II στο σύνολο Geometric Shapes, πραγματοποιήσαμε κάποια πειράματα στο MUTAG, για το οποίο η Μέθοδος I δεν παρείχε καλά αποτελέσματα. Αν και το MUTAG δεν έχει ground truth συνόψεις που αντιπροσωπεύουν τις κλάσεις, για να συγκρίνουμε τα αποτελέσματά μας, η Μέθοδος II παρήγαγε σχετικά συνεπείς συνόψεις που φαίνεται να αντιπροσωπεύουν αποτελεσματικά κάθε κλάση και να αποτυπώνουν τις σημαντικές πληροφορίες από τους γράφους στην είσοδο.

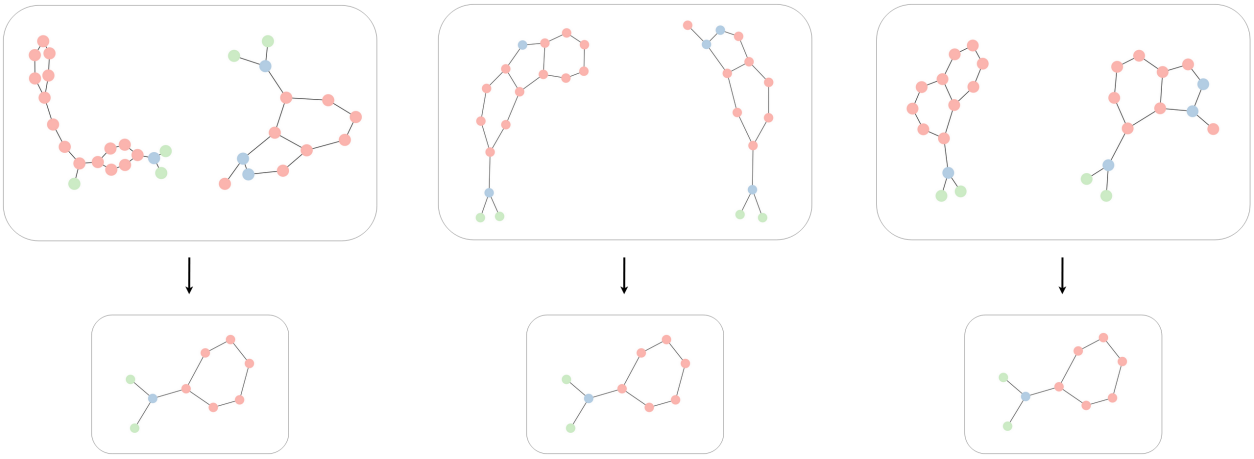


Figure 1.3.6: Ακριβή παραδείγματα για την μη μεταλλαξιογόνο κλάση του συνόλου δεδομένων MUTAG.

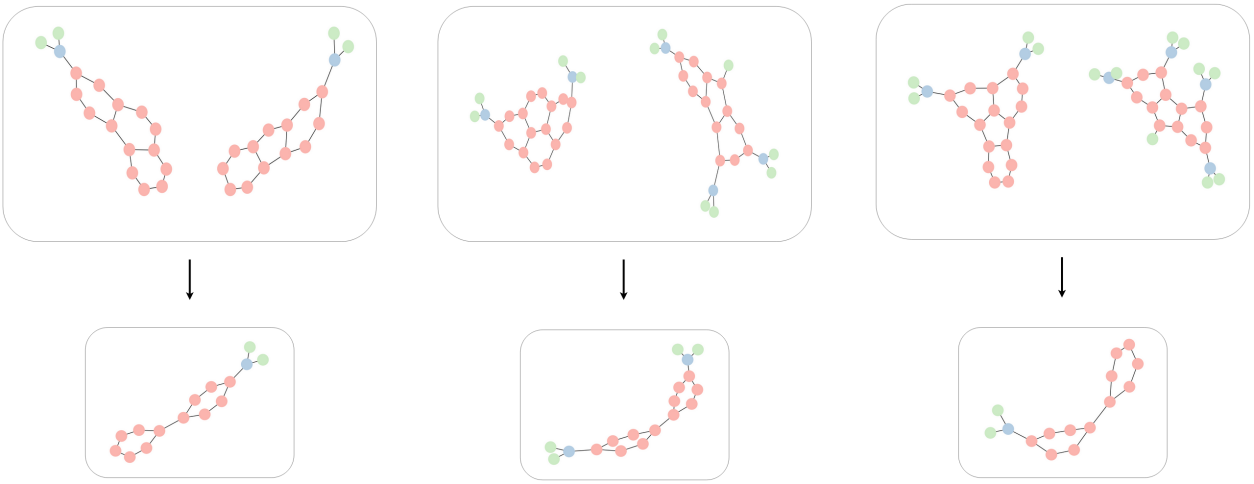


Figure 1.3.7: Ακριβή παραδείγματα για την μεταλλαξιογόνο κλάση του dataset MUTAG.

Στα Σχήματα 1.3.6 και 1.3.7, παρατηρούμε ότι για τη μη μεταλλαξιογόνο κλάση, οι παραγόμενες συνόψεις είναι συνεπείς και συνήθως σχηματίζονται ως ένωση ενός παραδείγματος κάθε ενός από τα δύο πρωτότυπα. Για την μεταλλαξιογόνο κλάση, οι παραγόμενες συνόψεις συχνά παρουσιάζουν παρόμοια μοτίβα και αποτελούνται από πολλαπλά παραδείγματα των δύο πρωτοτύπων, που αντιπροσωπεύουν σωστά τα μεταλλαξιογόνα χαρακτηριστικά.

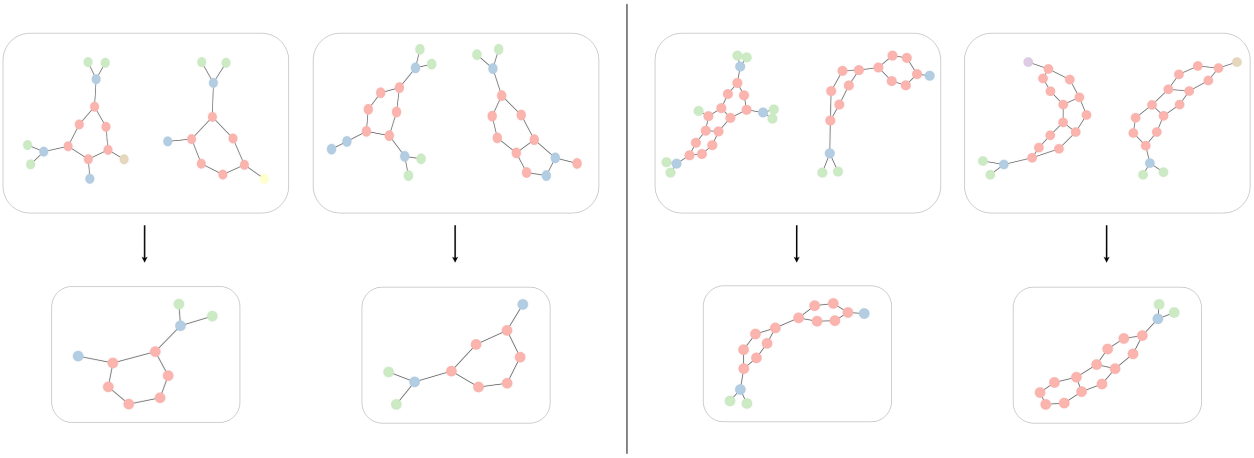


Figure 1.3.8: Λανθασμένα παραδείγματα με περισσότερους κόμβους για την μη μεταλλαξιογόνο κλάση (αριστερά) και την μεταλλαξιογόνο κλάση (δεξιά).

Ωστόσο, όμοια με το σύνολο δεδομένων Geometric Shapes, οι παραγόμενες συνόψεις για το MUTAG μπορεί επίσης να περιλαμβάνουν κόμβους που δεν πρέπει να είναι μέρος των συνόψεων, καθώς οι προσοχές μεταξύ τους είναι πάνω από το όριο που θέσαμε, ή να παραλείπουν κόμβους που πρέπει να περιλαμβάνονται λόγω του ότι οι προσοχές τους είναι κάτω από το όριο που θέσαμε. Αυτά τα παραδείγματα παρουσιάζονται στο Σχήμα 1.3.8 και στο Σχήμα 1.3.9.

Τέλος, ένα ενδιαφέρον μοτίβο παρατηρήθηκε κατά τη διάρκεια της ανάλυσης, εμφανιζόμενο πιο συχνά στην μη μεταλλαξιογόνο κλάση. Όταν δημιουργούμε τις σύνοψεις, αν χρησιμοποιήσουμε πρώτα τα ζεύγη αντιστοίχισης από το πρώτο επίπεδο μόνα τους, η παραγόμενη σύνοψη είναι συχνά ένα από τα δύο πρωτότυπα, συγκεκριμένα η μικρότερη, γραμμική δομή (αντιπροσωπευόμενη από τους πράσινους και μπλε κόμβους στα σχήματα). Στη συνέχεια, αφαιρώντας αυτούς τους κόμβους από τα ζεύγη αντιστοίχισης του δεύτερου επιπέδου, τα υπόλοιπα ζεύγη αντιστοίχισης συχνά παράγουν το άλλο πρωτότυπο, δηλαδή η κυκλική δομή (αντιπροσωπευόμενη από τους κόκκινους κόμβους στα σχήματα).

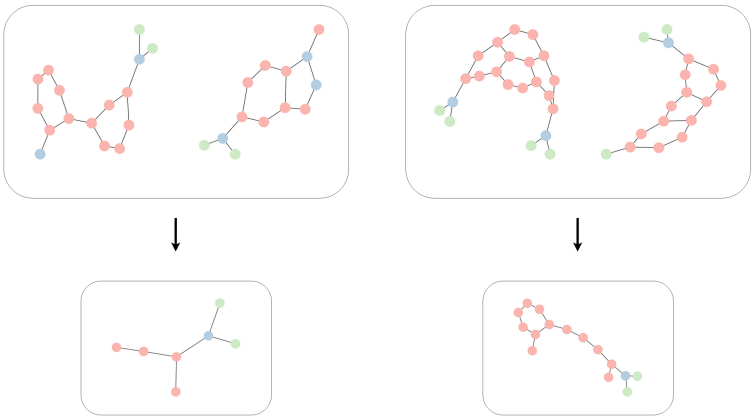
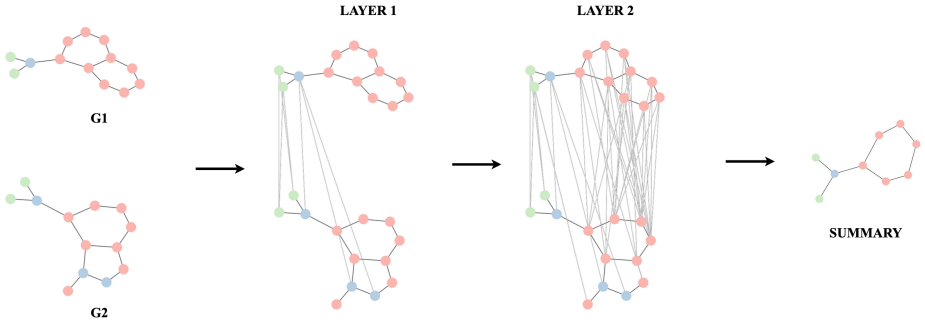
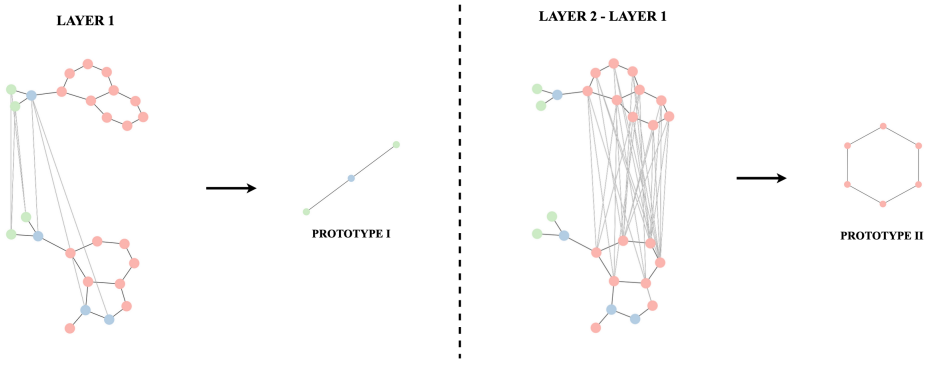


Figure 1.3.9: Λανθασμένα παραδείγματα με λιγότερους κόμβους για την μη μεταλλαξιογόνο κλάση (αριστερά) και την μεταλλαξιογόνο κλάση (δεξιά).

Αυτό το μοτίβο βρέθηκε να είναι σχετικά συνεπές, ιδίως το ότι τα ζεύγη αντιστοίχισης από το πρώτο επίπεδο παράγουν τη μικρότερη, γραμμική δομή. Αυτό συμβαίνει συχνά και στις δύο κλάσεις, καθώς οι αρχικές ισχυρές αντιστοιχίσεις που αναγνωρίζονται από το μοντέλο είναι συνήθως οι κόμβοι που σχηματίζουν αυτή τη γραμμική δομή. Αυτή η προσέγγιση της διαδοχικής χρήσης ζευγών αντιστοίχισης παρέχει μια εναλλακτική μέθοδο για την αναγνώριση μικρότερων βασικών υπογράφων των γράφων στο σύνολο δεδομένων. Ένα παράδειγμα αυτής της διαδικασίας παρουσιάζεται στο Σχήμα 1.3.10.



(a) Σύνοψη που δημιουργήθηκε εξάγοντας μια σύνοψη στο τέλος χρησιμοποιώντας όλα τα ζεύγη αντιστοίχισης.



(b) Συνόψεις που δημιουργήθηκαν διαδοχικά από τα ζεύγη αντιστοίχισης σε κάθε επίπεδο. Οι συνόψεις αντιστοιχίζονται με τα πρωτότυπα.

Figure 1.3.10: Σύγκριση μεθόδων δημιουργίας συνόψεων. Το πρώτο σχήμα δείχνει τη σύνοψη που δημιουργήθηκε εξάγοντας μια σύνοψη στο τέλος, χρησιμοποιώντας όλα τα ζεύγη αντιστοίχισης. Το δεύτερο σχήμα δείχνει τη διαδοχική δημιουργία συνόψεων από τα ζεύγη αντιστοίχισης σε κάθε επίπεδο.

1.4 Συμπεράσματα

1.4.1 Συζήτηση

Σε αυτή τη διπλωματική εργασία, ασχοληθήκαμε με το πρόβλημα της Σύνοψης Γράφων μέσω των Δικτύων Αντιστοίχισης Γράφων (ΔΑΓ). Συγκεκριμένα, επικεντρωθήκαμε στην ανάπτυξη μεθοδολογιών για τη δημιουργία συνόψεων που αντιπροσωπεύουν την κλάση από σύνολα δεδομένων γράφων, εξάγοντας υπογράφους από κάθε γράφο που αντιπροσωπεύουν αποτελεσματικά την κλάση στην οποία αυτός ανήκει. Για τον σκοπό αυτόν, εκπαιδεύσαμε ένα Δίκτυο Αντιστοίχισης Γράφων σε ένα πρόβλημα ομοιότητας γράφων, με στόχο την εξερεύνηση της ικανότητας του μοντέλου να μαθαίνει σημαντικά μοτίβα σε ζεύγη γράφων κατά την εκπαίδευση. Στη συνέχεια, αναπτύξαμε δύο μεθοδολογίες που αξιοποιούν τα διανύσματα των κόμβων για την αναγνώριση αυτών των μοτίβων και τη δημιουργία των τελικών συνόψεων. Η Μέθοδος Ι χρησιμοποιεί τα διανύσματα των κόμβων των δύο γράφων σε κάθε επίπεδο της αρχιτεκτονικής για τη δημιουργία υποψήφιων συνόψεων, χρησιμοποιώντας ένα επίπεδο TopK pooling, από τις οποίες προκύπτει η τελική σύνοψη. Η Μέθοδος ΙΙ επικεντρώνεται στον εντοπισμό ζευγών αντιστοίχισης στους δύο γράφους, δηλαδή ζευγών κόμβων που εμφανίζουν υψηλά αμφίδρομα σκορ προσοχής μεταξύ των γράφων. Χρησιμοποιώντας τα ζευγάρια αυτά ως περιορισμούς, βρίσκουμε τον μέγιστο κοινό επαγόμενο υπογράφο μεταξύ των δύο γράφων με τον αλγόριθμο McSplit.

Για την αξιολόγηση της απόδοσης του μοντέλου και των προτεινόμενων μεθόδων, δημιουργήσαμε ένα συνθετικό σύνολο δεδομένων Γεωμετρικών Σχημάτων, που αποτελείται από τέσσερα βασικά σχήματα: κυκλικούς, πλήρεις, γραμμικούς και αστεροειδείς γράφους. Προσθέσαμε θόρυβο με τη μορφή τυχαίων κόμβων και ακμών στους γράφους αυτούς. Αυτό το σύνολο δεδομένων παρείχε ένα ελεγχόμενο περιβάλλον με γνωστό ground truth, επιτρέποντας ακριβέστερη αξιολόγηση. Επιπλέον, πραγματοποιήσαμε πειράματα στο πραγματικό σύνολο δεδομένων MUTAG, που περιλαμβάνει μεταλλαξιογόνες και μη μεταλλαξιογόνες χημικές ενώσεις. Αν και το σύνολο δεδομένων MUTAG δεν περιλαμβάνει ground truth, περιέχει πρωτότυπα, που καθοδήγησαν την ποιοτική αξιολόγηση των συνόψεων που προβλέπει το μοντέλο μας.

Συγκρίναμε το μοντέλο μας με αρχιτεκτονικές ΝΔΓ που χρησιμοποιούν επίπεδα pooling, συγκεκριμένα τα DMoN, MinCut και JustBalance, για τη ομαδοποίηση των κόμβων και τη δημιουργία συνόψεων γράφων. Για να αξιολογήσουμε τη σημασία του μηχανισμού προσοχής μεταξύ γράφων που εισάγεται από τα ΔΑΓ, υλοποιήσαμε επίσης μια απλούστερη αρχιτεκτονική GAT, την οποία αξιολογήσαμε χρησιμοποιώντας μια προσέγγιση παρόμοια με τη Μέθοδο Ι. Στο σύνολο δεδομένων Geometric Shapes, το ΔΑΓ με τη Μέθοδο ΙΙ ήταν πολύ συνεπές και ξεπέρασε τα άλλα μοντέλα σε όλες τις εκδόσεις του συνόλου δεδομένων. Πέτυχε υψηλότερα σκορ ακρίβειας στις περισσότερες κλάσεις, με λίγες εξαιρέσεις στις κλάσεις πλήρων και αστεροειδών γράφων, όπου το ΔΑΓ με τη Μέθοδο Ι και το GAT αντίστοιχα απέδωσαν καλύτερα. Τα ΝΔΓ που βασίζονται στο pooling είχαν χαμηλή απόδοση, με σημαντικά χαμηλότερα σκορ ακρίβειας σε σύγκριση με τις προτεινόμενες μεθόδους μας.

Λόγω της καλής απόδοσης του ΔΑΓ με τη Μέθοδο ΙΙ στο σύνολο δεδομένων Geometric Shapes, πραγματοποιήσαμε πειράματα στο σύνολο δεδομένων MUTAG. Το μοντέλο απέδωσε καλά, συχνά αναγνωρίζοντας συνόψεις που αποτελούνταν είτε από ένα παράδειγμα των πρωτοτύπων για την μη μεταλλαξιογόνο κλάση, είτε από πολλαπλά παραδείγματα για τη μεταλλαξιογόνο κλάση. Αυτά τα αποτελέσματα αντανακλούν με ακρίβεια τη φύση των δύο κατηγοριών. Ωστόσο, όμοια με τα αποτελέσματα στο σύνολο δεδομένων Geometric Shapes, οι προβλεπόμενες συνόψεις περιλάμβαναν μερικές φορές κόμβους που δεν ανήκουν στα πρωτότυπα ή παρέλειπαν κόμβους που ανήκουν, λόγω του ότι οι προσοχές μεταξύ γράφων ήταν υψηλότερες ή χαμηλότερες από το καθορισμένο όριο, αντίστοιχα. Τέλος, ένα ενδιαφέρον μοτίβο που παρατηρήθηκε κατά τη διάρκεια της ανάλυσης, ιδιαίτερα στη μη μεταλλαξιογόνο κλάση, ήταν ότι, δημιουργώντας διαδοχικά συνόψεις σε κάθε επίπεδο χρησιμοποιώντας τα αντίστοιχα ζεύγη αντιστοίχισης και εξαιρώντας τους κόμβους τους από τα επόμενα επίπεδα, μπορούμε να κατασκευάσουμε τα πρωτότυπα του συνόλου δεδομένων.

Συνοψίζοντας, το ΔΑΓ, ιδιαίτερα με τη Μέθοδο ΙΙ, επέφερε καλά αποτελέσματα και ήταν σε θέση να αναγνωρίσει με ακρίβεια συνόψεις που αντιπροσωπεύουν τις κλάσεις. Αυτές οι συνόψεις παρέχουν πολύτιμες πληροφορίες για τα μοτίβα που αναγνωρίζει και μαθαίνει το μοντέλο κατά την εκπαίδευση στο πρόβλημα ομοιότητας γράφων. Αυτό ενισχύει την ερμηνευσιμότητα του μοντέλου, καθιστώντας τη συμπεριφορά του κατά την πρόβλεψη ομοιότητας πιο διαφανή και εξηγήσιμη.

1.4.2 Μελλοντικές Κατευθύνσεις

Για μελλοντικές εργασίες, μπορούν να εξερευνηθούν αρκετές κατευθύνσεις για να ενισχυθεί περαιτέρω η κατανόηση και η εφαρμογή των Δικτύων Αντιστοίχισης Γράφων στον τομέα της σύνοψης γράφων, αλλά και πέραν αυτού:

- **Βελτίωση Αλγορίθμων Μέγιστου Κοινού Υπογράφου (MCS) με ΝΔΓ:** Ενώ η αναγνώριση μέγιστων κοινών υπογράφων (MCS) χρησιμοποιώντας τις προσοχές μεταξύ γράφων ως περιορισμούς απέδωσε καλά αποτελέσματα στα πειράματά μας, το πρόβλημα του υπολογισμού του MCS είναι γνωστό ότι είναι NP-hard [27], αυξάνοντας σημαντικά το υπολογιστικό κόστος για μεγαλύτερους γράφους. Η εξερεύνηση προσεγγίσεων για την ανίχνευση MCS χρησιμοποιώντας ΝΔΓ, όπως το GLSearch [1], είναι μια υποσχόμενη κατεύθυνση για τη βελτίωση της αποτελεσματικότητας των μεθόδων μας.
- **Περαιτέρω Πειραματισμός σε Πραγματικά Δεδομένα:** Η διεξαγωγή πειραμάτων σε περισσότερα πραγματικά σύνολα δεδομένων, όπως το BA-Shape [68] και το Benzene [58], μπορεί να βοηθήσει στην περαιτέρω αξιολόγηση των προτεινόμενων μεθόδων, παρέχοντας βαθύτερη κατανόηση της συμπεριφοράς του μοντέλου και των μοτίβων που είναι σε θέση να μάθει σε πιο σύνθετες και ποικιλόμορφες δομές γράφων.
- **Εξερεύνηση Παραλλαγών ΔΑΓ που Επιτρέπουν την Μη Επιβλεπόμενη Εκπαίδευση με Χρήση Αντιθετικής Μάθησης:** Η διερεύνηση αντιθετικών παραλλαγών ΔΑΓ, όπως το CGMN [33], θα μπορούσε να επιτρέψει την εφαρμογή των μεθόδων μας σε μη επισημασμένα σύνολα δεδομένων, αξιοποιώντας τεχνικές αντιθετικής μάθησης για να ενισχυθεί η ικανότητα του μοντέλου να μαθαίνει χρήσιμες αναπαραστάσεις χωρίς να απαιτούνται επισημασμένα δεδομένα.

Ακολουθώντας αυτές τις μελλοντικές κατευθύνσεις, μπορούμε να συνεχίσουμε να εξερευνούμε το πρόβλημα της σύνοψης γράφων, βελτιώνοντας την αποτελεσματικότητα και την ακρίβεια των τεχνικών σύνοψης σε διάφορους τομείς, και ενισχύοντας την εξηγησιμότητα και την ερμηνευσιμότητα των ΝΔΓ.

Chapter 2

Introduction

Recent advancements in the field of Artificial Intelligence have been transformative for various domains, leading to significant breakthroughs in areas such as image recognition, natural language processing, and healthcare. Among these advancements, Graph Neural Networks (GNNs) have emerged as powerful models for analyzing and understanding complex graph-structured data, leveraging the inherent hierarchical information present in them. They have shown remarkable success across numerous applications, such as social network analysis, biological networks, drug discovery, and recommendation systems.

The focus of this thesis lies in addressing the problem of Graph Summarization. Given a multi-class graph dataset, we aim to extract a class-representative subgraph from each graph, that effectively represents the class to which the graph belongs. The core models we will work with are the Graph Matching Networks (GMNs), a specialized type of GNNs that, given a pair of graphs as input, computes a similarity score between them by jointly reasoning on the pair through a cross-graph attention-based matching mechanism. By training the model on a graph similarity task, we aim develop methodologies to identify and extract patterns that the model has learned, in order to inform the creation of the class-representative summaries.

Furthermore, the increasing demand for Explainable AI (XAI) has become critical due to the integration of neural networks in sectors such as healthcare and autonomous vehicles. The black-box nature of these models raises questions about their reliability and trustworthiness, necessitating the development of methodologies that provide insights into the models' decision-making processes. Summarizing complex graphs and creating prototype-like class-representative summaries, by identifying patterns that GNNs have learned, can contribute to better understanding their predictions and enhancing their explainability.

The outline of this thesis is as follows:

- We will begin by providing the theoretical background necessary for the methods and models we will use for the experiments, including foundational knowledge in Machine Learning and Deep Learning theory, in Graph Theory and in GNNs and their architectures.
- Subsequently, we will provide a comprehensive overview of the field of Graph Summarization and its subfields, covering existing traditional and GNN-based approaches.
- Lastly, we will provide a formal definition of the problem we aim to address, describe the GMN model we will use and present the proposed methodologies. We will evaluate our approach on a synthetic dataset tailored to the problem, comparing it to existing methods on the field of graph summarization, apply it to a real-world dataset, and present the quantitative and qualitative results.

Chapter 3

Background

The field of Artificial Intelligence (AI) encompasses a variety of technologies and methodologies aimed at emulating human-like intelligence through computational means. At its core, AI strives to enable machines to perform tasks that typically require human intelligence, such as visual perception, speech recognition, decision-making, and language understanding.

Machine Learning is a branch of AI focused on algorithms and statistical models that enable computer systems to effectively perform specific tasks by learning from data. This capability to learn and make decisions with minimal human intervention is what distinguishes ML from traditional computational approaches. Within the domain of Machine Learning, further advancements have led to the evolution of Deep Learning (DL).

Deep Learning utilizes model architectures with multiple layers, called deep neural networks. Inspired by the human brain, deep neural networks can learn from large amounts of data, allowing them to capture complex patterns and make decisions based on them. The depth of these models, which refers to the number of processing layers, enables the handling of vast complexities in data, making deep learning particularly effective for tasks that involve high-dimensional data such as images, sound, and text.

Contents

3.1 Machine Learning 30
3.1.1 Categories of Learning 30
3.1.2 Data Modalities 30
3.2 Deep Learning 31
3.2.1 Core Components of Neural Networks 31
3.2.2 Training Neural Networks 33
3.2.3 Evaluation and Validation 37
3.2.4 Embeddings 38
3.2.5 Transformers 38

3.1 Machine Learning

Machine Learning (ML) encompasses various methodologies that allow systems to learn from data and improve their performance. This section covers the different types of learning methodologies and the various data types used in ML.

3.1.1 Categories of Learning

Machine learning encompasses various paradigms, each tailored to different objectives and data types. The primary distinguishing factor among these paradigms is the presence or absence of supervisory signal during the learning process. The most common learning paradigms are detailed below.

Supervised Learning

Supervised learning involves training a model f on a dataset containing input-output pairs (x, y), where x represents the input features and y the corresponding labels. The goal is to learn a mapping $f : X \to Y$ that makes accurate predictions for y given new x values. Commonly, this involves minimizing a loss function $L(y, f(x))$ that measures the prediction errors over all examples in the training set. This framework is commonly used for classification and regression tasks.

Unsupervised Learning

Unsupervised learning algorithms operate on data without labels, aiming to discover the intrinsic structure within the dataset. The objective is to identify underlying patterns or structures within the data. Clustering and dimensionality reduction are common tasks in unsupervised learning. Algorithms like k-means clustering, hierarchical clustering, and principal component analysis (PCA) are popular in this category.

Semi-Supervised Learning

Semi-supervised learning is a hybrid approach that leverages both labeled and unlabeled data during training. Typically, a small amount of labeled data is supplemented with a large amount of unlabeled data. This method can significantly improve learning accuracy when labeled data is scarce or expensive to obtain. Techniques in semi-supervised learning include self-training, co-training, and graph-based methods. Applications include text classification, image recognition, and bioinformatics.

Self-Supervised Learning

Self-supervised learning is a type of unsupervised learning where the data itself provides the supervision. The model learns to predict part of the data from other parts, effectively creating its own labels. This approach is particularly useful for pre-training models on large datasets where manual labeling is not feasible. Examples include predicting missing words in sentences or predicting future frames in a video. This type of learning is commonly used in fields like natural language processing and computer vision.

Reinforcement Learning

Reinforcement learning (RL) involves training an agent to make a sequence of decisions by rewarding it for good actions and penalizing it for bad ones. The agent learns to maximize cumulative rewards over time, interacting with an environment that provides feedback in the form of rewards or punishments. RL algorithms include Q-learning, policy gradients, and deep reinforcement learning. This type of learning is prominently used in robotics, game playing, and autonomous systems.

3.1.2 Data Modalities

In the diverse field of machine learning, data modalities represent the various forms and sources of data that algorithms can process to learn and make predictions. The most common datatypes and their applications are detailed below.

Structured Data

Structured data refers to information that is highly organized and easily searchable in databases. This type of data is typically stored in tabular formats with rows and columns, such as spreadsheets or relational databases. Machine learning models utilize structured data effectively for predictive analytics, where attributes directly inform the predictive models.

Unstructured Data

Unstructured data lacks a predefined format or organization, making it more challenging to collect, process, and analyze. This type of data includes text, images, videos, and audio files. Natural language processing (NLP) and computer vision techniques are often used to extract meaningful information from unstructured data, enabling tasks such as sentiment analysis, text summarization, and language translation.

Image Data

Image data consists of visual information captured in the form of pictures or frames from videos. It is typically represented as pixel arrays, where each pixel contains color and intensity values. Image data is used in various applications, such as medical imaging, facial recognition, and autonomous driving. Convolutional neural networks (CNNs) are a popular architecture for processing and analyzing this type of data.

Time-Series Data

Time-series data is a sequence of data points collected or recorded at specific time intervals. This type of data is often used in forecasting and trend analysis. Examples include stock prices, weather data, and sensor readings. Specialized models like ARIMA for forecasting, and Long Short-Term Memory networks (LSTMs), a type of recurrent neural network, are often used to predict future data points in a series.

Graph Data

Graph data represents relationships between entities and is structured as nodes (vertices) and edges. This type of data is used to model networks such as social networks, communication networks, and biological networks. Graph-based algorithms and techniques are employed to analyze the structure and dynamics of these networks. Graph neural networks (GNNs) have emerged as powerful tools for learning and inference on graph-structured data.

3.2 Deep Learning

Deep Learning, a significant advancement in the field of Machine Learning, has emerged from the need to handle and learn from large volumes of data with high complexity. Building on the foundation of traditional machine learning, deep learning utilizes neural networks with many layers to automatically learn representations and features from raw data. This approach has proven particularly effective in domains such as computer vision, natural language processing, and speech recognition, where it significantly outperforms traditional methods.

Driven by the availability of large datasets, powerful computational resources, and improved algorithms, deep learning models have become essential tools in AI research and applications. Their ability to process and analyze complex data has led to breakthroughs in various fields, making deep learning a cornerstone of modern artificial intelligence.

3.2.1 Core Components of Neural Networks

Deep learning models are constructed from fundamental elements known as artificial neurons. These neurons are organized into layers and networks and are linked by weights that are adjusted during training, enabling them to learn complex patterns from data.

Neurons and Perceptrons

Artificial neurons, inspired by biological neurons, are the basic units of neural networks. A single artificial neuron, or perceptron, performs a mathematical operation on its inputs to produce an output. This operation involves calculating a weighted sum of the inputs and applying an activation function.

Mathematically, a perceptron takes a vector of inputs $\mathbf{x} = [x_1, x_2, \ldots, x_n]$ and computes a weighted sum:

$$z = \sum_{i=1}^{n} w_i x_i + b$$

where w_i are the weights associated with each input, and b is the bias term. The output is obtained by applying an activation function ϕ to this weighted sum:

$$y = \phi(z)$$

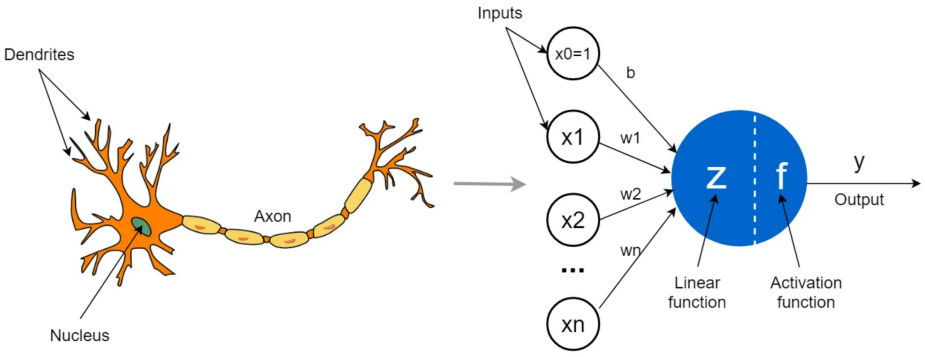


Figure 3.2.1: An illustration of the structure of a biological neuron and an artificial neuron (perceptron).

Activation Functions

Activation functions play an important role in the architecture of neural networks, as they introduce non-linearity to the model, which is essential for learning complex patterns and relationships in data. Without non-linearity, a neural network would simply behave like a linear model, regardless of the number of layers, and would be unable to capture the intricate structures and dependencies inherent in real-world data. The most commonly used activation functions are presented below.

- **Sigmoid**: The sigmoid function outputs values in the range $(0, 1)$, making it useful for binary classification tasks. Its mathematical form is:

  $$\sigma(z) = \frac{1}{1 + e^{-z}}$$

  It is differentiable everywhere in its domain, which makes it compatible with gradient-based optimization techniques like backpropagation. However, it can suffer from vanishing gradients, which slows down learning in deep neural networks.

- **Tanh**: The hyperbolic tangent function outputs values in the range $(-1, 1)$. It is zero-centered, which helps during optimization. Its form is:

  $$\tanh(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

  Like the sigmoid function, tanh can also face issues with vanishing gradients.

- **ReLU (Rectified Linear Unit)**: The ReLU function outputs the input directly if it is positive; otherwise, it outputs zero:

$$\text{ReLU}(z) = \max(0, z)$$

  ReLU helps mitigate the vanishing gradient problem and is computationally efficient, making it widely used in deep learning. Variants like Leaky ReLU and Parametric ReLU aim to address its tendency to output zero for all negative inputs.

- **Leaky ReLU**: This variant allows a small, non-zero gradient when the input is negative:

$$\text{Leaky ReLU}(z) = \max(\alpha z, z)$$

  where α is a small constant. It mitigates the "dying ReLU" problem where neurons could become inactive during training.

The choice of activation function can significantly impact the performance and training dynamics of a neural network and is influenced by the specific characteristics of the problem at hand and the network's architecture.

Basic Layers

Neural networks are composed of layers of neurons, each serving specific functions to collectively create a powerful model. The various types of layers are designed to handle different aspects of data processing, enabling the network to learn and represent complex patterns and features effectively. The most commonly used layers are presented below.

- **Fully Connected (Dense) Layers**: In a fully connected layer, every neuron is connected to every neuron in the previous layer. The operation performed by a dense layer is a matrix multiplication followed by an activation function:

$$\mathbf{y} = \phi(\mathbf{W}\mathbf{x} + \mathbf{b})$$

  where $\mathbf{W}$ is the weight matrix, $\mathbf{b}$ is the bias vector, and ϕ is the activation function applied element-wise. Fully connected layers are typically used in the final stages of a neural network to combine features learned in previous layers and make predictions.

- **Convolutional Layers**: Convolutional layers are primarily used in image processing tasks. They apply convolution operations to the input, which involves sliding a filter (or kernel) over the input and computing the dot product between the filter and local regions of the input. This helps capture spatial hierarchies and patterns in the data.

- **Pooling Layers**: Pooling layers reduce the spatial dimensions of the input, which decreases computational load and helps control overfitting. Max pooling and average pooling are common types.

- **Recurrent Layers**: Recurrent layers are designed for sequential data, such as time series or text. They maintain hidden states that capture information from previous steps in the sequence. A basic Recurrent Neural Network (RNN) operation is:

$$h_t = \phi(W_h h_{t-1} + W_x x_t + b)$$

  where h_t is the hidden state at time step t, x_t is the input at time step t, and W_h, W_x, b are the parameters of the network.

These components form the foundational elements of deep learning models. By combining different types of layers and activation functions, deep learning models can learn to perform a wide range of tasks.

3.2.2 Training Neural Networks

Training a neural network involves adjusting its parameters to minimize the error between predicted and actual outputs. This process includes various techniques such as using loss functions, gradient descent, backpropagation, and regularization methods, which are detailed below.

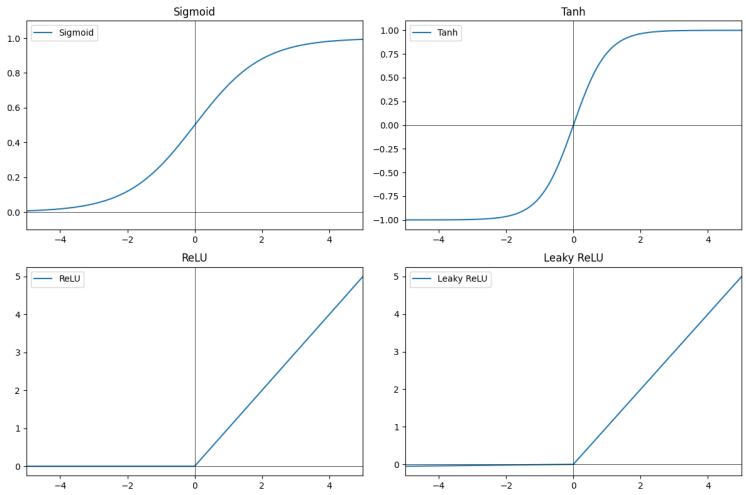


Figure 3.2.2: An illustration of the diagrams of the discussed Activation Functions.

Loss Functions

In deep learning, a loss function, also known as a cost function, quantifies the differences between the predicted outputs of the model and the actual target values. It serves as a guide for the optimization process by providing a criterion that the model aims to minimize during training. The choice of loss function depends on the type of problem being solved, such as classification or regression.

- **Mean Squared Error (MSE)**: Commonly used for regression tasks, it calculates the average of the squares of the differences between the predicted and actual values. The formula for MSE is:

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2$$

  where $\hat{y}_i$ represents the predicted value, y_i is the actual value, and N is the number of observations. MSE is sensitive to outliers due to the squaring of differences.

- **Cross-Entropy Loss**: Cross-entropy loss is widely used for classification tasks, particularly in binary and multi-class classification. For binary classification, the cross-entropy loss is defined as:

$$L = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)]$$

  For multi-class classification, the categorical cross-entropy loss generalizes to:

$$L = -\frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{C} y_{i,c}\log(\hat{y}_{i,c})$$

  where C is the number of classes, $y_{i,c}$ is a binary indicator (0 or 1) if class label c is the correct classification for observation i, and $\hat{y}_{i,c}$ is the predicted probability that observation i belongs to class c.

Gradient Descent

Gradient Descent is an optimization algorithm widely used in machine learning and deep learning for minimizing the loss function of a model. The method updates the parameters of the model iteratively to find the minimum value of the loss function.

The basic idea behind gradient descent is to update each parameter in the model in a direction that reduces the loss function, which is determined by the negative gradient of the loss function at the current point. The update rule for gradient descent is:

$$\theta' = \theta - \eta \cdot \nabla_\theta J(\theta)$$

where θ represents the parameters, η is the learning rate (a scalar that determines the step size during the minimization process), and $\nabla_\theta J(\theta)$ is the gradient of the loss function.

There are several variants of the Gradient Descent algorithm, which can improve its efficiency and convergence:

- **Batch Gradient Descent**: Computes the gradient of the loss function using the entire dataset. This method is precise but can be very slow and computationally expensive with large datasets.
- **Stochastic Gradient Descent (SGD)**: Computes the gradient using a single sample at a time. This helps to speed up the computations and can lead to faster convergence, especially with large datasets. However, the updates can be noisy due to the high variance of the gradients based on single samples.
- **Mini-Batch Gradient Descent**: A compromise between batch and stochastic gradient descent, this method uses small batches of samples to compute the gradient. It balances the efficiency of batch processing with the reduced variance of the estimates from stochastic processing.

Backpropagation

Backpropagation [57] is an algorithm used for training neural networks, enabling the efficient computation of gradients of the loss function with respect to the weights. It is primarily used in conjunction with optimization methods like gradient descent to adjust the weights of the network based on the error between the predicted outputs and the actual outputs.

The backpropagation process involves calculating the gradient of the loss function at the output and propagating this error backward through the network, layer by layer. It computes the partial derivatives of the loss function with respect to each weight by applying the chain rule from calculus:

$$\frac{\partial J}{\partial w_{ij}} = \frac{\partial J}{\partial a_j} \cdot \frac{\partial a_j}{\partial z_j} \cdot \frac{\partial z_j}{\partial w_{ij}}$$

where:

- J is the loss function,
- w_{ij} is the weight between the i^{th} and j^{th} neurons,
- a_j is the activation of the j^{th} neuron,
- z_j is the input sum to the j^{th} neuron.

This gradient information is used to update the weights in a direction that aims to reduce the loss:

$$w'_{ij} = w_{ij} - \eta \frac{\partial J}{\partial w_{ij}}$$

where η is the learning rate. This weight update rule is applied iteratively across the network, updating all weights to minimize the overall loss.

Generalization, Overfitting, and Underfitting

When training neural networks, achieving a balance between model complexity and performance is crucial to ensure that the model generalizes well to new, unseen data. Generalization refers to the model's ability to perform accurately on data that it has not encountered during training. However, models can suffer from overfitting or underfitting, which adversely affect generalization.

Overfitting occurs when a model learns the training data too well, including its noise and outliers. This results in excellent performance on the training data but poor performance on validation and test data. Overfitted models are typically too complex for the amount of training data available. To mitigate overfitting, techniques such as regularization are employed.

Conversely, underfitting happens when a model is too simple to capture the underlying patterns in the data, leading to poor performance on both training and validation/test data. Underfitted models fail to learn from the training data adequately. Addressing underfitting involves increasing model complexity, reducing regularization strength, improving feature engineering, and ensuring sufficient training time.

Regularization

Regularization techniques are methods used to prevent overfitting, ensuring that the model generalizes well to new, unseen data. Overfitting occurs when a model learns the detail and noise in the training data to the extent that it negatively impacts the performance of the model on new data. Regularization provides a way to constrain or regularize the learning process, typically by adding a penalty on the complexity of the model. The following are commonly used regularization techniques in deep learning:

- **Dropout**: Dropout is a regularization technique that involves randomly setting a fraction of the neurons' activations to zero during each training iteration. This prevents the network from becoming too reliant on particular neurons.

  Mathematically, for each training sample, dropout modifies the forward pass as follows:

  $$a^l_{\text{dropout}} = a^l \odot \mathbf{r}$$

  where $\mathbf{r}$ is a vector of Bernoulli random variables with probability p of being 1 (kept) and $1-p$ of being 0 (dropped). During testing, all neurons are used, and their outputs are scaled by p.

- **L1 and L2 Regularization**: L1 and L2 regularization add penalty terms to the loss function to constrain the magnitude of the weights.
  - **L1 Regularization**: Adds the absolute values of the weights to the loss function, promoting sparsity.

    $$L_{\text{total}} = L + \lambda \sum_i |w_i|$$

  - **L2 Regularization (Ridge Regression)**: Adds the squared values of the weights to the loss function, encouraging smaller weights.

    $$L_{\text{total}} = L + \lambda \sum_i w_i^2$$

  where L is the original loss function, w_i are the weights, and λ is the regularization parameter controlling the penalty's strength.

- **Early Stopping**: Early stopping involves monitoring the model's performance on a validation set during training and stopping the training process when performance stops improving. This prevents the model from overfitting the training data.

- **Data Augmentation**: Data augmentation generates new training samples by applying random transformations such as rotations, translations, and flips to the existing data. This technique is particularly useful in image classification tasks, as it increases the diversity of the training set and reduces overfitting.

Hyperparameter Tuning

Hyperparameter tuning involves selecting the optimal values for parameters like the learning rate, batch size, number of layers, and regularization strength, which are set before training begins. Proper tuning of these hyperparameters can significantly enhance a model's performance and generalization capability. Common

methods for hyperparameter tuning include grid search, random search, and Bayesian optimization. These methods systematically explore different combinations of hyperparameters to find the best configuration.

3.2.3 Evaluation and Validation

Training, Validation, and Test Sets

Proper evaluation of a neural network's performance requires dividing the available data into three distinct sets: the training set, the validation set, and the test set. This division ensures that the model is trained, validated, and tested on different subsets of data to accurately assess its performance and generalization ability.

- **Training Set**: The training set is the subset of data used to train the neural network. The model learns from this data by adjusting its parameters (weights and biases) to minimize the loss function. Typically, the majority of the available data is allocated to the training set.
- **Validation Set**: The validation set is used to tune the model's hyperparameters and to provide an unbiased evaluation of the model during the training process.
- **Test Set**: The test set is a separate subset of data used to evaluate the final model's performance after training and validation. It provides an unbiased estimate of the model's generalization ability to new, unseen data.

Performance Metrics

Evaluating the performance of a neural network on classification tasks requires metrics that reflect the model's ability to correctly predict the classes of new, unseen data. The most common performance metrics are presented below.

- **Accuracy**: Accuracy is the ratio of correctly predicted instances to the total number of instances. It is a simple and intuitive metric but can be misleading when dealing with imbalanced datasets.

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$

- **Precision**: Precision measures the proportion of true positive predictions out of all positive predictions. It is useful for scenarios where the cost of false positives is high.

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

- **Recall**: Recall measures the proportion of true positive predictions out of all actual positives. It is useful when the cost of false negatives is high.

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

- **F1 Score**: The F1 score is the harmonic mean of precision and recall, providing a balance between the two. It is particularly useful for imbalanced datasets.

$$\text{F1-Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

where TP represents True Positives, TN represents True Negatives, FP represents False Positives, and FN represents False Negatives.

3.2.4 Embeddings

A key concept we will often refer to in the context of this thesis is *embeddings*. Embeddings are continuous vector representations of data in a lower-dimensional space, a transformation that facilitates its efficient processing and analysis by neural networks. The goal of embeddings is to bring semantically similar objects closer together in the embedding space, by effectively capturing the semantics of the input.

Embeddings are most commonly used in the field of natural language processing (NLP), serving as word or sentence representations. They generally are real-valued vectors representing the semantic meaning of words in such a way that maps words with similar meaning closer together in the vector space. With the emergence of Graph Neural Networks (GNNs), a class of models we will cover in a later chapter, the concept of embeddings has been extended beyond NLP. In GNNs, embeddings can represent node features, capturing the intrinsic characteristics of individual nodes within a graph, and edge embeddings can encapsulate the properties and relationships between connected nodes. Furthermore, graph-level embeddings can summarize subgraphs or entire graphs, encoding structural and feature information in a compact form.

3.2.5 Transformers

The Transformer architecture, introduced by Vaswani et al. [64] in 2017, has revolutionized natural language processing (NLP) and various other fields. Its self-attention mechanism enables capturing complex dependencies in sequential data, making it particularly effective in tasks such as language translation, summarization, and image processing. Moreover, as we will see later on, the attention mechanism inspired the Graph Attention Networks [65].

The Transformer architecture is summarized below:

- **Input Representation**: The input sequence is first embedded into continuous vector representations. Positional embeddings are added to these embeddings to provide information about the position of each token in the sequence.
- **Encoder**: The encoder consists of a stack of identical layers. Each layer has two sub-layers:
  - **Multi-Head Self-Attention**: This mechanism allows the model to focus on different parts of the input sequence simultaneously. The mathematical formulation for self-attention is as follows:

    Given an input sequence of token embeddings $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n]$, the self-attention is computed as:

    $$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}}\right)\mathbf{V}$$

    where $\mathbf{Q}$ (queries), $\mathbf{K}$ (keys), and $\mathbf{V}$ (values) are linear transformations of the input $\mathbf{X}$:

    $$\begin{aligned}\mathbf{Q} &= \mathbf{X}\mathbf{W}^Q,\\ \mathbf{K} &= \mathbf{X}\mathbf{W}^K,\\ \mathbf{V} &= \mathbf{X}\mathbf{W}^V\end{aligned}$$

    Here, $\mathbf{W}^Q$, $\mathbf{W}^K$, and $\mathbf{W}^V$ are learnable weight matrices, and d_k is the dimension of the key vectors. The scaled dot-product attention is computed by taking the dot product of the query and key vectors, scaling by $\sqrt{d_k}$, applying a softmax function to obtain the attention weights, and then multiplying by the value vectors.
  - **Position-wise Feed-Forward Neural Network**: After the attention mechanism, each token's representation is passed through a position-wise feed-forward neural network. This introduces non-linearity and further refines the token representations.

  Residual connections, followed by layer normalization, are employed around each of the sub-layers.

- **Decoder**: The decoder also consists of a stack of identical layers, each containing three sub-layers:
  - **Masked Multi-Head Self-Attention**: This sub-layer acts similar to the corresponding encoder's sub-layer but with a mask applied to prevent attending to future positions during training.
  - **Multi-Head Encoder-Decoder Attention**: This sub-layer focuses on the encoded input sequence, allowing the decoder to consider the relevant parts of the input during sequence generation.
  - **Position-wise Feed-Forward Neural Network**: Similar to the encoder, this sub-layer follows the attention mechanisms.

  As with the encoder, residual connections are used around each sub-layer, followed by layer normalization.
- **Output Generation**: The output of the final decoder layer is transformed into probability distributions over the output vocabulary using a linear transformation followed by a softmax activation. Throughout the training process, the model is fed with a word sequence as input to predict the subsequent word.

Transformers have found applications across a wide range of fields due to their ability to model long-range dependencies and handle large-scale data efficiently. In natural language processing (NLP), they are used for tasks such as machine translation, sentiment analysis, and text summarization. Beyond NLP, transformers are also employed in computer vision for image classification and object detection, as well as in the healthcare sector for tasks like medical diagnosis and protein structure prediction. Their versatility and effectiveness have made transformers a foundational model in many state-of-the-art AI systems.

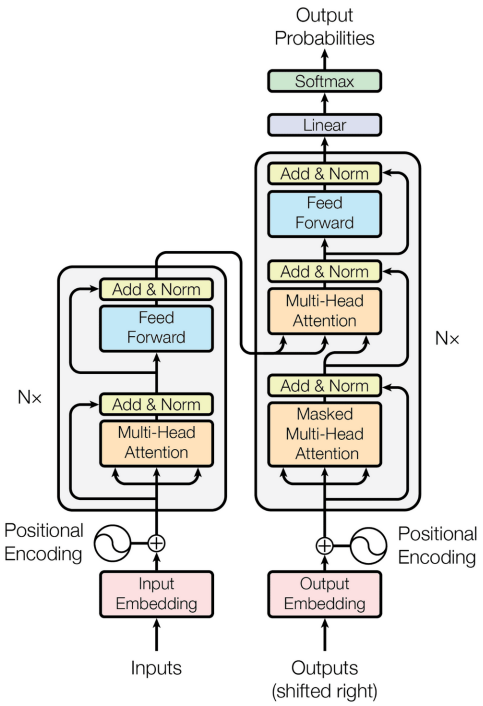


Figure 3.2.3: An illustration of the Transformer - model architecture, as presented in [64].

Chapter 4

Graphs

Contents

4.1 Graph Theory **42**
4.2 Graph Similarity **43**
4.2.1 Graph Edit Distance 43
4.2.2 Graph Kernels 44
4.3 Graph Matching **46**
4.3.1 Exact Graph Matching (Graph Isomorphism) 46
4.3.2 Inexact Graph Matching 47
4.4 Maximum Common Subgraph **47**
4.4.1 Maximum Common Induced Subgraph (MCIS) 47
4.4.2 Maximum Common Edge Subgraph (MCES) 47

4.1 Graph Theory

Graphs are fundamental mathematical structures used to model pairwise relations between objects. Formally, a graph is denoted as $G = (V, E)$, where V is a set of vertices, also known as nodes or points, and E is a set of edges, also known as links or lines. The vertices u and v of an edge {u, v} are called the edge's endpoints. The *order* of a graph is its number $|V|$ of vertices, usually denoted by n. The size of a graph is its number $|E|$ of edges, typically denoted by m.

In terms of connectivity, any two vertices in a graph may be connected by zero, one, or multiple edges. Graphs that allow multiple edges to have the same endpoints are called multigraphs. Sometimes, graphs are allowed to contain loops, which are edges that join a vertex to itself. The degree of a vertex is the number of edges connected to it, which helps define its connectivity within the graph. In a graph of order n, the maximum degree of each vertex is $n - 1$ (or $n + 1$ if loops are allowed, because a loop contributes 2 to the degree), and the maximum number of edges is $n(n - 1)/2$ (or $n(n + 1)/2$ if loops are allowed).

Graphs can be categorized based on the directionality of their edges:

- **Directed Graphs**: In directed graphs, each edge has a direction associated with it, typically represented with an arrow pointing from one vertex to another. This direction indicates the asymmetry in the relationship, meaning the connection from vertex u to vertex v does not imply a connection from v to u.
- **Undirected Graphs**: In undirected graphs, edges are bidirectional, implying that each connection is symmetric. Thus, an edge from u to v automatically entails an edge from v to u.

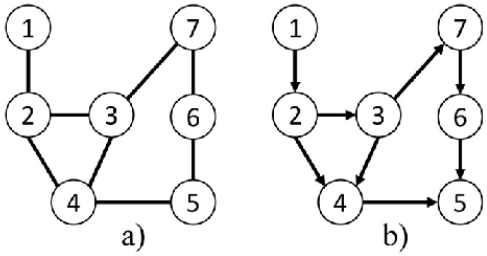


Figure 4.1.1: An example of a directed and an undirected graph.

A graph is fully determined and can be represented by its adjacency matrix or its adjacency list:

- **Adjacency Matrix**: An adjacency matrix A of a graph of order n is a $n \times n$ matrix where non-zero elements indicate the presence of an edge between the vertices. In undirected graphs, this matrix is symmetric, which simplifies certain computations.
- **Adjacency List**: This is a more space-efficient way to represent graphs, particularly sparse ones. Each vertex maintains a list of all vertices to which it is directly connected, facilitating efficient traversal and modifications.

Vertices and edges can also have attributes or weights associated with them, which are often represented as matrices or lists that hold data relevant to each vertex or edge. Such weights might represent for example costs, lengths or capacities, depending on the problem at hand. Such graphs arise in many contexts, for example in shortest path problems such as the traveling salesman problem.

Furthermore, the following key concepts describe the relationships and structures within graphs:

- **Neighborhood**: The neighborhood of a vertex v in a graph is the set of all vertices adjacent to v. Formally, for a vertex v, its neighborhood $N(v)$ is defined as $\{u \in V \mid \{u, v\} \in E\}$.
- **Path**: A path in a graph is a sequence of edges that connects a sequence of distinct vertices. A path is simple if all vertices (and hence all edges) are distinct.

- **Walk**: A walk is a sequence of edges and vertices where repetition of vertices and edges is allowed. A walk can be open, if it starts and ends at different vertices, or closed, if it starts and ends at the same vertex.
- **Cycle**: A cycle is a closed path with no repeated vertices or edges, except for the starting and ending vertex.
- **Subgraph**: A subgraph is a graph formed from a subset of the vertices and edges of a larger graph. If a subgraph includes all edges between the vertices that appear in the original graph, it is called an induced subgraph.

4.2 Graph Similarity

Graph similarity measures are mathematical tools used to quantify the similarity between two graphs. These measures are essential for comparing and analyzing the structural properties of different graphs, enabling tasks like graph classification, clustering, and matching. Traditional methods for graph similarity include Graph Edit Distance (GED) and Graph Kernels, which are detailed below.

4.2.1 Graph Edit Distance

Graph Edit Distance (GED) is a widely-used method for measuring the similarity between two graphs. The first mathematical formulation was introduced by Alberto Sanfeliu and King-Sun Fu in 1983 [59]. It is defined as the minimum number of edit operations required to transform one graph into another. These edit operations include insertions, deletions, and substitutions of nodes and edges. The GED between two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ can be formally defined as:

$$d(G_1, G_2) = \min_{\gamma \in \Gamma} \sum_{e \in \gamma} c(e)$$

where Γ represents the set of all possible sequences of edit operations that transform G_1 into G_2, and $c(e)$ is the cost associated with the edit operation e. The main edit operations include:

- **Node Insertion/Deletion**: Adding or removing a node from the graph.
- **Edge Insertion/Deletion**: Adding or removing an edge from the graph.
- **Node Substitution**: Replacing a node in one graph with a node from another graph, which might involve changing its label or attributes.
- **Edge Substitution**: Changing an edge in one graph to match an edge in another graph, which might involve altering its weight or attributes.

The cost of each operation can vary depending on the application and the specific characteristics of the graphs. For example, in some applications, node substitutions might be more expensive than edge insertions, reflecting the relative importance of preserving node identities.

While GED is a powerful measure, computing it is NP-hard, which makes its exact computation is infeasible for large graphs. Therefore, various heuristic and approximation algorithms have been developed to make GED computation more practical. These methods often trade off exactness for efficiency, providing near-optimal solutions while reducing the time complexity.

Exact Algorithms for GED Calculation: Exact algorithms for computing GED typically involve combinatorial search techniques that explore all possible sequences of edit operations. These algorithms guarantee finding the minimum edit distance but are computationally expensive and impractical for large graphs. One commonly used exact algorithm is the **A* Search Algorithm**, which is an informed search algorithm that uses heuristics to guide the search process towards the most promising paths, thereby reducing the number of states explored. In the context of GED, the algorithm maintains a priority queue of partial edit sequences and expands the most promising sequence based on a cost function $f(n) = g(n) + h(n)$, where $g(n)$ is the cost of the sequence so far, and $h(n)$ is the heuristic estimate of the remaining cost to reach the goal.

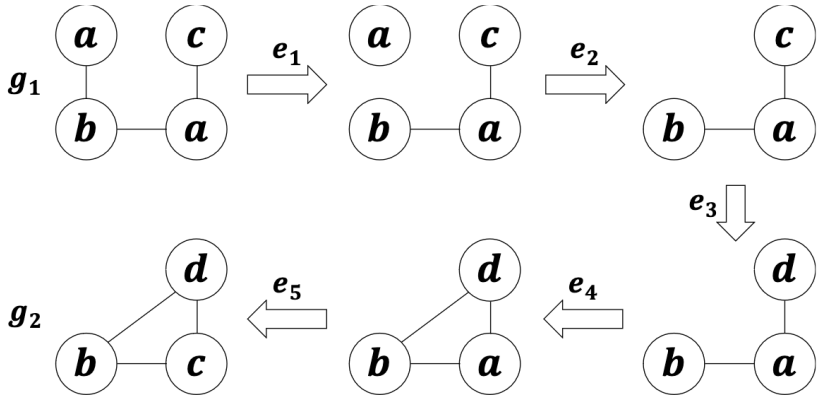


Figure 4.2.1: Graph Edit Distance between a pair of graphs.

Approximate Algorithms for GED Calculation: Approximate algorithms for GED aim to find near-optimal solutions more efficiently by reducing the search space or using probabilistic methods. These algorithms include:

- **Hungarian Algorithm**: The Hungarian algorithm [42], also known as the Kuhn-Munkres algorithm, is used for finding the maximum matching in a bipartite graph. In the context of GED, it can be applied to match nodes of the two graphs, minimizing the total cost of the edit operations. Although it provides an exact solution for the matching problem, it is used as a subroutine in approximate GED algorithms to improve efficiency.
- **A* Beamsearch**: A* Beamsearch, introduced in [53], is a variation of the A* search algorithm that limits the number of paths explored by keeping only the most promising paths at each level of the search tree. This approach significantly reduces the computational complexity while still providing good approximate solutions for the GED.
- **Bipartite Matching**: Bipartite matching algorithms [35] are used to find maximum matchings in bipartite graphs. These algorithms can be adapted to approximate GED by treating the nodes of the two graphs as bipartite sets and finding a matching that minimizes the edit distance.

4.2.2 Graph Kernels

Graph Kernels provide another approach to measuring graph similarity, particularly useful in the context of machine learning. They map graphs into a high-dimensional feature space and compute similarities using inner products. This approach leverages the power of kernel methods, making it possible to apply algorithms like Support Vector Machines (SVMs) to graph-structured data. Some popular graph kernels include:

- **Random Walk Kernels**: Random walk kernels measure similarity based on the number of matching random walks in two graphs. The most widely-used kernel from this family is the Geometric Random Walk Kernel [28] which compares walks up to infinity assigning a weight λ^k ($\lambda < 1$) to walks of length k in order to ensure convergence of the corresponding geometric series. Given graphs $G_i = (V_i, E_i)$ and $G_j = (V_j, E_j)$, their product graph $G_\times = (V_\times, E_\times)$ has the vertex set $V_\times = \{(v_i, v_j) : v_i \in V_i \wedge v_j \in V_j \wedge \ell(v_i) = \ell(v_j)\}$ and the edge set $E_\times = \{((v_i, v_j), (u_i, u_j)) : \{v_i, u_i\} \in E_i \wedge \{v_j, u_j\} \in E_j\}$. The geometric random walk kernel is defined as:

$$K_\times^\infty(G_i, G_j) = \sum_{p,q=1}^{|V_\times|} \left[\sum_{l=0}^{\infty} \lambda^l (A_\times^l) \right]_{pq} = e^T (I - \lambda A_\times)^{-1} e$$

  where I is the identity matrix, e is the all-ones vector, and λ is a positive, real-valued weight. The geometric random walk kernel converges only if $\lambda < \frac{1}{\lambda_\times}$ where $\lambda_\times$ is the largest eigenvalue of $A_\times$.

- **Shortest Path Kernels**: Shortest path kernels [7] transform graphs into shortest-path graphs S, where edges represent shortest paths in the original graph G. Given graphs G_i and G_j with shortest-path graphs S_i and S_j, the kernel is defined as:

$$k(S_i, S_j) = \sum_{e_i \in E_i} \sum_{e_j \in E_j} k^{(1)}_{\text{walk}}(e_i, e_j)$$

where $k^{(1)}_{\text{walk}}(e_i, e_j)$ compares edge lengths and endpoint labels:

$$\begin{aligned} k^{(1)}_{\text{walk}}(e_i, e_j) = {} & k_v(\ell(v_i), \ell(v_j))\, k_e(\ell(e_i), \ell(e_j))\, k_v(\ell(u_i), \ell(u_j)) \\ & + k_v(\ell(v_i), \ell(u_j))\, k_e(\ell(e_i), \ell(e_j))\, k_v(\ell(u_i), \ell(v_j)) \end{aligned}$$

with k_v and k_e as vertex and edge label comparison kernels.

- **Weisfeiler-Lehman (WL) Kernels**: Weisfeiler-Lehman kernels [61] use the Weisfeiler-Lehman test of isomorphism to create graph representations that capture structural information. For graphs G and G', and a base kernel k, the Weisfeiler-Lehman kernel with h iterations is:

$$k_{WL}(G, G') = k(G_0, G'_0) + k(G_1, G'_1) + \ldots + k(G_h, G'_h)$$

where $\{G_0, G_1, \ldots, G_h\}$ and $\{G'_0, G'_1, \ldots, G'_h\}$ are the Weisfeiler-Lehman sequences of G and G'. This kernel leverages the Weisfeiler-Lehman subtree kernel for efficient graph classification.

- **Subgraph Kernels**: Subgraph kernels [41] compare the frequency of various subgraph patterns within the graphs. These patterns could be small, predefined structures like triangles, cliques, or more complex motifs. Subgraph kernels are effective in capturing specific structural properties relevant to the application domain.

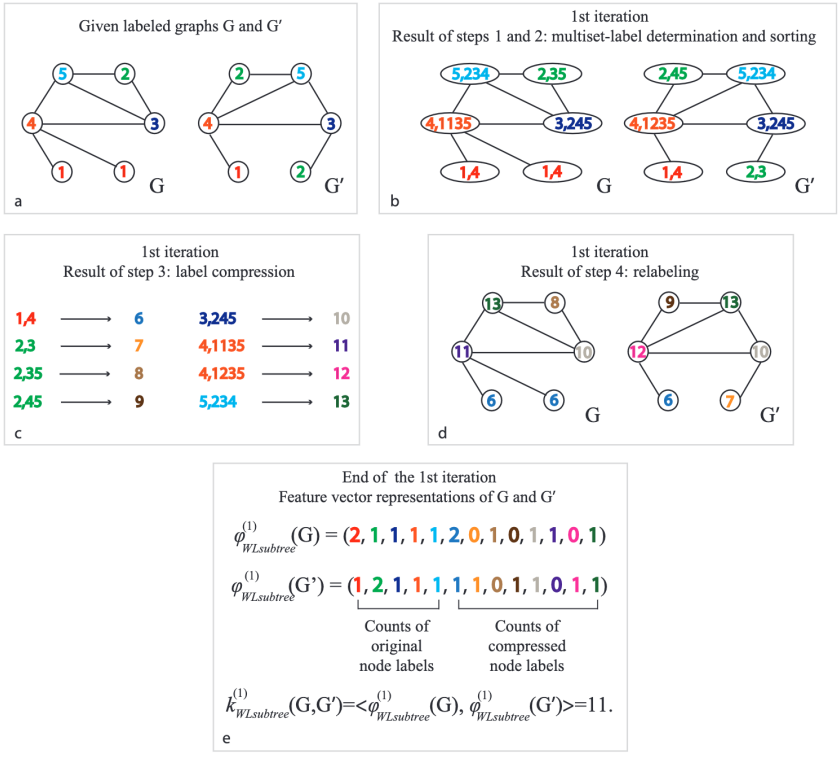


Figure 4.2.2: An illustration of the computation of the Weisfeiler-Lehman kernel for two graphs [61].

4.3 Graph Matching

Graph matching is the problem of finding correspondences between the vertices and edges of two graphs in a way that reflects their structural similarity. It is fundamental in many applications, such as pattern recognition, computer vision, bioinformatics, and social network analysis. Graph matching can be categorized based on the strictness of the correspondence criteria into exact and approximate matching.

4.3.1 Exact Graph Matching (Graph Isomorphism)

Exact graph matching, also known as graph isomorphism, is the problem of finding a one-to-one correspondence between the vertices of two graphs such that the edge connectivity is preserved. Two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are isomorphic if there exists a bijection $f : V_1 \to V_2$ such that $(u, v) \in E_1$ if and only if $(f(u), f(v)) \in E_2$. This kind of bijection is commonly described as "edge-preserving bijection", in accordance with the general notion of isomorphism being a structure-preserving bijection. If an isomorphism exists between two graphs, then the graphs are called isomorphic and denoted as $G_1 \simeq G_2$.

Formally, the problem can be stated as finding an isomorphism f that satisfies:

$$G_1 \simeq G_2 \iff \exists f : V_1 \to V_2 \text{ such that } (u, v) \in E_1 \iff (f(u), f(v)) \in E_2$$

Graph isomorphism is an equivalence relation on graphs and, as such, it partitions the class of all graphs into equivalence classes. A set of graphs isomorphic to each other is called an isomorphism class of graphs.

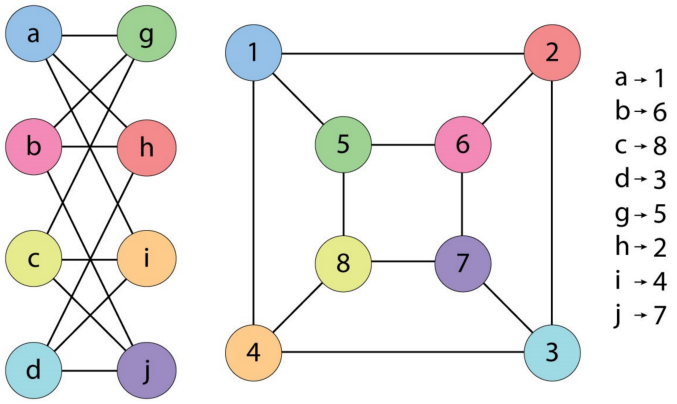


Figure 4.3.1: A visual illustration of two isomorphic graphs.

The question of whether exact graph matching can be determined in polynomial time is an unsolved problem in computer science, known as the graph isomorphism problem, and its generalization, the subgraph isomorphism problem, is known to be NP-complete.

The **Weisfeiler-Lehman (WL) Test** [20] is a heuristic test for the existence of an isomorphism between two graphs. If the test fails, the two input graphs are guaranteed to be non-isomorphic. If the test succeeds, the graphs may or may not be isomorphic. There are generalizations of the WL test algorithm that are guaranteed to detect isomorphisms, although their run time is exponential.

A well-known algorithm for graph isomorphism is the **VF2 Algorithm**, developed by Cordella et al. in 2001 [25]. The VF2 algorithm is a depth-first search algorithm that tries to build an isomorphism between two graphs incrementally and uses a set of feasibility rules to prune the search space. The VF2 algorithm has been used in a wide range of applications, such as pattern recognition, computer vision, and bioinformatics. While it has a worst-case exponential time complexity, it performs well in practice for many types of graphs.

4.3.2 Inexact Graph Matching

Inexact, or approximate, graph matching seeks to find the best possible match between two graphs when an exact match is not feasible. This situation often arises when the graphs differ in size or contain noise and imperfections. Inexact matching is particularly useful in fields like image recognition, where the data graphs generated from image segmentation typically have a different number of vertices compared to the model graphs they are matched against. In attributed graphs, even if the number of vertices and edges are identical, matching may still be inexact due to differing attributes.

Inexact graph matching methods can be broadly categorized into two groups: those based on identifying possible and impossible pairings of vertices and those that formulate the problem as an optimization task. Search-based methods focus on identifying feasible vertex pairings by using heuristics and constraints to prune the search space. Contrarily, optimization-based methods aim to find the best possible correspondence between graphs by optimizing a similarity or cost function. Graph Edit Distance is a notable example of an optimization-based method, where, as previously discussed, the objective is to minimize the number of edit operations needed to transform one graph into another.

4.4 Maximum Common Subgraph

The Maximum Common Subgraph (MCS) problem is a fundamental problem in graph theory, which aims to find the largest subgraph common to two given graphs. This problem has significant applications in various domains, such as cheminformatics, bioinformatics, and pattern recognition, where it is used to identify similar structures within different graphs.

There are two primary versions of the MCS problem: Maximum Common Induced Subgraph (MCIS) and Maximum Common Edge Subgraph (MCES). Each version has its own specific constraints and applications, which we will explore in the following subsections.

4.4.1 Maximum Common Induced Subgraph (MCIS)

The Maximum Common Induced Subgraph (MCIS) problem seeks to find the largest induced subgraph common to both input graphs. An induced subgraph is a subset of the vertices of a graph along with all the edges connecting pairs of vertices in that subset. More formally, an induced subgraph $I = (V_I, E_I)$ of a graph $G = (V_G, E_G)$ is defined such that $V_I \subseteq V_G$ and E_I consists of all edges $(u, v) \in E_G$ where $u, v \in V_I$.

Formally, given two graphs $G = (V_G, E_G)$ and $H = (V_H, E_H)$, the MCIS problem aims to find a graph $I = (V_I, E_I)$ such that:

- I is an induced subgraph of both G and H, and
- I has the maximum number of vertices among all such subgraphs.

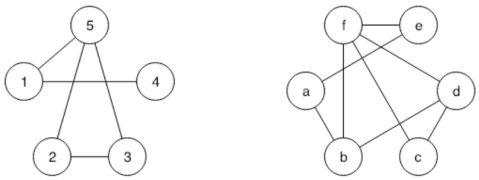
The MCIS problem is NP-hard. The associated decision problem is also NP-complete [27]: given two graphs G and H and a number k, it is computationally infeasible to determine whether G and H have a common induced subgraph with at least k vertices. Due to its complexity, the MCIS problem is also hard to approximate [34].

One approach to solve the MCIS problem is to construct a modular product graph of G and H. In this graph, the largest clique corresponds to a maximum common induced subgraph of G and H. Therefore, algorithms designed for finding maximum cliques can be adapted to find the MCIS [2]. Additionally, modified maximum-clique algorithms can be used to find maximum common connected subgraphs [52].

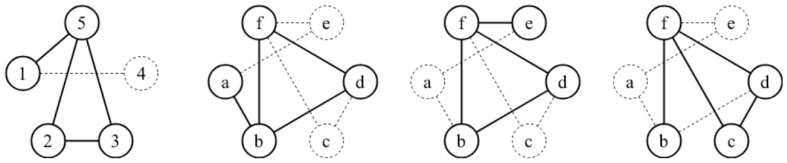
The McSplit algorithm and its variant McSplit↓ [51] are forward-checking algorithms that do not rely on the clique encoding. Instead, they use a compact data structure to keep track of the vertices in graph H to which each vertex in graph G may be mapped. These algorithms often outperform clique-based methods for many graph classes.

4.4.2 Maximum Common Edge Subgraph (MCES)

The Maximum Common Edge Subgraph (MCES) problem, on the other hand, focuses on finding the largest subgraph based on common edges rather than vertices. In this version, the goal is to maximize the number



(a) Undirected graphs G and H.



(b) Different MCS mappings for G and H.

Figure 4.4.1: Given the graphs G and H presented in Fig.(a), Fig.(b) reports 4 different possible MCSs. [50]

of edges in the subgraph that are common to both input graphs, regardless of whether all connecting vertices are included.

Given two graphs $G = (V_G, E_G)$ and $H = (V_H, E_H)$, the MCES problem aims to find a subgraph $S = (V_S, E_S)$ such that:

- S is a subgraph of both G and H, and
- S has the maximum number of edges among all such subgraphs.

The maximum common edge subgraph problem on general graphs is NP-complete, as it is a generalization of subgraph isomorphism.

The MCES problem is useful in scenarios where the relationships or interactions represented by the edges are more significant than the specific vertices involved, such as in social network analysis or biological interaction networks.

Chapter 5

Graph Neural Networks (GNN)

As established in the previous chapters, graphs are prevalent in numerous domains, including social networks, biological networks, knowledge graphs and recommendation systems. Traditional neural networks are not inherently suited for handling graph-structured data, due to their requirement for grid-like inputs with fixed size. These limitations have led to the development of Graph Neural Networks (GNNs), which are designed to directly process graph structures.

GNNs fall under the broader field of Geometric Deep Learning (GDL) [10], [9], which aims to generalize deep learning techniques to non-Euclidean domains such as graphs, grids, groups and manifolds. GDL aims to extend the capabilities of deep learning to complex geometric structures by incorporating their intrinsic properties into the learning process, allowing models to effectively learn from and generalize to graph-structured data.

Building on these principles, GNNs are designed to capture the dependencies between nodes (vertices) and edges in a graph through message passing, aggregation, and update functions. These networks iteratively aggregate information from a node's neighbors to learn node, edge, or graph-level representations, enabling tasks such as node classification, link prediction, and graph classification.

Contents

5.1 Machine Learning on Graphs 50
5.1.1 Motivation 50
5.1.2 Permutation Invariance and Equivariace 50
5.2 Graph Convolution 52
5.2.1 Spectral Graph Convolution 52
5.2.2 Spatial Graph Convolution 53
5.3 GNN Architectures 54
5.3.1 Spectral Variants 54
5.3.2 Spatial Variants 55
5.4 Graph Matching Networks 58
5.4.1 Graph Matching Network Architecture 58
5.4.2 Training Graph Matching Networks 59

5.1 Machine Learning on Graphs

5.1.1 Motivation

The motivation for the development of Graph Neural Networks (GNNs) can be attributed to the remarkable success of Convolutional Neural Networks (CNNs) [44] and the concept of applying convolutional filters to images. Images, on which CNNs operate, can be viewed as a special instance of graph-structured data, where nodes are pixels and edges represent adjacency between pixels. Convolutional filters in CNNs exploit the spatial locality of pixels by sliding rectangular kernels with a small receptive field over the image to produce feature maps, capturing spatial hierarchies and local patterns effectively.

While CNNs have demonstrated unparalleled efficiency in handling grid-like Euclidean data, many real-world data structures, such as graphs, are inherently non-Euclidean. These structures do not conform to the grid-like patterns of images and require a more generalized approach for processing and learning. GNNs emerged as a natural evolution of CNNs, generalizing the convolutional framework to graphs. This adaptation enables the extraction of meaningful features and relationships from complex networks. Specifically, GNNs define an embedding vector for each node, usually initialized with inherent node properties, which are then transformed by a sequence of learnable layers. This process involves iteratively gathering information from neighboring nodes through message passing, effectively capturing the local and global structure of the graph. However, unlike grids, graphs do not have an inherent ordering of their nodes, and the number of possible permutations increases factorially with the number of nodes. To address this, GNNs incorporate notions such as permutation invariance and equivariance in their architecture, ensuring that the learned representations are robust to the different permutations of the nodes.

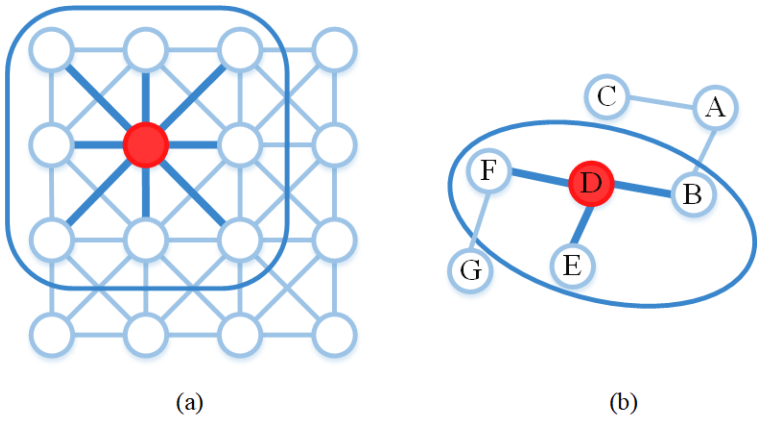


Figure 5.1.1: A comparison between a Euclidean and a Non-Euclidean space.

5.1.2 Permutation Invariance and Equivariace

As established in section 4.1, a graph with N nodes can be represented by its $N \times N$ adjacency matrix A and the nodes features can be represented by the $N \times D$ feature matrix X, where row n corresponds to the D-dimensional feature vector x_n of node n. In order to use this representation, an ordering of the nodes should be predefined. However, the graph and its properties remain the same regardless of the decided ordering of the nodes. In order to address this challenge, the neural network should learn a function which preserves the various symmetries of the graph.

A permutation matrix P specifies a particular permutation of the node ordering and has the same dimensionality with the adjacency matrix. It is a square binary matrix that has exactly one entry of 1 in each row and each column with all other entries being 0. For example, if we have a graph with 4 nodes we can define P as:

$$P = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

If we name the nodes (A, B, C, D), this permutation matrix corresponds to the transformation $(A, B, C, D) \rightarrow (B, C, D, A)$. When we perform such a node re-ordering, this results in the permutation of the rows of the feature matrix X, which can be expressed by the following matrix multiplication:

$$\tilde{X} = PX$$

This re-ordering also results in a permutation of both the rows and columns of the adjacency matrix A, which is expressed by the following matrix multiplication:

$$\tilde{A} = PAP^T$$

GNNs aim to learn a function f that is not dependent on the ordering of the nodes, which should therefore be permutation invariant.

Permutation Invariance: A function f is permutation invariant if its output remains unchanged under any permutation of its input. Formally:

$$f(PX) = f(X),$$

for any permutation matrix P. This property is important for graph-level tasks such as graph classification, where the order of nodes should not affect the final output. However, although a permutation invariant function ensures that any global property of the graph does not depend on the node ordering, it can lead to a loss of information for individual nodes. This is because it treats nodes as sets and not each one individually, potentially overlooking node-specific information.

Therefore, for node-level tasks, a re-ordering of the nodes should be reflected in the output predictions by correspondingly permuting the node features. This leads us to the notion of permutation equivariance.

Permutation Equivariance: A function f is permutation equivariant if permuting the input results in a corresponding permutation of the output. Formally:

$$f(PX) = Pf(X),$$

for any permutation matrix P.

Extending these definitions to include both the feature and adjacency matrices, we have the following definitions:

- **Permutation Invariance:** $f(PX, PAP^T) = f(X, A)$
- **Permutation Equivariance:** $f(PX, PAP^T) = Pf(X, A)$

Graph neural networks typically consist of several equivariant layers, and when predictions at the graph level are needed, a global pooling layer invariant to permutations is applied.

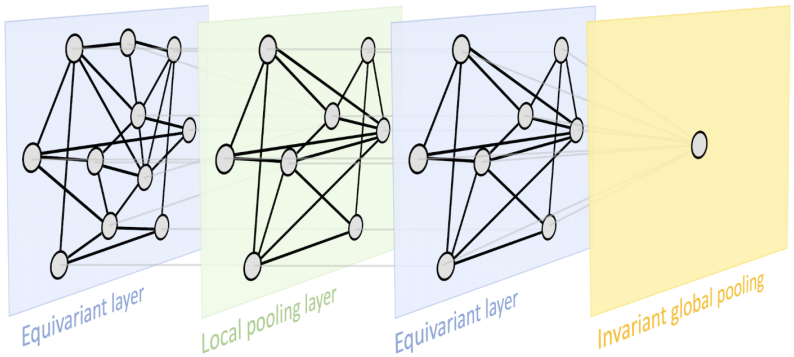


Figure 5.1.2: Geometric Deep Learning blueprint, exemplified on a graph. A GNN architecture may contain permutation equivariant layers, local pooling, and a permutation invariant global pooling layer [10].

5.2 Graph Convolution

Graph convolution is the fundamental operation that Graph Neural Networks (GNNs) employ to process and analyze graph-structured data. This operation enables GNNs to effectively learn and extract features from nodes and their local neighborhoods, leveraging the relationships inherent in the graph.

From a Graph Signal Processing (GSP) [54], [19] perspective, graph convolution can be understood as a function acting on a graph signal. A graph signal assigns values to the nodes of a graph, representing various attributes or measurements associated with these nodes. The convolution operation involves aggregating and transforming these node values, taking into account the graph's structure defined by its edges.

There are two primary approaches to defining graph convolution: spectral and spatial. Each approach offers a different perspective and set of techniques for implementing convolution on graphs, which we will analyze in this section.

5.2.1 Spectral Graph Convolution

Spectral graph convolution is one of the primary approaches to defining convolution operations on graphs, grounded in the principles of Graph Signal Processing (GSP). This approach leverages the spectral properties of graphs, particularly the eigenvalues and eigenvectors of the graph Laplacian matrix, to perform convolution in the frequency domain.

Graph Fourier Transform

The foundation of spectral graph convolution lies in the Graph Fourier Transform (GFT). The GFT generalizes the classical Fourier transform to graph-structured data. It uses the eigenvectors of the Graph Laplacian to define a frequency domain for graph signals.

The Graph Laplacian is a mathematical representation of an undirected graph and is defined as:

$$L = D - A,$$

where D is the degree matrix, which is a diagonal matrix with $D_{i,i} = \sum_j A_{ij}$, and A is the adjacency matrix. However, usually the normalized Laplacian is used. Normalizing the Laplacian makes the spectral properties of the graph less dependent on the scale of the graph. It allows for better comparison between graphs of different sizes and structures and it often leads to more interpretable spectral properties. The normalized Laplacian is defined as:

$$L = I - D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$$

The nomarlized Laplacian is real, symmetric and positive semidefinite, implying that all its eigenvalues are non-negative and it can be diagonalized. The eigen-decomposition of the Laplacian matrix L is:

$$L = U\Lambda U^T,$$

where U is the matrix of eigenvectors and Λ is the diagonal matrix of eigenvalues. The columns of U represent the orthogonal basis for the graph signal space, and the eigenvalues in Λ correspond to the frequencies of these basis components. The eigenvectors of the normalized Laplacian matrix form an orthonormal space because the Laplacian L is normalized and thus $U^TU = I$.

The Graph Fourier Transform of a graph signal $x \in \mathbb{R}^N$, which assigns a value to each node, is defined [54], [19], [6] as:

$$\hat{x} = U^T x$$

This transformation projects the graph signal x onto the orthonormal space defined by the eigenvectors of the normalized laplacian, converting the signal into the graph's frequency domain. The inverse Graph Fourier Transform is given by:

$$x = U\hat{x}$$

which reconstructs the original graph signal from its frequency components.

In the spectral domain, the convolution of a graph signal x with a filter g can be interpreted as a multiplication in the frequency domain. According to the convolution theorem, the Fourier transform of a convolution between two signals corresponds to the pointwise multiplication of their Fourier transforms. Applying this to graph signals, we define the spectral graph convolution as follows:

Given the Graph Fourier Transform, a graph signal x, and a filter $g \in \mathbb{R}^n$, the convolution operation can be expressed as:

$$x * g = U\left((U^T g) \odot (U^T x)\right)$$

where $\odot$ denotes the element-wise Hadamard product. By representing the filter in terms of the eigenvalues, $g_\theta(\Lambda) = \text{diag}(U^T g)$, the convolution can be simplified to:

$$x * g_\theta = U g_\theta U^T x$$

Here, $g_\theta = \text{diag}(\theta)$ is a diagonal matrix containing the filter's spectral coefficients. This spectral filter is critical, as different selections of g_θ lead to various implementations of spectral-based Graph Convolutional Networks (GCNs), which we will analyze in the next section.

5.2.2 Spatial Graph Convolution

Spatial graph convolution is an alternative approach to defining convolution operations on graphs, which directly utilizes the graph structure in the spatial domain. Unlike spectral methods that rely on the eigen-decomposition of the graph Laplacian, spatial methods operate on the local neighborhood of each node, making them more intuitive and often computationally more efficient.

Neighborhood Aggregation

The core idea behind spatial graph convolution is neighborhood aggregation, introduced by Kipf et al. in 2016 [38]. Each node aggregates information from its neighbors to update its feature representation. This process can be expressed as:

$$h_i^{(l+1)} = \sigma\left(\sum_{j \in \mathcal{N}(i)} \frac{1}{c_{ij}} W^{(l)} h_j^{(l)} + b^{(l)}\right)$$

where $h_i^{(l+1)}$ is the feature of node i at layer $l+1$, $\mathcal{N}(i)$ denotes the set of neighbors of node i, c_{ij} is a normalization constant (often set to $\sqrt{d_i d_j}$, where d_i and d_j are the degrees of nodes i and j, respectively), $W^{(l)}$ is a learnable weight matrix, $b^{(l)}$ is a bias term, and σ is a non-linear activation function. The normalization constant c_{ij} helps in stabilizing the training process by ensuring that the aggregated features remain appropriately scaled.

Message Passing Neural Networks (MPNNs)

Message Passing Neural Networks (MPNNs) [29] generalize the neighborhood aggregation framework by defining a more flexible and powerful approach to propagate information through the graph and are a general framework describing the family of Convolutional GNNs. The MPNN framework consists of two main phases: message passing and readout.

During the message passing phase, nodes exchange messages with their neighbors over multiple iterations. The message passing phase for node i is defined by the message function M_t and the vertex update function U_t:

$$m_i^{(t+1)} = \sum_{j \in \mathcal{N}(i)} M_t(h_i^{(t)}, h_j^{(t)}, e_{ij})$$

$$h_i^{(t+1)} = U_t(h_i^{(t)}, m_i^{(t+1)})$$

where $h_i^{(t)}$ is the hidden state of node i at time step t, e_{ij} is the feature of the edge between nodes i and j, and M_t and U_t are the learnable message and update functions, respectively.

In the readout phase, a feature vector for the entire graph is computed using a readout function R:

$$\hat{y} = R(\{h_i^{(T)} | i \in G\})$$

where $h_i^{(T)}$ is the final hidden state of node i after T iterations, and R is a readout function that aggregates the node features to produce the final output, ensuring permutation invariance. The readout function is optional and it is used when graph level predictions are required.

Most existing spatial-based ConvGNNs, which will be presented in the following section, can be viewed as variations of MPNNs by appropriately setting the message function M_t, the update function U_t, and the readout function R.

5.3 GNN Architectures

Building on the spectral and spatial approaches discussed in the previous section, in this section we will describe some of the most important GNN architectures that leverage these techniques to effectively learn from graph-structured data.

5.3.1 Spectral Variants

As explained in the previous section, spectral-based GNNs utilize the spectral properties of graphs, particularly the eigenvalues and eigenvectors of the graph Laplacian, to define convolution operations in the frequency domain.

Spectral Convolutional Neural Network

The Spectral Convolutional Neural Network (SCNN), introduced by Bruna et al. [11], is one of the earliest attempts to generalize CNNs to graph-structured data using spectral methods. SCNNs perform convolution in the spectral domain by transforming graph signals using the eigenvectors of the graph Laplacian. The spectral convolution operation is defined as:

$$x * g = U g_\theta(\Lambda) U^T x$$

where U and Λ are the eigenvectors and eigenvalues of the graph Laplacian, respectively, and $g_\theta(\Lambda)$ is the spectral filter applied in the frequency domain. The parameters of the SCNN include the learnable spectral filter coefficients θ. However, this method suffers from high computational complexity because eigendecomposition is computationally expensive, depends on the input graph's structure, and has non-spatial locality.

ChebNet

ChebNet [16] uses Chebyshev polynomials to approximate spectral filters, reducing the computational complexity associated with the eigen-decomposition of the graph Laplacian. The convolution operation in ChebNet is defined as:

$$x * g_\theta \approx \sum_{k=0}^{K} \theta_k T_k(\tilde{L}) x$$

where $T_k(\tilde{L})$ are the Chebyshev polynomials of the rescaled Laplacian $\tilde{L}$.

Graph Convolutional Network

The Graph Convolutional Network (GCN) [38] simplifies spectral graph convolutions by approximating the convolution operation with a localized first-order approximation. The propagation rule for GCN is given by:

$$H^{(l+1)} = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right)$$

where $\tilde{A} = A + I$ is the adjacency matrix with added self-connections, $\tilde{D}$ is the degree matrix of $\tilde{A}$, $H^{(l)}$ is the matrix of node features at layer l, $W^{(l)}$ is the learnable weight matrix at layer l, and σ is a non-linear activation function.

Spectral Attention Networks

Spectral Attention Networks [40] extend spectral-based GNNs by incorporating attention mechanisms in the spectral domain. These models leverage the eigenvalues and eigenvectors of the graph Laplacian to focus on important frequencies, enhancing the model's ability to capture relevant patterns in the graph data.

In Spectral Attention Networks, the attention mechanism assigns different weights to different frequency components, allowing the network to focus on the most relevant parts of the graph signal. This is achieved by learning attention coefficients in the spectral domain.

5.3.2 Spatial Variants

In this subsection we will present some of the most prevalent spatial-based GNNs, the majority of which can be derived as a variant of the MPNN.

GraphSAGE

GraphSAGE (Graph Sample and AggregatE), introduced by Hamilton et al. in 2017 [32], is designed to generate node embeddings by sampling and aggregating features from a fixed-size set of neighbors. This approach supports efficient computation on large graphs and generalization to unseen nodes, making it scalable and applicable to various real-world scenarios.

GraphSAGE's key innovation lies in its ability to perform inductive learning. Unlike transductive methods, which require retraining to incorporate new nodes, GraphSAGE can generalize its learned representations to previously unseen nodes through its sampling and aggregation strategy.

GraphSAGE employs a two-step process at each layer: sampling a fixed-size set of neighbors and then aggregating their features using a specified aggregation function (e.g. mean, LSTM, pooling). These aggregation functions are learnable, allowing the model to adapt and refine its strategy based on the graph's characteristics. This flexibility, combined with its efficient neighborhood sampling, makes GraphSAGE scalable to large graphs with millions or even billions of edges.

The inductive learning capability of GraphSAGE enables it to generalize learned representations to nodes that were not present during training. This property makes GraphSAGE highly versatile for tasks such as node classification, link prediction, and graph classification, where new nodes may frequently appear.

GAT

Graph Attention Networks (GAT), proposed by Veličković et al. in 2018 [65], incorporate an attention mechanism, first introduced in [64], to assign different weights to different neighbors. This attention mechanism enables the network to focus on the most relevant nodes for each target node, allowing for more flexible aggregation of neighborhood information.

In GAT, the attention mechanism is implemented as follows:

1. **Attention Coefficients**: For each node i and its neighbor j, the attention coefficient α_{ij} is calculated using a shared attention mechanism. This coefficient represents the importance of node j's features to node i:

$$\alpha_{ij} = \frac{\exp(\text{LeakyReLU}(a^T[Wh_i \| Wh_j]))}{\sum_{k \in \mathcal{N}(i)} \exp(\text{LeakyReLU}(a^T[Wh_i \| Wh_k]))}$$

where $\|$ denotes concatenation, W is a learnable weight matrix, a is a learnable attention vector, and LeakyReLU is the Leaky Rectified Linear Unit activation function.

2. **Neighborhood Aggregation**: The node features are then updated by aggregating the features of the neighboring nodes, weighted by the attention coefficients:

$$h_i^{(l+1)} = \sigma\left(\sum_{j \in \mathcal{N}(i)} \alpha_{ij} W h_j^{(l)}\right)$$

where σ is a non-linear activation function, and $h_i^{(l+1)}$ is the updated feature of node i.

3. **Multi-Head Attention**: To stabilize the learning process and improve the model's expressive power, GAT employs multi-head attention. Multiple attention mechanisms, or heads, are applied in parallel, and their outputs are concatenated or averaged:

$$h_i^{(l+1)} = \|_{k=1}^{K} f\left(\sum_{j \in N(i)} \alpha_{ij}^{(k)} W^{(k)} h_j^{(l)}\right)$$

$$h_i^{(l+1)} = f\left(\frac{1}{K}\sum_{k=1}^{K}\sum_{j \in N(i)} \alpha_{ij}^{(k)} W^{(k)} h_j^{(l)}\right)$$

where K is the number of attention heads, and $\alpha_{ij}^{(k)}$ and $W^{(k)}$ are the attention coefficients and weight matrices for the k-th head, respectively.

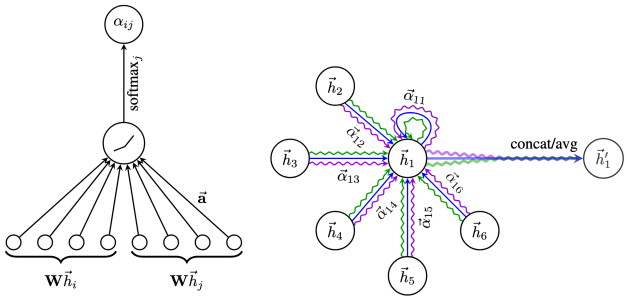


Figure 5.3.1: **Left:** The attention mechanism $\alpha(\mathbf{W}\vec{h}_i, \mathbf{W}\vec{h}_j)$ employed by GAT, parametrized by a weight vector $\vec{a} \in \mathbb{R}^{2F'}$, applying a LeakyReLU activation. **Right:** An illustration of multi-head attention ($K = 3$ heads) by node 1 on its neighborhood. Different arrow styles and colors denote independent attention computations. The aggregated features from each head are concatenated or averaged to obtain $\vec{h}'_1$ [65].

GATv2

GATv2 [8] is an extension of the original Graph Attention Network (GAT), introduced to address certain limitations and improve performance. GATv2 enhances the original attention mechanism by introducing dynamic attention coefficients that are more expressive and capable of capturing complex relationships in graph-structured data.

In the original GAT, the attention coefficients are static, meaning they are solely determined by the initial features of the nodes and remain fixed during each layer's forward pass, which can limit the flexibility and adaptability of the model. Contrarily, GATv2 introduces dynamic attention coefficients that can change during training, which allows the network to adapt the importance of neighbors throughout the learning process. The proposed function that computes the attention weights α_{ij} is:

$$\alpha_{ij} = \frac{\exp(a^T \text{LeakyReLU}([Wh_i \| Wh_j]))}{\sum_{k \in \mathcal{N}(i)} \exp(a^T \text{LeakyReLU}([Wh_i \| Wh_k]))}$$

where $\|$ denotes concatenation, W is a learnable weight matrix, a is a learnable attention vector, b_{ij} is an additional learnable parameter specific to each pair of nodes, and LeakyReLU is the Leaky Rectified Linear Unit activation function. The key difference here is that a^T is now outside the non-linearity effectively computing dynamic attention. This is much more expressive than the GAT layer with the same number of parameters.

GIN

Graph Isomorphism Network (GIN) [66] addresses the challenge of achieving the same discriminative power as the Weisfeiler-Lehman (WL) test for graph isomorphism. The goal of GIN is to map different graphs to distinct embeddings if they are determined to be non-isomorphic by the WL test. GIN is the first spatial approach designed to match the discriminative capacity of the WL test, providing strong theoretical guarantees for distinguishing between different graph structures.

The WL test iteratively updates node labels by aggregating the labels of neighboring nodes, a process that continues until the labels stabilize. If two graphs have different sets of labels at the end of this process, they are considered non-isomorphic. GIN's update rule, which is inspired by this iterative process to ensure the model's expressiveness, is as follows:

$$h_i^{(l+1)} = \text{MLP}\left((1+\epsilon) \cdot h_i^{(l)} + \sum_{j \in \mathcal{N}(i)} h_j^{(l)}\right)$$

where MLP denotes a multi-layer perceptron, ϵ is a learnable parameter or a fixed scalar, $h_i^{(l)}$ is the feature of node i at layer l, and $\mathcal{N}(i)$ represents the set of neighbors of node i.

Key aspects of GIN include:

1. **Injectivity**: GIN's sum-based aggregation ensures that the aggregation function is injective, meaning it can uniquely represent different multisets of node features.

2. **Learnable Parameter** ϵ: The parameter ϵ allows control over the relative importance of a node's own features versus its neighbors' features, which can be fixed or learned during training.

3. **Multi-Layer Perceptron (MLP)**: Using an MLP for updating node features allows GIN to perform complex, non-linear transformations on the aggregated features.

To obtain graph-level representations, GIN aggregates node embeddings from all layers of the network. This aggregation is performed by summing the node embeddings at each layer and concatenating them to form the final graph embedding:

$$h_G = \|_{k=0}^{K} \left(\text{READOUT}(\{h_v^{(k)} | v \in G\})\right)$$

where READOUT is a function that aggregates the node embeddings, and $\|$ denotes concatenation of these aggregated embeddings across different layers.

5.4 Graph Matching Networks

Graph Matching Networks (GMNs), proposed by Li et al. [46], represent a specialized approach within GNNs, designed to address the problem of graph similarity learning. Unlike traditional GNNs, which independently embed each graph into a vector space, GMNs leverage a cross-graph attention mechanism that allows for joint reasoning across graph pairs. This makes GMNs particularly effective for tasks where the relational structure between graphs plays an important role in determining similarity. GMNs are the primary GNN model we will use in our experiments in the following chapters.

5.4.1 Graph Matching Network Architecture

GMNs enhance the basic embedding models by incorporating a cross-graph attention mechanism, which aligns nodes from one graph with nodes in the other graph. The key components of the GMN architecture are outlined below:

1. **Cross-Graph Attention Mechanism**: For each node in each of the two graphs, the cross-graph attention mechanism computes an attention score with every node in the other graph:

$$a_{j\to i} = \frac{\exp(\text{sim}(h_i^{(t)}, h_j^{(t)}))}{\sum_{j'} \exp(\text{sim}(h_i^{(t)}, h_{j'}^{(t)}))}$$

   where sim is a similarity function, such as cosine similarity, and $h_i^{(t)}, h_j^{(t)}$ are the representations of nodes i and j after layer t.

2. **Matching Vector**: The cross-graph matching vector $\mu_{j\to i}$ measures how well a node in one graph can be matched to one or more nodes in the other:

$$\mu_{j\to i} = a_{j\to i} \cdot (h_i^{(t)} - h_j^{(t)})$$

3. **Node Update Module**: The node update module takes into account both the aggregated messages on the edges for each graph, as in traditional GNNs, and the cross-graph matching vector:

$$h_i^{(t+1)} = f_{\text{node}}\left(h_i^{(t)}, \sum_j m_{j\to i}, \sum_{j'} \mu_{j'\to i}\right)$$

4. **Aggregation:** After a certain number T rounds of propagations, an aggregator takes the set of node representations $\{\mathbf{h}_i^{(T)}\}$ as input, and computes a graph level representation $\mathbf{h}_G = f_G(\{\mathbf{h}_i^{(T)}\})$. The following aggregation module, proposed in [45], is used:

$$\mathbf{h}_G = \text{MLP}_G\left(\sum_{i\in V} \sigma(\text{MLP}_{\text{gate}}(\mathbf{h}_i^{(T)})) \odot \text{MLP}(\mathbf{h}_i^{(T)})\right),$$

   which transforms node representations and then uses a weighted sum with gating vectors to aggregate across nodes. The weighted sum can filter out irrelevant information, it is more powerful than a simple sum and also works significantly better empirically.

5. **Graph Representation and Similarity Score**: The aggregated graph-level representations h_{G_1} and h_{G_2} are then used to compute the similarity score.

$$s(G_1, G_2) = f_s(h_{G_1}, h_{G_2})$$

   where f_s is a similarity function applied to the graph-level embeddings.

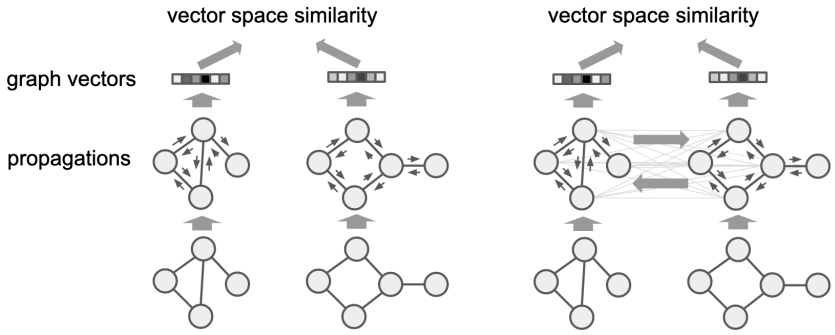


Figure 5.4.1: Illustration of the graph embedding (left) and matching models (right) [46].

5.4.2 Training Graph Matching Networks

Training GMNs involves optimizing a loss function that encourages similar graphs to have high similarity scores and dissimilar graphs to have low similarity scores. Two common loss formulations are:

1. **Pairwise Loss**: Given pairs of graphs labeled as similar ($t = 1$) or dissimilar ($t = -1$):

$$L_{\text{pair}} = \mathbb{E}_{(G_1,G_2,t)} \left[\max\{0, \gamma - t(1 - d(G_1, G_2))\}\right]$$

   where $d(G_1, G_2)$ is the distance between the graph embeddings, and γ is a margin parameter.

2. **Triplet Loss**: Given triplets of graphs (G_1, G_2, G_3) where G_1 is more similar to G_2 than to G_3:

$$L_{\text{triplet}} = \mathbb{E}_{(G_1,G_2,G_3)} \left[\max\{0, d(G_1, G_2) - d(G_1, G_3) + \gamma\}\right]$$

The authors of the paper evaluated the GMN model using the Graph Edit Distance (GED) [59], on a synthetic dataset that was generated by creating random binomial graphs G_1 with n nodes and edge probability p, using the Erdős–Rényi model [23]. Positive examples G_2 were created by substituting k_p edges from G_1 with new edges, while negative examples G_3 were created by substituting k_n edges, where $k_p < k_n$.

The model was trained to predict a higher similarity score for positive pairs (i.e., pairs with small GED) than for negative pairs, and the results were compared against the Weisfeiler-Lehman (WL) kernel [61]. Performance was measured using two metrics: pairwise AUC (Area Under the ROC Curve) for classifying pairs as similar or not, and triplet accuracy for correctly ranking the similarity of triplet sets. GMNs consistently outperformed both the WL kernel and the graph embedding models, especially with a higher number of propagation steps.

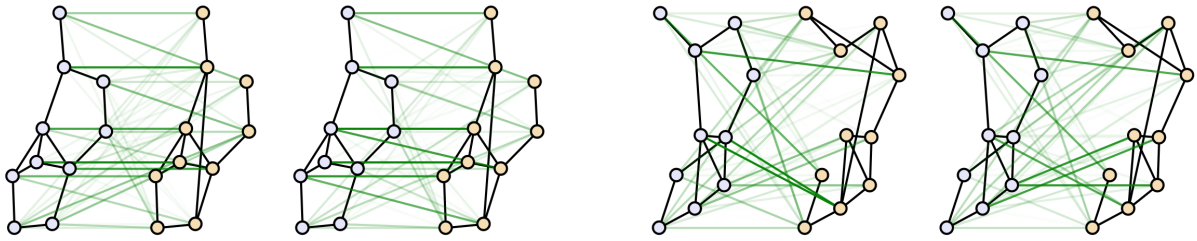


Figure 5.4.2: Visualization of cross-graph attention for GMNs after 5 propagation layers. In each pair of graphs the left figure shows the attention from left graph to the right and the right figure shows the opposite [46].

Chapter 6

Graph Summarization

Graph summarization refers to the process of creating a compact representation of a large graph while preserving essential structural properties and information. This technique offers numerous benefits, including faster runtime of algorithms, reduced storage needs, and noise reduction. It has extensive applications, such as clustering, classification, community detection, outlier detection, pattern mining, finding sources of infection in large graphs, and visualization across various domains, including social network analysis, biological networks, knowledge graphs, recommendation systems, and bioinformatics.

Graph summarization algorithms often produce either summary graphs in the form of supergraphs or sparsified graphs, or a list of independent structures. Commonly employed methods in graph summarization include graph pooling, graph clustering and graph prototyping, which will be described in the following sections.

Contents

6.1 Definitions **62**

6.2 Prior Work **62**

6.2.1 Traditional Methods 62

6.2.2 GNN-based Methods 63

6.1 Definitions

Graph Pooling

Graph pooling is a technique primarily used in the context of GNN architectures that aims to create a reduced representation of a graph by combining or selecting nodes or features. The term "pooling" originates from CNNs, where it refers to operations that downsample feature maps by aggregating information, typically through operations like max pooling or average pooling. In the context of graphs, pooling aggregates information from a set of nodes into a single representation, effectively creating a hierarchy of progressively smaller graphs.

Graph Clustering

Graph clustering aims to partition the nodes of a graph into clusters such that nodes within the same cluster are more similar to each other than to those in other clusters. This similarity can be based on various criteria, such as connectivity, shared attributes, or common roles within the graph. Recent works on graph clustering, which we will explore in the following section, are based on GNNs and utilize pooling layers as part of the overall architecture. Graph clustering is applied in social network analysis to identify groups of nodes forming distinct communities, and in biological networks to reveal functional modules within pathways.

Graph Prototyping

Graph prototyping is the process that aims to identify representative subgraphs, called prototypes, that capture the essential characteristics of different regions or aspects of a large graph. These prototypes serve as archetypes or typical examples, summarizing the key structural and functional patterns present in the original graph, that serve as compact and interpretable summaries of the data. Each prototype represents a significant and recurring substructure within the graph, and the collection of prototypes can be used to approximate the original graph's properties. Unlike graph pooling and clustering, which primarily focus on node aggregation or partitioning, graph prototyping focuses on extracting key substructures that can act as exemplars of the graph's overall structure and information.

This method is particularly useful in tasks such as graph classification, where the prototypes can serve as reference points for comparing different graphs, in anomaly detection, where deviations from the typical prototypes can indicate unusual patterns or outliers, and in the generation of new graph instances that adhere to the learned structural properties.

6.2 Prior Work

The field of graph summarization has been studied extensively over the years, with several approaches proposed for pooling, clustering, and prototype creation. In this section, we will present an overview of both the traditional methods and the more recent GNN-based methods that have emerged with the rapid advancements in deep learning.

6.2.1 Traditional Methods

Traditional methods have laid the groundwork for graph summarization by providing various techniques to simplify and interpret graph structures. This subsection highlights key clustering and prototyping strategies that have been foundational in the field.

Graph Clustering

Several traditional methods are commonly used for graph clustering:

- **Modularity-Based Clustering**: Algorithms such as the Louvain method [5] maximize modularity, a measure of the density of links inside clusters compared to links between clusters.
- **Spectral Clustering**: Spectral Clustering (SC) [49] uses the eigenvalues of the graph Laplacian matrix to perform dimensionality reduction before applying traditional clustering methods like k-means.

- **Hierarchical Clustering**: In hierarchical clustering [69], nodes are successively merged or split based on their connectivity, forming a tree of clusters (dendrogram) from which a final clustering can be derived.
- **Density-Based Clustering**: Methods like DBSCAN [24] identify clusters based on the density of nodes in the graph, making them effective for graphs with irregular cluster shapes.

Graph Prototyping

Key traditional methods for graph prototyping include:

- **Frequent Subgraph Mining**: Frequent Subgraph Mining identifies subgraphs that frequently occur within the original graph. Techniques like gSpan [67] and Frequent Subgraph Discovery (FSG) [43] are used to discover frequent substructures that can serve as prototypes representing common patterns within the graph.
- **Graph Partitioning**: Graph Partitioning divides the graph into smaller, representative subgraphs using partitioning algorithms. Algorithms like METIS and the Kernighan-Lin (KL) [36] algorithm aim to minimize edge cuts while partitioning the graph, resulting in subgraphs that can act as prototypes.

6.2.2 GNN-based Methods

Recent advancements in deep learning have led to the development of GNN-based methods for graph summarization [60], [48]. These techniques leverage the capabilities of Graph Neural Networks to capture complex patterns and relationships within graphs. In this subsection we present the latest work in graph pooling, clustering and prototyping.

Graph Pooling and Clustering

Liu et al. [47] and Grattarola et al. [30] provide comprehensive overviews of the most prominent pooling methods utilized in GNN architectures for node and graph-level tasks. These methods can be broadly categorized into global pooling and hierarchical pooling.

Global pooling techniques [21], [66], such as max pooling, average pooling, and sum pooling, aggregate the features of all nodes in the graph to create a single vector representation. For example, max pooling takes the maximum value of each feature across all nodes, while average pooling computes the mean. These methods are effective for summarizing the entire graph into a fixed-size representation, which can be used for tasks like graph classification.

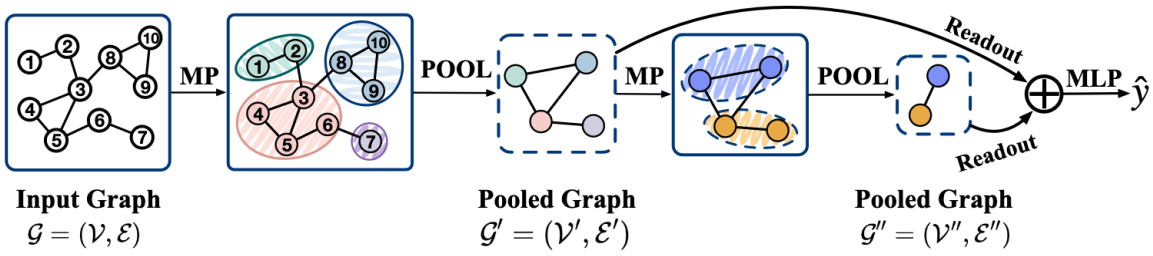


Figure 6.2.1: An illustrative example of graph pooling [47].

Contrarily, hierarchical pooling methods aim to preserve the hierarchical graph's structural information by iteratively coarsening the graph into a new graph of smaller size, allowing for multi-scale representation learning. These methods are divided into node clustering pooling, which considers graph pooling as a node clustering problem that maps nodes into a set of clusters, and node drop pooling, which uses learnable scoring functions to delete nodes with comparatively lower significance scores.

MinCutPool: One significant node clustering pooling method is MinCutPool, proposed by Bianchi et al. [4]. MinCutPool formulates a continuous relaxation of the normalized minCUT problem and trains a GNN to compute cluster assignments by optimizing this objective. The pooling layer computes soft cluster assignments, which are then used to generate a coarsened graph. This method overcomes limitations of traditional spectral clustering by being fully differentiable and not requiring the spectral decomposition of the Laplacian. MinCutPool computes soft cluster assignments as follows:

$$S = \text{softmax}(\text{MLP}(\tilde{X}, \Theta_{MLP})) \in \mathbb{R}^{N \times K}$$

where K is the number of clusters, $\tilde{X}$ is the node feature matrix generated by the Message Passing (MP) layers of the GNN and $\text{MLP}(\cdot)$ is a Multi-Layer Perceptron with trainable parameters Θ_{MLP}. MinCutPool optimizes the following unsupervised loss function:

$$\mathcal{L}_{MC} = -\frac{\text{Tr}\left(S^T \tilde{A} S\right)}{\text{Tr}\left(S^T \tilde{D} S\right)} + \left\| \frac{S^T S}{\|S^T S\|_F} - \frac{I_K}{\sqrt{K}} \right\|_F,$$

where $\|\cdot\|$ is the Frobenius norm, $\tilde{A} = D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$ and $\tilde{D}$ is the degree matrix of $\tilde{A}$. The first term, $\mathcal{L}_c$ (MinCut Loss), minimizes the Local Quadratic Variation (LQV), while the second, $\mathcal{L}_o$ (Orthogonality Loss), is a balancing term. The computational complexity of MinCut is $\mathcal{O}(N^2K + NK^2)$, which is reduced to $\mathcal{O}(EK + NK^2)$ when using sparse operations.

DMoN: Another notable node clustering pooling method is DMoN, proposed by Tsitsulin et al. [63]. DMoN optimizes a modularity-based loss composed of a local quadratic variation (LQV) term and a balancing term. The pooling layer assigns nodes to clusters, ulitizing an MLP layer, such that strongly connected components are grouped together, while the balancing term ensures clusters of similar size. The mathematical formulation of DMoN's loss function is:

$$\mathcal{L}_{DMoN} = -\frac{\text{Tr}\left(S^T \tilde{A} S\right)}{2E} + \frac{\sqrt{K}}{N} \left\| \sum_i {S_i}^T \right\|_F - 1,$$

where $\|\cdot\|$ is the Frobenius norm, $\tilde{A} = A - D^T D$ and D is the degree vector of A. As with MinCut, the computational complexity of DMoN is $\mathcal{O}(N^2K + NK^2)$, or $\mathcal{O}(EK + NK^2)$ when using sparse operations.

JustBalance: The last node clustering pooling method we will cover is JustBalance pooling, introduced by Bianchi [3]. JustBalance, which is inspired by MinCutPool, calculates the soft cluster assignments S ulitizing an MLP layer, similarly to the previous methods, but proposes a simplification in the calculation of the loss function, which consists only of a balancing term:

$$\mathcal{L}_{JB} = -\text{Tr}(\sqrt{S^T S})$$

Because the LQV term is removed in the above loss function, the computational complexity of JustBalance is $\mathcal{O}(NK^2)$, which is lower than that of the other two pooling layers for most graphs, without significantly sacrificing performance on most tasks.

TopK: In contrast, a significant node drop pooling method is TopK pooling [39], [12], [26], which is serves as a foundational method, upon which many subsequent node drop pooling methods are based. It is a pooling operator designed to select the most important nodes based on their feature representations. This method reduces the graph size while preserving significant structural information by retaining nodes with the highest scores according to a learnable projection. The projection scores for the nodes are computed as:

$$y = \sigma\left(\frac{Xp}{\|p\|}\right)$$

where X is the node feature matrix, p is a learnable parameter vector, and σ is the sigmoid function. This projection assigns a score to each node based on its features. Nodes with the top k scores are selected using:

$$i = \text{top}_k(y)$$

where top_k is an operation that selects the indices of the top k values in y, based on a user-defined pooling ratio r, such that $k = \lceil r \cdot N \rceil$, with N being the number of nodes. The node features and adjacency matrix are then updated to include only the selected nodes:

$$X' = (X \odot \tanh(y))_i$$

$$A' = A_{i,i}$$

where $\odot$ denotes element-wise multiplication and $A_{i,i}$ is the submatrix of the adjacency matrix corresponding to the selected nodes.

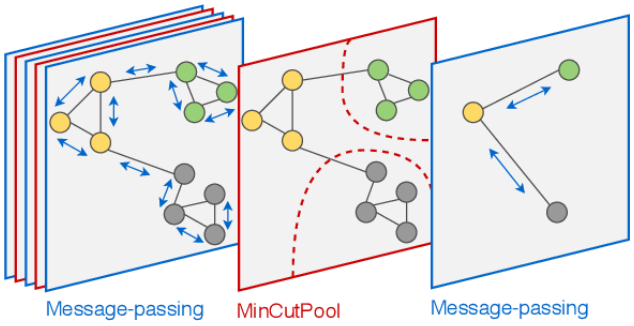


Figure 6.2.2: A deep GNN architecture where message-passing is fol- lowed by the MinCutPool layer [4].

Graph Prototyping

Recent works have explored various methodologies to achieve effective graph prototyping within the framework of Graph Neural Networks (GNNs). In this context, significant contributions have been made using graph-based techniques for counterfactual explanations (CE). Dimitriou et al. (2024) [18] propose a model-agnostic approach that uses semantic graphs to represent images and leverages GNNs for efficient Graph Edit Distance (GED) computation to retrieve counterfactuals through minimal graph edits. Similarly, Dimitriou and Chaidos et al. (2024) [17] conduct a comparative study on various graph machine learning algorithms to determine the most effective approach for generating minimal and meaningful counterfactual explanations based on graph edits. The findings and challenges addressed in these works have greatly influenced the problem we present in this thesis and the approaches we follow to solve it.

In addition to these works, several methods have been proposed that leverage the capabilities of GNNs to create representative subgraphs that capture essential patterns in the data, enhancing their interpretability. Notable approaches include:

ProtGNN: One notable approach is ProtGNN [70], which introduces a self-explaining mechanism by integrating prototype learning with GNNs to provide built-in interpretability. Unlike post-hoc explanation methods, ProtGNN uses prototypes in the latent space to make predictions based on the similarity between input graphs and learned prototypes. The architecture includes a graph encoder, a prototype layer, and a fully connected layer. The graph encoder maps the input graph to an embedding vector, and the prototype layer computes similarity scores between this embedding and predefined prototypes. The similarity function used is:

$$\text{sim}(p_k, h) = \log\left(\frac{\|p_k - h\|_2^2 + 1}{\|p_k - h\|_2^2 + \epsilon}\right)$$

The final classification is achieved using a softmax function. The learning objective combines cross-entropy loss with cluster, separation, and diversity costs to ensure meaningful and diverse prototypes. ProtGNN+ further enhances interpretability by using a conditional subgraph sampling module to identify subgraphs most similar to prototypes.

PxGNN: Building on the idea of prototype learning, PxGNN [14] focuses on learning prototype graphs that capture representative patterns of each class. PxGNN uses a prototype graph generator to create prototype graphs from learnable prototype embeddings, ensuring high-quality prototypes through self-supervision and graph reconstruction. The architecture of PxGNN includes an encoder to match test graphs with generated prototypes and optimizes a combination of classification loss and reconstruction loss to enhance the quality of the prototypes, ensuring both high prediction accuracy and reliable explanations. The overall loss function for PxGNN is given by:

$$\min_{\theta,\tilde{\mathcal{H}}} \mathcal{L}_c + \alpha \mathcal{L}_{\text{rec}} + \beta \mathcal{L}_R,$$

where $\mathcal{L}_c$ is the classification loss ensuring accurate predictions by making the graph more similar to its class prototypes, $\mathcal{L}_{\text{rec}}$ is the reconstruction loss composed of attribute and adjacency matrix reconstruction losses to ensure the quality of the generated prototype graphs, and $\mathcal{L}_R$ is a regularization term to maintain the stability and generalizability of the prototype embeddings. In this formulation, θ and $\tilde{\mathcal{H}}$ denote all model parameters and the set of learnable prototype embeddings, respectively, while α and β are hyperparameters controlling the contribution of the reconstruction loss and the regularization term.

PAGE: Another significant contribution in this area is PAGE [62], which provides model-level explanations for GNNs by discovering human-interpretable prototype graphs. PAGE's methodology involves clustering graph-level embeddings using a Gaussian Mixture Model (GMM) and selecting k-nearest neighbors to represent each cluster. The prototype discovery phase uses a prototype scoring function:

$$s(v_1, \ldots, v_k) = \mathbf{1}^\top (v_1 \odot \cdots \odot v_k)$$

to iteratively search for common subgraph patterns. The final prototype graph is the subgraph with the highest matching score among the selected candidates. PAGE effectively explains the GNN model's behavior by identifying class-distinctive subgraph patterns within the graph dataset

CPCA: Lastly, the Class Prototype Construction and Augmentation (CPCA) method [55] addresses class-incremental graph learning by constructing class prototypes to represent past data and employing prototype augmentation (PA) to create virtual classes. These prototypes mitigate catastrophic forgetting while maintaining data privacy by avoiding the need to store raw data. Prototypes are constructed as multivariate Gaussian distributions $N(\mu_i, \sigma_i^2)$, capturing the mean and covariance of class embeddings. PA generates embeddings $h_{\text{vir}_{ij}}$ of virtual classes by interpolating between existing class embeddings:

$$h_{\text{vir}_{ij}} = \lambda h_i + (1 - \lambda) h_j$$

where λ is sampled from a Beta distribution. The total loss function combines classification loss, old data loss, and knowledge distillation loss:

$$\mathcal{L} = \mathcal{L}_{\text{cls,PA}} + \alpha \mathcal{L}_{\text{old}} + \beta \mathcal{L}_{\text{kd}}$$

CPCA enhances model adaptability to new classes by enriching the embedding space with virtual classes.

Chapter 7

Proposal

In this chapter, we provide a formal definition of the problem within the field of graph summarization we aim to address, describe the architecture of the Graph Matching Network, which is trained on a graph similarity task, and propose the two methodologies we will use to create the graph summaries, utilizing the patterns that the model has learned during training. In the following sections, we first highlight the main contributions of this thesis and, after that, we present the model architecture and the proposed methodologies in detail.

7.1 Contributions

The key contributions of this thesis are summarized below:

- We employ Graph Matching Networks to address the problem of graph summarization. To our knowledge, there has been limited academic focus on utilizing GMNs to tackle this problem. Therefore, our objective is to provide a comprehensive overview of the problem we aim to address, and the methodologies we employ to solve it.
- By training the GMN on a graph similarity task, rather than explicitly on a graph summarization task, we aim to leverage the model's ability to learn important patterns and features across pairs of graphs. The learned patterns are then used to extract meaningful subgraphs, with the expectation that these subgraphs will form summaries that are highly similar for graphs within the same class. This approach can enhance the explainability of these models by highlighting nodes and patterns across the graph pairs that are most influential in computing their similarity.
- We propose and evaluate two methods for extracting graph summaries from the learned embeddings. The first method involves generating candidate summaries using TopK pooling and selecting the best based on specific criteria. The second method extends an existing Maximum Common Subgraph (MCS) algorithm by imposing additional constraints based on the cross-graph attentions to extract the summaries.
- We create a dataset of Geometric Shapes with added noise, where the ground truth summaries are known, to evaluate the proposed methods. Additionally, we use an accuracy metric that considers both exact and approximate matches to compare our methods to existing works on graph summarization.

7.2 Problem Definition

The problem we aim to address is identifying class-representative subgraphs within a multi-class graph dataset, where each graph is associated with a specific class. The primary objective is to extract a subgraph from a given graph that effectively represents the class to which the graph belongs. The extracted subgraphs for graphs within the same class should be as similar as possible, ideally being identical.

Formally, given a graph dataset $G = \{G_1, G_2, \ldots, G_N\}$, where each graph G_i is labeled with a class $c_i \in C$ (with C being the set of all classes), the goal is to develop methodologies to extract a subgraph $S_i \subseteq G_i$ for each graph G_i. The extracted subgraph S_i should be highly representative of the class c_i and, therefore, subgraphs of graphs within the same class should have high similarity between them.

While the problem shares similarities with graph prototyping in identifying representative structures, our goal is, for each graph, to find one representative subgraph that best characterizes the class the graph belongs to, instead of the multiple smaller subgraphs found by graph prototyping, which are often present in more than one classes. Additionally, the proposed approach draws elements from graph pooling methods we covered in the previous chapter, as one of the two proposed methods uses a node drop pooling operator as part of the architecture to isolate a subgraph that is representative of its corresponding class, using the node embeddings learned by the model.

7.3 Proposed Model

The GNN model we experiment with is based on the Graph Matching Network [46] architecture and consists of multiple **Graph Matching Convolution** layers, which include a cross-graph attention mechanism, followed by a **Graph Aggregator**. Each Graph Matching Convolution layer transforms node features with linear layers, batch normalization, and ReLU activation functions, while the cross-graph attention mechanism allows node feature interactions across different graphs by passing messages along edges and combining them with the original node features. Dropout is applied after the linear transformations within these layers to prevent overfitting. Finally, the Graph Aggregator condenses the node embeddings into graph-level embeddings using a gate mechanism and mean aggregation and the graph-level embeddings are used to compute the similarity, using a similarity function. This architecture explicitly determines the way the models are trained and inferenced.

Training

Following the approach outlined in Section 5.4, the GMN is trained on a pairwise graph similarity task. It receives pairs of graphs labeled as positive (similar) if they belong to the same class, or negative (dissimilar) otherwise. The model learns to predict high similarity for positive pairs and low similarity for negative pairs by minimizing the following pairwise loss:

$$L_{\text{pair}} = \mathbb{E}_{(G_1,G_2,t)} \left[\max\{0, \gamma - t(1 - d(G_1, G_2))\}\right]$$

where $d(G_1, G_2)$ is the distance between the graph embeddings, γ is a margin parameter, and t is 1 for positive pairs and -1 for negative pairs. It is important to note that the training is strictly focused on the similarity task and does not involve explicitly forming and evaluating clusters.

Inference

During inference, the model is provided with a pair of graphs that can either belong to the same or to different classes. The pair is passed through the model and after each propagation layer, the updated node embeddings and the cross-graph attentions are calculated and utilized to create the graph summaries. While the model is not explicitly trained to create these subgraphs, the expectation is that the features learned during the training on the graph similarity task will naturally lead to similar summaries for graphs of the same class. The two proposed methods for graph summary creation we experimented with are outlined in the following subsections.

7.3.1 Method I

In the first method, we aim to extract a class-representative graph summary from a pair of input graphs by leveraging the node embeddings computed at each layer during the forward pass of the model. The process involves predicting the number of important nodes that the summary should consist of, creating subgraphs using TopK pooling and, from them, selecting a final summary.

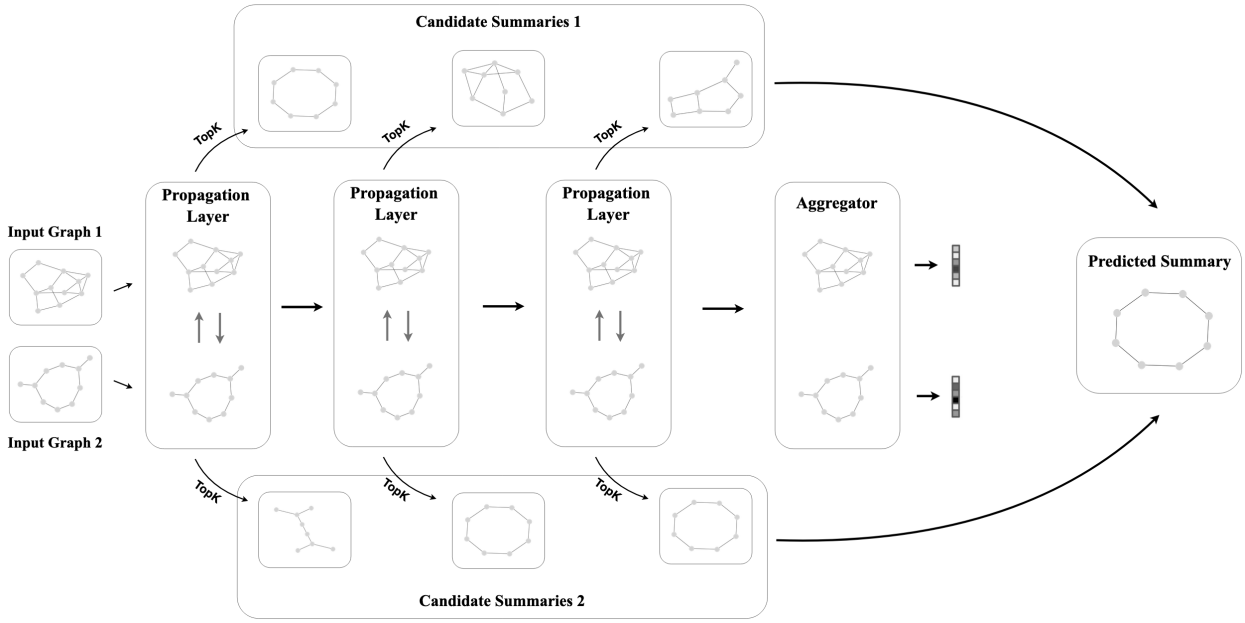


Figure 7.3.1: Overview of Method I.

The steps of the method are as follows:

1. **Forward Pass:** The pair of graphs (G_1, G_2) is passed through the GMN model with L layers. After each layer, the node embeddings are extracted. Let X_1^l denote the node embeddings of G_1 and X_2^l denote the node embeddings of G_2 after layer l.
2. **Predicting k:** The number of important nodes k is predicted using the node embeddings X_1^l and X_2^l. For each layer l:
   - For each node in G_1 and G_2, importance scores are computed using a projection weight vector w, which is initialized by drawing values from a uniform distribution within the range $[-\frac{1}{\sqrt{d}}, \frac{1}{\sqrt{d}}]$, where d is the dimension of the node features. The importance scores are given by:

$$s_{1,i}^l = \frac{X_{1,i}^l \cdot w}{\|w\|}, \quad s_{2,i}^l = \frac{X_{2,i}^l \cdot w}{\|w\|}$$

   - These scores are then normalized using the softmax function:

$$\hat{s}_{1,i}^l = \frac{e^{s_{1,i}^l}}{\sum_{j=1}^{N_1} e^{s_{1,j}^l}}, \quad \hat{s}_{2,i}^l = \frac{e^{s_{2,i}^l}}{\sum_{j=1}^{N_2} e^{s_{2,j}^l}}$$

   - The nodes are then sorted based on the normalized scores in descending order. Let $\hat{s}_{1,i}^{l,\text{sorted}}$ and $\hat{s}_{2,i}^{l,\text{sorted}}$ denote the sorted normalized scores for G_1 and G_2, respectively.
   - The cumulative sum of the sorted, normalized scores is calculated to determine k as the smallest number of nodes needed to reach a threshold τ:

$$k_1^l = \min\left\{ n \,\middle|\, \sum_{i=1}^{n} \hat{s}_{1,i}^{l,\text{sorted}} \geq \tau \right\}$$

$$k_2^l = \min\left\{ n \,\middle|\, \sum_{i=1}^{n} \hat{s}_{2,i}^{l,\text{sorted}} \geq \tau \right\}$$

     The threshold τ is a hyperparameter of the model.

After obtaining k-values for each layer for both graphs, the final k is computed as the average of all the predicted k-values:

$$k = \left\lceil \frac{1}{2L} \sum_{l=1}^{L} (k_1^l + k_2^l) \right\rceil$$

3. **Creating Subgraphs:** For each layer l, the subgraphs S_1^l and S_2^l for G_1 and G_2 are created using a TopK pooling layer with the predicted k:

$$S_1^l = \text{TopK}(X_1^l, k),$$
$$S_2^l = \text{TopK}(X_2^l, k)$$

These subgraphs act as candidates from which the final summary will be derived.

4. **Isomorphism Check:** At each layer l, the subgraphs S_1^l and S_2^l are checked for isomorphism. Based on empirical observations, if the subgraphs at a layer are isomorphic, they are often the ground truth summary. Therefore, if $S_1^l \simeq S_2^l$, the combined graph summary S^l is returned. If no isomorphic subgraphs are found at the same layer, we then look across all layers, attempting to find any subgraphs from Graph 1 that are isomorphic with any subgraphs from Graph 2: $\exists l_1, l_2$ such that $S_1^{l_1} \simeq S_2^{l_2}$. In this case, we return the most frequently occurring isomorphic subgraph.

5. **Final Summary Selection:** If no pair of isomorphic summaries is found across all layers, the most frequently occurring summary in the list of potential summaries is returned.

$$S^* = \arg \max_{S \in \{P^1, P^2, \ldots, P^L\}} \text{count}(S).$$

7.3.2 Method II

In the second method, we utilize the cross-graph attentions calculated by the GMN model after each propagation layer to construct the graph summaries.

The MCS algorithm we implemented is an extension of the McSplit algorithm [51], designed to find the maximum common induced subgraph between two graphs, with additional constraints. This extension not only considers the number of matching nodes but also the number of edges, prioritizing denser subgraphs when multiple subgraphs of the same size are found, as this approach better captures the structural relevance in certain applications [56], [22], [13]. The process involves two main steps: deriving matching pairs between the two graphs and executing the algorithm using these pairs as constraints.

Deriving Matching Pairs: The matching pairs are derived using the cross-graph attentions obtained from the GMN after each propagation layer. A pair of nodes, one from each graph, is considered a matching pair if they exhibit high attention scores towards each other in both directions. This is determined by a threshold τ, which is a hyperparemeter of the model. These matching pairs form the set MP.

Formally, let α_{ij} represent the attention score from node $i \in V_1$ to node $j \in V_2$. The matching pairs are determined as follows:

- For each node $v_i \in V_1$, nodes in V_2 that it attends to with a score above a threshold θ are identified.
- For each node $v_j \in V_2$, nodes in V_1 that it attends to with a score above θ are identified.
- A pair (v_i, v_j) is considered a matching pair if $\alpha_{ij} > \theta$ and $\alpha_{ji} > \theta$.

MCS Algorithm: Using the set of matching pairs MP across all layers as constraints, the MCS algorithm finds the maximum common induced subgraph between two graphs G_1 and G_2. The algorithm proceeds as follows:

Algorithm Maximum Common Subgraph with constraints

```
1: Procedure Search(future, M, E, mp)
2: begin
3: if |M| > |best_mapping| or (|M| == |best_mapping| and E >best_edge_count) then
4:    best_mapping ← M
5:    best_edge_count ← E
6: end if
7: bound ← |M| + Σ_{⟨G,H⟩∈future} min(|G|, |H|)
8: if bound ≤ |best_mapping| then
9:    return
10: end if
11: ⟨G, H⟩ ← SelectLabelClass(future)
12: v ← SelectVertex(G)
13: for w ∈ H do
14:    if (v, w) ∉ mp then
15:       continue
16:    end if
17:    future' ← ∅
18:    for ⟨G', H'⟩ ∈ future do
19:       G'' ← G' ∩ N(G, v) \ {v}
20:       H'' ← H' ∩ N(H, w) \ {w}
21:       if G'' ≠ ∅ and H'' ≠ ∅ then
22:          future' ← future' ∪{⟨G'', H''⟩}
23:       end if
24:       G'' ← G' ∩ N̄(G, v) \ {v}
25:       H'' ← H' ∩ N̄(H, w) \ {w}
26:       if G'' ≠ ∅ and H'' ≠ ∅ then
27:          future' ← future' ∪{⟨G'', H''⟩}
28:       end if
29:    end for
30:    Search(future', M ∪{(v, w)}, E + new_edges(M, (v, w)), mp)
31: end for
32: G' ← G \ {v}
33: future ← future \{⟨G, H⟩}
34: if G' ≠ ∅ then
35:    future ← future ∪{⟨G', H⟩}
36: end if
37: Search(future, M, E, mp)
38: end
39:
40: Procedure McSplit(G, H)
41: begin
42: best_mapping ← ∅
43: best_edge_count ← 0
44: Search({⟨V(G), V(H)⟩}, ∅, 0, mp)
45: return best_mapping
46: end
```

To avoid exploring subgraphs that have already been explored, a canonical form for each mapping is defined. This canonical form is created by relabeling the nodes of the subgraph induced by the mapping M in a consistent manner, so that isomorphic subgraphs have the same representation.

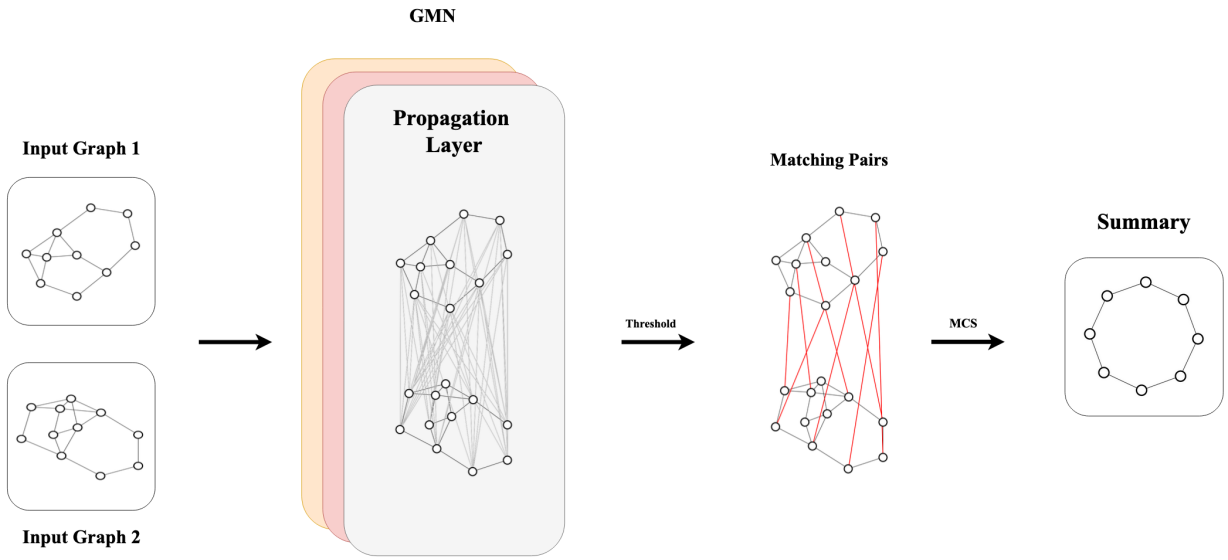


Figure 7.3.2: Overview of Method II.

Core Subgraph Extraction: To further improve the robustness of the method, we experiment with an approach to derive a final summary by identifying the maximum core subgraph from a group of predicted summaries. The maximum core subgraph is defined as the graph from the group of predicted summaries that appears most frequently as a subgraph of the other predicted summaries. In case of ties, the largest graph among the tied subgraphs is selected. Formally:

$$S^* = \arg \max_{S \in \{S^1, S^2, \ldots, S^n\}} \left(\sum_{i=1}^{n} \mathbb{I}(S \subseteq S^i) \right)$$

where $\mathbb{I}$ is the indicator function that returns 1 if S is a subgraph of S^i and 0 otherwise.

If there are multiple graphs with the same maximum subgraph count, the maximum core subgraph S^{**} is defined as:

$$S^{**} = \arg \max_{S \in \{S \mid \sum_{i=1}^{n} \mathbb{I}(S \subseteq S^i) = \sum_{i=1}^{n} \mathbb{I}(S^* \subseteq S^i)\}} |V(S)|$$

where $|V(S)|$ denotes the number of vertices in the graph S.

Chapter 8

Experiments

In this chapter, we will provide a thorough explanation of the experiments we conducted to assess the proposed model and methods, and compare them to existing approaches for graph summarization. In the first section, we will present preliminary information about the MUTAG Dataset [15], which has been extensively used in many existing works as a benchmark for evaluating methods in graph classification, and the synthetic Geometric Shapes Dataset we created that is tailored to the specific nature of our problem.

Subsequently, we will analyze the details for the training and the inference of GMN model and its hyperparameters. Finally, we will evaluate the effectiveness of the proposed methods and present the quantitative and qualitative results of our experiments.

Contents

8.1 Preliminaries 74
8.1.1 Datasets 74
8.1.2 Evaluation Metrics 75
8.1.3 Baseline Models 75
8.2 Training and Inference Details 76
8.2.1 GMN 76
8.2.2 Baseline Models 76
8.3 Results 77
8.3.1 Quantitative Results 77
8.3.2 Qualitative Results 79

8.1 Preliminaries

8.1.1 Datasets

The two datasets we will use in our experiments are MUTAG and Geometric Shapes.

Geometric Shapes

To better address the specific nature of our problem, we created a synthetic dataset of Geometric Shapes. This dataset consists of graphs representing four different geometric shapes: cycles, lines, stars, and complete graphs. To introduce variability, we add noise in the form of random nodes and edges to each graph. These edges can connect two nodes of the basic shapes, two noise nodes, or one node from each category. We experimented with two approaches for node features. In the first approach, all nodes are assigned features consisting of all 1s. In the second approach, we use Node2Vec [31] to generate node features, aiming to capture the structural context of each node within the graph. The former provided better results, so this was the approach we followed during the experiments.

This setup provides a controlled environment where we know the ground truth class-representative summary for each graph, and allows us to more accurately evaluate the effectiveness of our proposed model and methods. We created three versions of the Geometric Shapes dataset, with basic geometric shapes consisting of 8, 15, and 25 nodes, respectively. Each version contains 360 graphs, with 90 graphs per class. For the dataset with 8-node shapes, we add 2-4 noise nodes and 1-3 noise edges per added node. As the size of the basic shapes increases to 15 and 25 nodes, we slightly increase the number of added noise nodes and edges to maintain a relatively consistent proportion of noise relative to the basic shape.

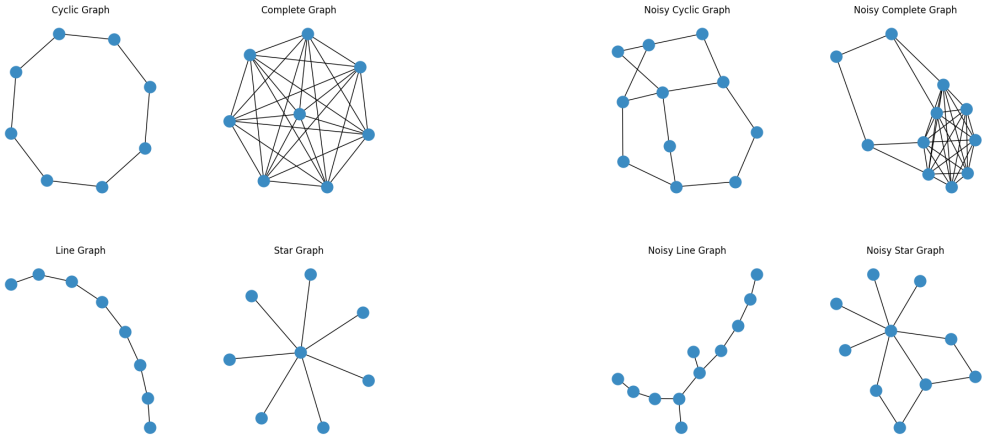


Figure 8.1.1: The four core geometric shapes in the Geometric Shapes dataset and examples of noisy graphs.

MUTAG

MUTAG [15] is a collection of nitroaromatic compounds and the goal is to predict their mutagenicity on Salmonella typhimurium. Input graphs are used to represent chemical compounds, where vertices stand for atoms and are labeled by the atom type (represented by one-hot encoding), while edges between vertices represent bonds between the corresponding atoms. It includes 188 samples of chemical compounds with 7 discrete node labels. Each graph is labeled as either mutagenic (positive class) or non-mutagenic (negative class), indicating whether the compound has a mutagenic effect on a specific organism.

The MUTAG dataset does not include ground truth class-representative subgraphs. However, it does provide ground truth prototypes, making it particularly suitable for experiments with Method II.

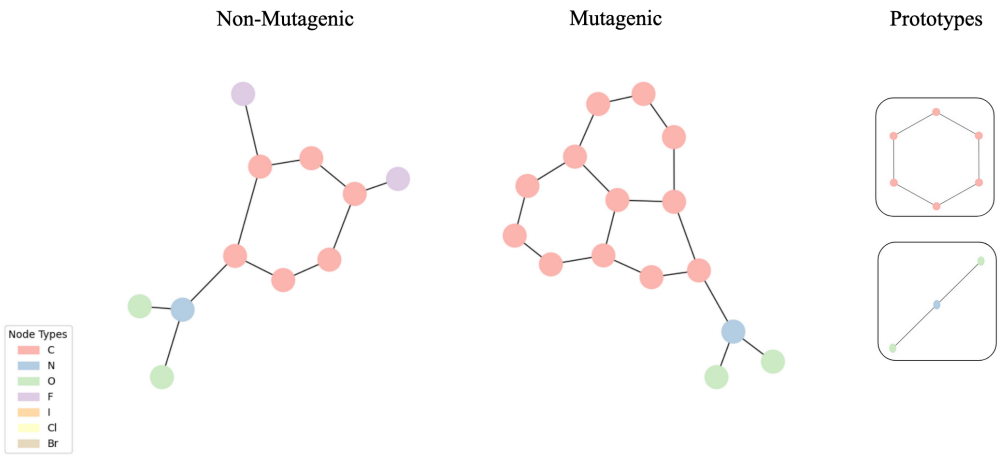


Figure 8.1.2: Examples of graphs from each of the two classes in the MUTAG dataset (non-mutagenic and mutagenic), along with the prototypes of the dataset.

8.1.2 Evaluation Metrics

Accuracy

To evaluate the effectiveness of the proposed methods, we employ an accuracy measure tailored to the problem. For each class in the dataset, a ground truth subgraph is provided. The accuracy of the model is determined by comparing the nodes selected by the model from the original graph with the nodes in the ground truth subgraph. If the selected nodes match the ground truth subgraph, the accuracy for that graph is considered to be 1; otherwise, it is 0. The overall accuracy is then computed as the average of these values across all graphs in the evaluation dataset. We consider two types of accuracy, Exact and Approximate. Exact refers to the cases where the summary produced by the model is the correct shape and has the correct number of nodes, matching the ground truth exactly, while Approximate refers to the cases where the summary is the correct shape, but a slight deviation in the number of nodes is allowed (± 1 node compared to the ground truth).

To compare our model with other summarization methods introduced in Section 6.2.2, which generate summaries by assigning labels to nodes and clustering nodes with the same label, we added labels to the nodes of the graphs in the Geometric Shapes dataset. Specifically, each node in the basic geometric shape of the graph is assigned a unique label. Each noise node is assigned the union of the labels of the nodes it is connected to. This way, the model only needs to predict for a noise node the same label as one of the basic shape nodes to which it has a path through other noise nodes. Based on these labels, we calculate the accuracy of the model. This accuracy metric is a relaxation of the traditional accuracy used in node classification, since a node is considered correctly classified if the model predicts any one of its assigned labels correctly.

8.1.3 Baseline Models

To evaluate the performance of our proposed GMN model, we compare it against several established graph summarization methods, which we have already described in Section 6.2.2. Specifically, we implemented the GNN architectures proposed in the respective papers that introduced the MinCut [4], DMoN [63], and JustBalance [3] pooling layers for graph clustering, making slight modifications to better suit our dataset.

Additionally, we compare our model against a traditional GAT model [65]. Since both graphs use attention mechanisms, this comparison allows us to assess whether the cross-graph attention-based matching mechanism proposed by Li et al. [46] provides better results compared to the self-attention mechanism used in GAT. Since the GAT model processes a single graph as input and cannot be trained on a graph similarity task, we train it on the graph classification task and evaluate its performance on graph summarization using our proposed approach described in Method I, generating candidate summaries at each layer and selecting the most frequently occurring one.

8.2 Training and Inference Details

8.2.1 GMN

Training

As outlined in Section 7.3, we train the GMN model [46] on a graph similarity task. For both datasets, we form batches of randomly selected pairs and their corresponding labels, which is 1 if the two graphs belong to the same class or -1 otherwise.

The hyperparameters that required tuning included the margin parameter γ of the pairwise loss, the dimension of the model's hidden layers, the number of layers, the learning rate, the batch size, and the total number of pairs. To optimize these hyperparameters, we considered the model's performance in both the graph similarity training task and the downstream graph summarization task.

For the margin parameter γ, the model adapted well to different values during training but performed better on the downstream task with a lower γ value. The dimension of the model's hidden layers was set to a higher value than the dimension of the node features of the dataset to allow for more expressivity. Given the small size of the dataset, we chose a higher learning rate and smaller values for the batch size. We also opted for smaller values for the number of pairs in the training split to avoid overfitting. Finally, we observed that model architectures with more propagation layers (5-7) yielded better results in the downstream task.

Specifically, for the training of the final model, the selected value for the γ parameter was 0.2, the learning rate was set to 0.01 and the batch size was set to 64. Moreover, we trained the model on 400 pairs of graphs, which were formed by randomly sampling graphs from the training split. The dimension of the model's hidden layers was set to 32 and the number of layers in the architecture was set to 7. Lastly, we used the Adam optimizer [37], with the weight decay parameter set to 10^{-5}.

Inference

During inference, a pair of graphs is provided as input to the model. As the forward pass progresses, the embeddings and cross-graph attentions are calculated after each propagation layer, and are then used to extract the summaries according to Method I and Method II. The two graphs in the pair belong to the same class.

To further understand the model's behavior, we also examine the summaries generated at each layer when the input pair consists of graphs from different classes. This analysis provides insights on how the model updates the node embeddings when the predicted similarity between the input graphs is low, and whether these updates lead to the generation of meaningful summaries at each layer.

The results of these experiments, including examples of the summaries generated for same-class graph pairs, are presented and discussed in the following section.

8.2.2 Baseline Models

Training

For the architectures proposed in the respective papers that introduced the MinCut [4], DMoN [63], and JustBalance [3] pooling layers, we trained the models using either the loss of the pooling layers alone, as originally proposed, or a combination of the pooling layer loss and a graph classification loss. The GAT model [65] was trained on a graph classification task. Given that the GAT model processes a single graph as input and cannot be directly trained on a graph similarity task, this approach allowed us to evaluate its performance in a comparable manner to our proposed GMN model.

Inference

During inference, the architectures utilizing the MinCut, DMoN, and JustBalance pooling layers were provided with a graph, which was passed through the network. At the end of the process, the cluster produced by the pooling layer was obtained as the predicted graph summary. For the GAT model, a graph was passed through the network, and, similar to the GMN, a summary was generated at each layer using TopK pooling.

The most frequent summary across all layers was then selected, following a methodology similar to Method I used for the GMN. This approach allowed us to maintain consistency in the evaluation process and ensure a fair comparison of the models' performance.

8.3 Results

8.3.1 Quantitative Results

First, we will present the Exact and Approximate Accuracy scores achieved by the GMN, using Methods I and II, and the baseline models on the Geometric Shapes dataset. Each model corresponds to the optimal model obtained after hyperparemeter tuning. The results are shown in Table 8.1.

Model	Geometric Shapes (8)	Geometric Shapes (15)	Geometric Shapes (25)
GMN (Method I)	0.543 (0.622)	0.302 (0.539)	0.316 (0.452)
GMN (Method II)	**0.671** (**0.695**)	**0.653** (**0.687**)	**0.625** (**0.641**)
GAT	0.482 (0.488)	0.429 (0.436)	0.382 (0.388)
GNN with MinCut	0.194	0.092	0.049
GNN with DMoN	0.203	0.097	0.050
GNN with JustBalance	0.231	0.103	0.052

Table 8.1: Total Exact and Approximate Accuracy scores across all classes for all the models on the Geometric Shapes dataset.

The first observation is that both the GMN and GAT models significantly outperform the clustering-based models across all three versions of the dataset. This suggests that training the GNN models by optimizing the loss corresponding to the problem that the pooling layers formulate does not directly produce summaries that closely match the ground-truth.

Comparing the GMN and GAT models, the GMN with Method II consistently achieved higher accuracy with both methods across all three versions of the dataset. To provide a more detailed understanding of the performance of these models, the accuracy metrics for each class are presented in Table 8.2 (Approximate) and Table 8.3 (Exact).

(a) Geometric Shapes (8)

Model	Cycle	Complete	Line	Star	Total
GMN (Method I)	0.607 (0.806)	**0.977** (0.977)	0.491 (0.751)	0.416 (0.816)	0.622 (0.838)
GMN (Method II)	**0.713**	0.764	**0.763**	**0.541**	**0.695**
GAT	0.573 (0.722)	0.418 (0.567)	0.457 (0.713)	0.502 (0.688)	0.488 (0.672)

(b) Geometric Shapes (15)

Model	Cycle	Complete	Line	Star	Total
GMN (Method I)	0.407 (0.452)	**0.971** (0.971)	0.285 (0.454)	0.493 (0.753)	0.539 (0.658)
GMN (Method II)	**0.682**	0.764	**0.753**	**0.516**	**0.687**
GAT	0.483 (0.612)	0.344 (0.578)	0.465 (0.632)	0.453 (0.657)	0.436 (0.620)

(c) Geometric Shapes (25)

Model	Cycle	Complete	Line	Star	Total
GMN (Method I)	0.235 (0.274)	**0.948** (0.948)	0.220 (0.293)	**0.477** (0.613)	0.452 (0.532)
GMN (Method II)	**0.662**	0.738	**0.727**	0.438	**0.641**
GAT	0.451 (0.607)	0.268 (0.564)	0.406 (0.491)	0.425 (0.694)	0.388 (0.587)

Table 8.2: Approximate Accuracy scores by class for GMN and GAT on the Geometric Shapes dataset. The values in parentheses represent the percentage of pairs where the ground truth summary appeared at least once in the subgraphs generated after each layer.

(a) Geometric Shapes (8)

Model	Cycle	Complete	Line	Star	Total
GMN (Method I)	0.607 (0.806)	0.760 (0.773)	0.474 (0.713)	0.331 (0.708)	0.543 (0.750)
GMN (Method II)	**0.713**	**0.764**	**0.721**	0.486	**0.671**
GAT	0.573 (0.722)	0.418 (0.567)	0.446 (0.656)	**0.493** (0.615)	0.482 (0.640)

(b) Geometric Shapes (15)

Model	Cycle	Complete	Line	Star	Total
GMN (Method I)	0.407 (0.452)	0.286 (0.295)	0.239 (0.425)	0.279 (0.448)	0.302 (0.405)
GMN (Method II)	**0.682**	**0.764**	**0.746**	0.421	**0.653**
GAT	0.483 (0.612)	0.344 (0.578)	0.451 (0.586)	**0.436** (0.641)	0.429 (0.604)

(c) Geometric Shapes (25)

Model	Cycle	Complete	Line	Star	Total
GMN (Method I)	0.235 (0.274)	0.583 (0.583)	0.207 (0.276)	0.237 (0.335)	0.316 (0.367)
GMN (Method II)	**0.662**	**0.738**	**0.704**	0.396	**0.625**
GAT	0.451 (0.607)	0.268 (0.564)	0.393 (0.477)	**0.418** (0.668)	0.382 (0.579)

Table 8.3: Exact Accuracy scores by class for GMN and GAT on the Geometric Shapes dataset. The values in parentheses represent the percentage of pairs where the ground truth summary appeared at least once in the subgraphs generated after each layer.

Overall, the results indicate that different models and methods exhibit varying levels of performance across classes and datasets. Among the three, the GMN with Method II was the most consistent across all datasets, achieving the highest exact accuracy scores in all classes except for the star shaped class, where it was slightly outperformed by the GAT model. Additionally, the exact and approximate accuracy scores for each class are very similar. This similarity in accuracy scores is due to the fact that when the predicted summaries are not entirely correct, particularly for star shapes, they often include noise nodes that match across the two graphs, as we will showcase with some examples in the following section. Consequently, when the predicted summary does not have the exact number of nodes, it usually fails to maintain the correct shape as well.

Method I, on the other hand, performs well in approximate accuracy, particularly for complete graphs, but struggles with exact accuracy. Although it often predicts the correct number of nodes (k), its exact accuracy is not as high as expected. Despite this, Method I still performs well when the predicted number of nodes is within ± 1 of the correct value, indicating that it effectively captures the general structure.

Lastly, the GAT model exhibits consistent performance across different classes, performing well in exact accuracy, when the correct number of nodes (k) is predicted, and often outperforming Method I. However, it performs poorly when the number of nodes is mispredicted, failing to generate meaningful summaries, as highlighted by the very similar exact and approximate accuracy scores, unlike Method I, which can still produce relatively accurate summaries even with minor deviations in k.

Classes	**Geometric Shapes (8)**			**Geometric Shapes (15)**			**Geometric Shapes (25)**		
	1	2	Both	1	2	Both	1	2	Both
Cycle (1) and Complete (2)	0.62	0.97	0.61	0.51	0.89	0.49	0.29	0.83	0.27
Cycle (1) and Line (2)	0.66	0.46	0.34	0.38	0.32	0.13	0.31	0.27	0.08
Cycle (1) and Star (2)	0.53	0.59	0.33	0.41	0.55	0.27	0.36	0.57	0.27
Complete (1) and Line (2)	0.97	0.41	0.40	0.83	0.35	0.32	0.85	0.33	0.31
Complete (1) and Star (2)	0.96	0.53	0.52	0.86	0.47	0.42	0.80	0.64	0.56
Line (1) and Star (2)	0.47	0.54	0.23	0.32	0.46	0.18	0.24	0.60	0.17

Table 8.4: GMN results with pairs of graphs from different classes on the Geometric Shapes dataset. Columns "1" and "2" show the percentage of pairs where the ground truth summary for the first and second graph, respectively, appeared at least once in the generated subgraphs. "Both" indicates the percentage of pairs where ground truth summaries for both graphs were present.

Subsequently, we examine the GMN model's behavior when the input pair consists of graphs belonging to different classes. Since these graphs belong to different classes, Methods I and II cannot be directly used to evaluate the model's accuracy. Instead, we follow a simplified approach similar to the initial steps of Method I, generating subgraphs for the two graphs after each layer l. We then count the number of pairs where the generated subgraphs match the ground-truth summaries. Specifically, we measure how often the ground-truth summary for each graph appears, as well as how often the ground-truth summaries for both graphs appear. The results are presented in Table 8.4

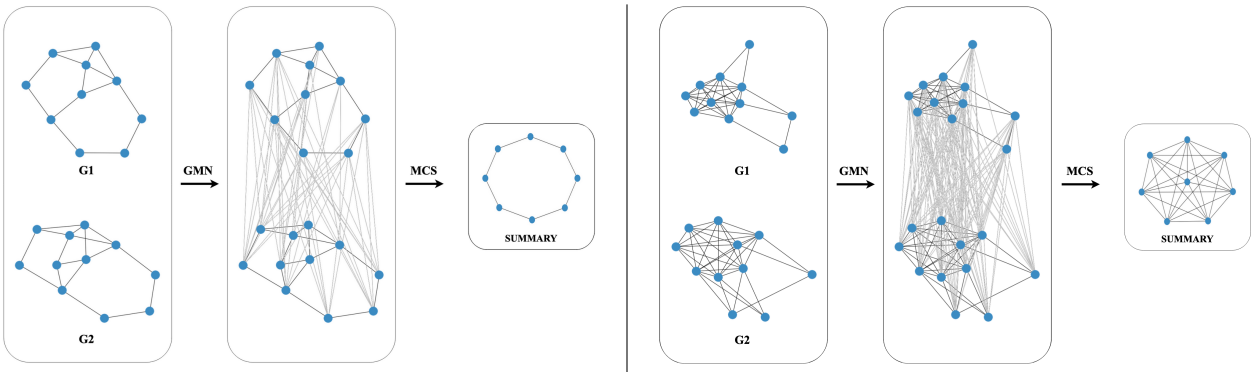
A general observation is that the accuracy metrics remain relatively consistent for each class, regardless of the class of the other graph in the pair. Additionally, these accuracy metrics are comparable to those observed when both graphs in the pair belong to the same class, as shown in Table 8.2. This consistency indicates that the model's ability to identify class-representative subgraphs is robust and not significantly affected by the presence of a graph from a different class in the pair. It effectively identifies the relevant subgraph features corresponding to each class, even in the presence of potentially distracting information from a different class.

8.3.2 Qualitative Results

Lastly, to qualitatively evaluate Method II, which has demonstrated the best performance across the evaluated datasets, we present examples of predicted summaries of graph pairs from the Geometric Shapes dataset and some results from our experiments on the MUTAG dataset.

Geometric Shapes

Starting with Geometric Shapes, we highlight cases where the method performs well by identifying the exact ground truth, instances where it also matches noise nodes across the two graphs, leading to either approximate or incorrect solutions, and cases where it finds approximate solutions by identifying a subset of the ground truth that maintains the correct shape. We also discuss scenarios where the predicted summary is incorrect. For better visual clarity, we present examples from the dataset with ground truth graphs of 8 nodes, but similar patterns and observations are also found in the dataset versions with 15 and 25 nodes.



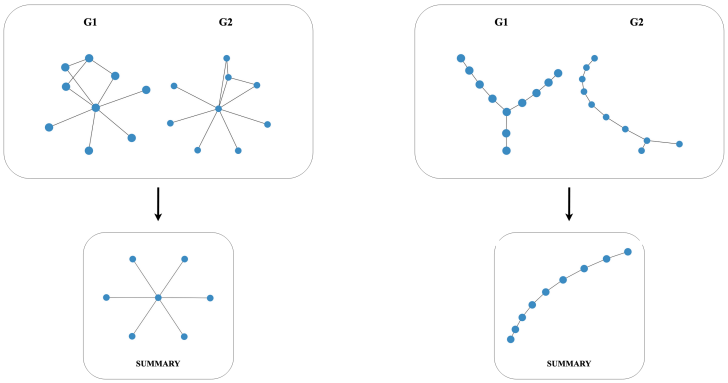
(a) Exact match example for the Cycle class.

(b) Exact match example for the Complete class.

Figure 8.3.1: Exact match examples for the Geometric Shapes (8) dataset.

In Figure 8.3.1, we present two examples where the model accurately predicts the exact ground truth summary. In the middle section, where we visualize the cross-graph attentions, we observe that the noise nodes, which are not part of the ground truth, either attend to no nodes from the other graph or attend to nodes that are not meaningful, as their neighborhoods do not match. Therefore, although the maximum common induced subgraph of the input graphs G_1 and G_2 is a superset of the ground truth, the matching pairs we derive from passing them through the GMN allow us to accurately predict the correct summary.

Subsequently, in Figure 8.3.2 we showcase two examples where the predicted summaries approximately match the ground truth, meaning they maintain the correct shape but differ in the number of nodes by ± 1. More specifically, in Figure 8.3.2a, the predicted star-shaped summary has 7 nodes, compared to the 8 of the ground-truth. This discrepancy arises because the cross-graph attentions between the noise nodes of the



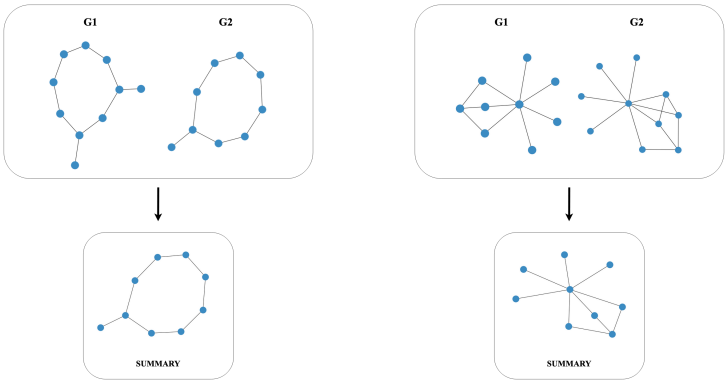
(a) Approximate match example for the Star class.

(b) Approximate match example for the Line class.

Figure 8.3.2: Approximate match examples for the Geometric Shapes (8) dataset.

two graphs were filtered out by the threshold we set, but the same filtering also affected one of the nodes connected to them in each graph, resulting in a smaller star-shaped summary. Conversely, in Figure 8.3.2b, the predicted summary has 9 nodes instead of 8. This occurred because, in addition to the cross-graph attentions between the nodes of the basic shape, the cross-graph attention between a pair of noise nodes that extended the line were also above the set threshold, resulting in the larger predicted summary.

Finally, in Figure 8.3.3, we present two examples where the predicted summaries are incorrect. Similar to Figure 8.3.2b, the cross-graph attentions between the matching noise nodes in the two graphs are above the set threshold. However, in these cases, the noise nodes do not extend the basic shape but rather disrupt it, resulting in a noisy, incorrect predicted summary.



(a) Incorrect example for the Star class.

(b) Incorrect example for the Line class.

Figure 8.3.3: Incorrect examples for the Geometric Shapes (8) dataset.

MUTAG

Following the good performance of Method II on the Geometric Shapes dataset, we conducted some experiments on MUTAG, for which Method I did not provide good results. Although MUTAG does not have ground truth summaries to compare our results to, Method II produced relatively consistent summaries that appear to effectively represent each class and capture the important information from the input graphs.

In Figure 8.3.4, we observe that for the non-mutagenic class, the predicted summaries are consistent and typically formed as a union of one instance of each of the two prototypes. Similarly, in Figure 8.3.5, the predicted summaries for the mutagenic class often exhibit similar patterns and consist of multiple instances of the two prototypes, which correctly represent the mutagenic characteristics.

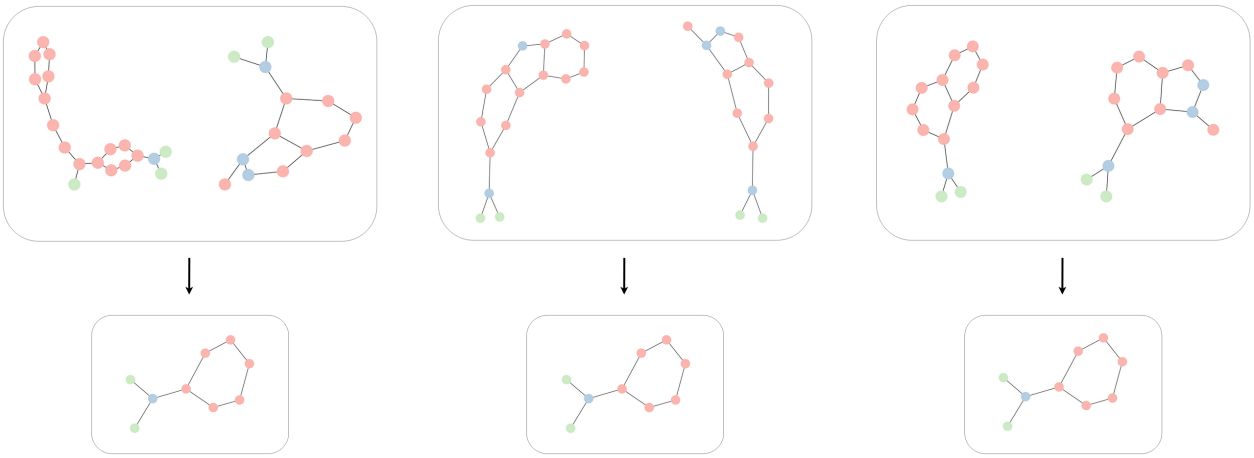


Figure 8.3.4: Accurate examples for the non-mutagenic class of the MUTAG dataset.

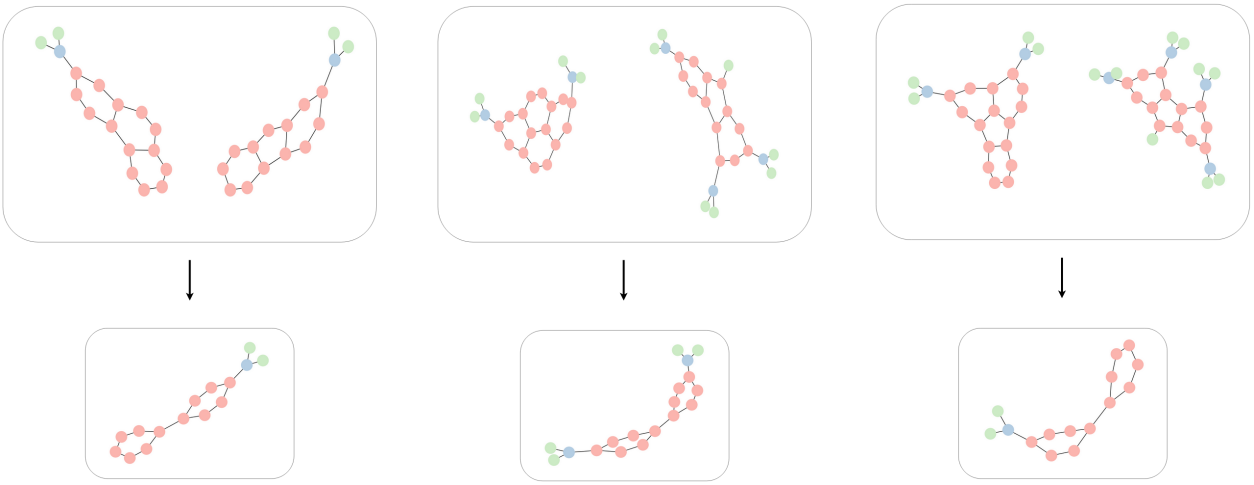


Figure 8.3.5: Accurate examples for the mutagenic class of the MUTAG dataset.

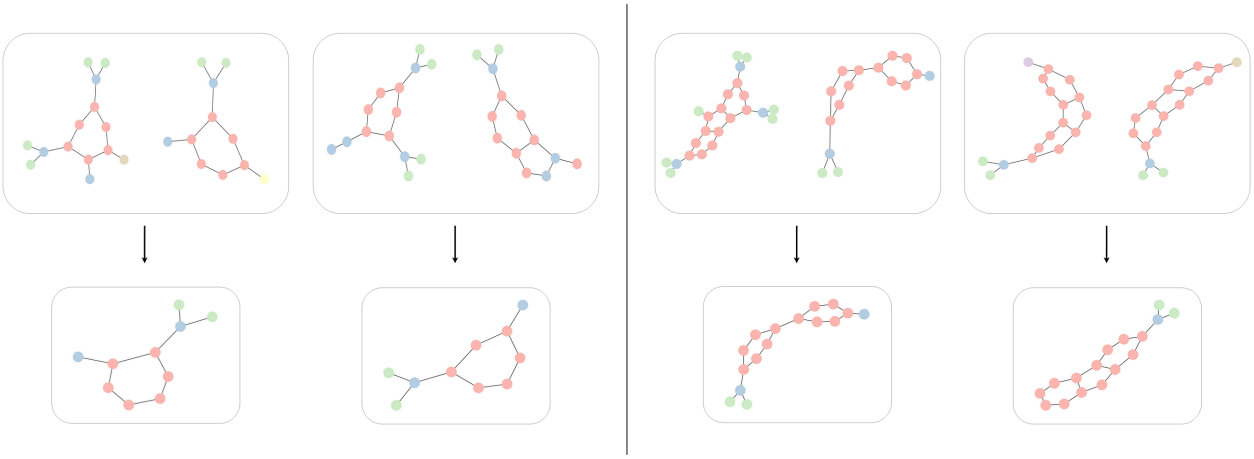


Figure 8.3.6: Incorrect examples with more nodes for the non-mutagenic (left) and mutagenic (right) classes.

However, similar to the Geometric Shapes dataset, the predicted summaries for MUTAG can also include nodes that should not be part of the summaries, due to their cross-graph attentions being above the set threshold, or omit nodes that should be included due to their cross-graph attentions being below the set threshold. These examples are presented in Figure 8.3.6 and Figure 8.3.7.

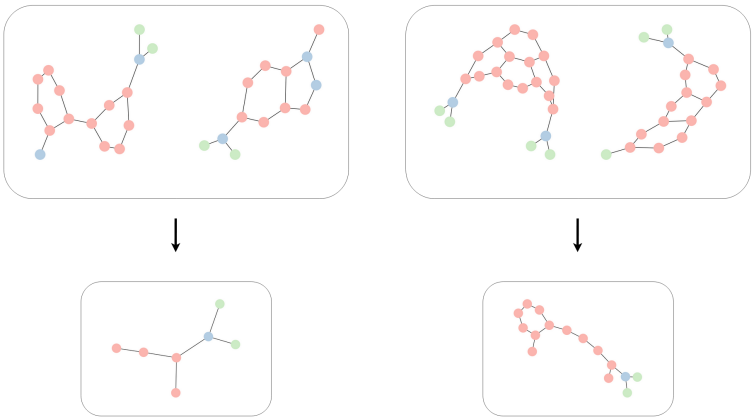
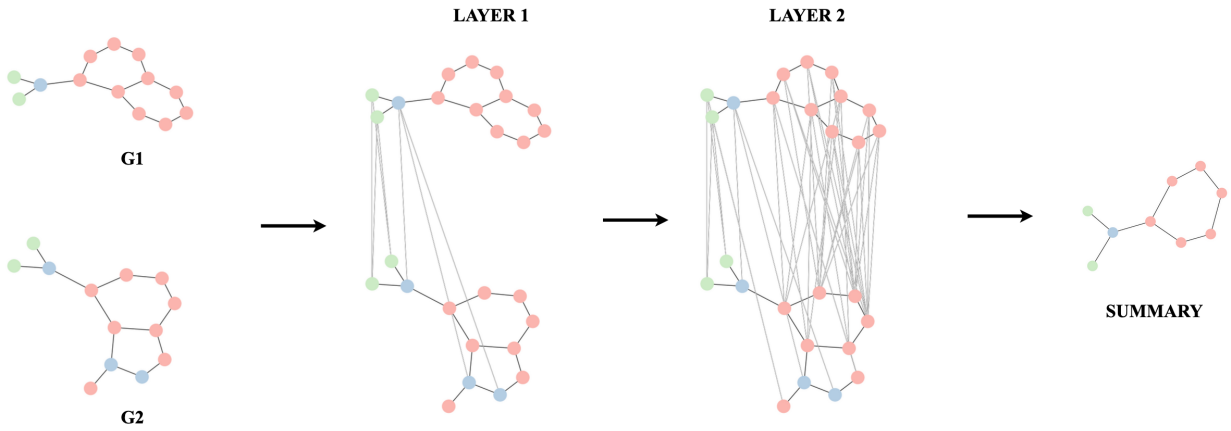


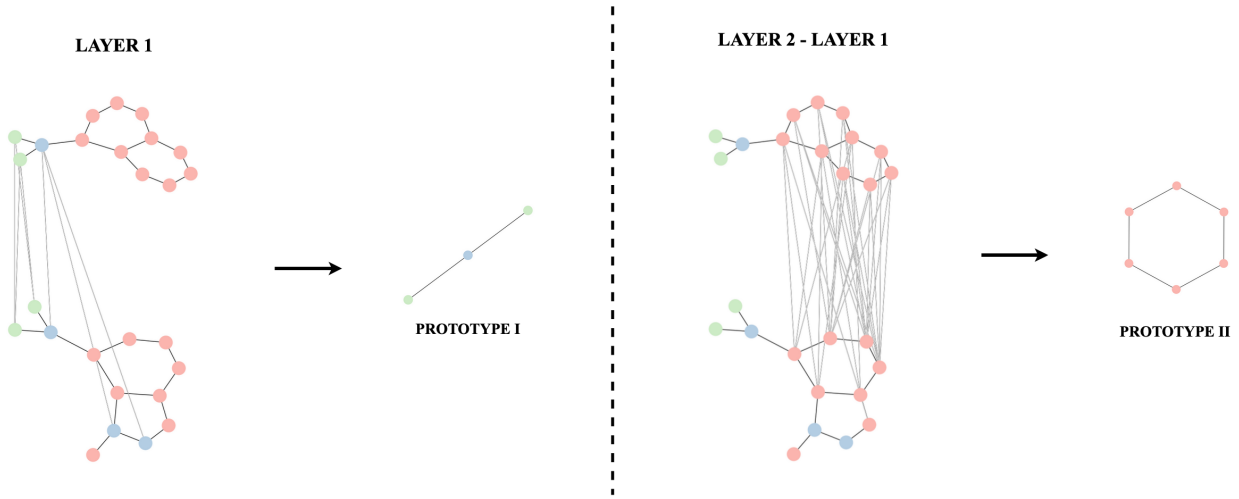
Figure 8.3.7: Incorrect examples with less nodes for the non-mutagenic (left) and mutagenic (right) classes.

Lastly, an interesting pattern was observed during the analysis, appearing more frequently in the non-mutagenic class. When creating the summary, if we first use the matching pairs from the first layer alone, the resulting summary is often one of the two prototypes, specifically a smaller, linear structure (represented by the green and blue nodes in the figures). Then, by excluding these nodes from the matching pairs of the second layer, the remaining matching pairs frequently produce the other prototype, a cyclic structure (represented by the red nodes in the figures).

This pattern was found to be relatively consistent, especially the matching pairs from the first layer resulting in the smaller, linear structure. This occurs frequently in both classes, as the initial strong matches identified by the model are typically the nodes forming this linear structure. This approach of sequentially using matching pairs from subsequent layers provides an alternative method for identifying smaller core subgraphs of the graphs in the dataset. An example of this process is demonstrated in Figure 8.3.8.



(a) Summary generated by extracting one summary at the end using all matching pairs.



(b) Summaries generated sequentially from matching pairs at each layer. The summaries match the ground truth prototypes.

Figure 8.3.8: Comparison of summary generation methods. Subfigure (a) shows the summary generated by extracting one summary at the end using all matching pairs. Subfigure (b) demonstrates the sequential generation of summaries from the matching pairs at each layer.

Chapter 9

Conclusion

9.1 Discussion

In this thesis, we addressed the problem of Graph Summarization through the lens of Graph Matching Networks. Specifically, we focused on developing methodologies to generate class-representative summaries from graph datasets by extracting subgraphs within individual graphs that effectively represent the class to which the graph belongs. To this end, we trained a Graph Matching Network (GMN) on a graph similarity task, aiming to explore the model's ability to learn important patterns across pairs of graphs during training. Subsequently, we developed two methodologies that leverage the learned embeddings to identify these patterns and form the final summaries. Method I utilizes the node embeddings of the two graphs at each layer of the architecture to create candidate summaries, using a TopK pooling layer, from which the final predicted summary is derived. Method II focuses on identifying matching pairs of nodes across the two graphs, i.e. pairs of nodes that exhibit high bidirectional cross-graph attention scores. Using these pairs as constraints, it finds the maximum common induced subgraph between the two graphs with an extension of the McSplit algorithm.

To evaluate the performance of our model and proposed methods, we created a synthetic dataset of Geometric Shapes, consisting of four basic shapes (cyclic, complete, line, and star graphs), to which we added noise in the form of random nodes and edges. This dataset provided a controlled environment with known ground truth summaries, allowing for more accurate evaluation of the proposed methods. In addition to the Geometric Shapes dataset, we conducted experiments on the real-world MUTAG dataset, which comprises mutagenic and non-mutagenic chemical compounds. Although the MUTAG dataset lacks ground truth summaries, it provides ground truth prototypes, which guided the qualitative evaluation of the summaries predicted by our model.

We compared our model against GNN architectures that utilize pooling layers, specifically DMoN, MinCut, and JustBalance, to cluster the nodes and create graph summaries. To evaluate the importance of the cross-graph attention mechanism introduced by GMNs, we also implemented a GAT architecture, which we evaluated using an approach similar to Method I. On Geometric Shapes, the GMN with Method II was very consistent and outperformed the other models in all versions of the dataset. It achieved higher accuracy scores in most classes, with few exceptions in the complete and star graph classes, where the GMN with Method I and GAT, respectively, performed better. The clustering-based GNNs performed poorly, with significantly lower accuracy scores compared to our proposed methods.

Following the strong performance of the GMN with Method II on the Geometric Shapes dataset, we conducted experiments on MUTAG. The model performed well, often identifying summaries that consisted of either one instance of the ground truth prototypes for the non-mutagenic class, or multiple instances for the mutagenic class. These results accurately reflect the nature of the two classes. However, similar to the results on Geometric Shapes, the predicted summaries sometimes included nodes that did not belong to the prototypes or omitted nodes that did, due to the cross-graph attention scores between them being higher or lower than the set threshold, respectively. Lastly, an interesting pattern observed during the analysis, particularly in

the non-mutagenic class, was that by sequentially creating summaries at each layer using the corresponding matching pairs and excluding their nodes from subsequent layers, we were able to construct the ground truth prototypes of the dataset.

In summary, the GMN, particularly with Method II, performed well and was able to accurately identify class-representative summaries. These summaries provide valuable insights into the patterns that the model recognizes and learns during training on the graph similarity task. This enhances the explainability of the model, making its behavior when predicting similarity more transparent and interpretable.

9.2 Future Work

For future work, several paths could be explored to further enhance the understanding and application of Graph Matching Networks in the field of graph summarization and beyond:

- **Improving Maximum Common Subgraph (MCS) Algorithms with GNNs:** While identifying maximum common subgraphs (MCS) using cross-graph attentions as constraints yielded good results in our experiments, the problem of computing the MCS is known to be NP-hard [27], significantly increasing the computational cost for larger graphs. Exploring approaches for MCS detection using GNNs, such as GLSearch [1], is a promising avenue to improve the efficiency of our methods.
- **Further Experimentation on Real-World Datasets:** Conducting experiments on additional real-world datasets, such as BA-Shape [68] and Benzene [58], can help further evaluate the proposed methods, providing deeper insights into the behavior of the model and the patterns it is able to learn in more complex and diverse graph structures.
- **Exploring GMN Variants that Enable Unsupervised Training Using Contrastive Learning:** Investigating contrastive GMN variants, such as CGMN [33], could enable the application of our methods to unlabeled datasets, leveraging contrastive learning techniques to enhance the model's ability to learn useful representations without requiring extensive labeled data.

By exploring these future directions, we can continue to advance the field of graph summarization, improving the efficiency, accuracy, and applicability of summarization techniques across various domains, and enhancing the explainability and interpretability of GNN models.

Chapter 10

Bibliography

[1] Bai, Y. et al. *GLSearch: Maximum Common Subgraph Detection via Learning to Search*. 2021. arXiv: 2002.03129 [cs.LG]. URL:

[2] Barrow, H. G. and Burstall, R. M. "Subgraph Isomorphism, Matching Relational Structures and Maximal Cliques". In: *Inf. Process. Lett.* 4 (1976), pp. 83–84. URL:

[3] Bianchi, F. M. "Simplifying Clustering with Graph Neural Networks". In: *Proceedings of the Northern Lights Deep Learning Workshop* 4 (Jan. 2023). ISSN: 2703-6928. DOI: 10.7557/18.6790. URL:

[4] Bianchi, F. M., Grattarola, D., and Alippi, C. *Spectral Clustering with Graph Neural Networks for Graph Pooling*. 2020. arXiv: 1907.00481 [cs.LG]. URL:

[5] Blondel, V. D. et al. "Fast unfolding of communities in large networks". In: *Journal of Statistical Mechanics: Theory and Experiment* 2008.10 (Oct. 2008), P10008. ISSN: 1742-5468. DOI: 10.1088/1742-5468/2008/10/p10008. URL:

[6] Bo, D. et al. *A Survey on Spectral Graph Neural Networks*. 2023. arXiv: 2302.05631 [cs.LG]. URL:

[7] Borgwardt, K. M. and Kriegel, H.-P. "Shortest-path kernels on graphs". In: *Fifth IEEE International Conference on Data Mining (ICDM'05)* (2005), 8 pp.-. URL:

[8] Brody, S., Alon, U., and Yahav, E. *How Attentive are Graph Attention Networks?* 2022. arXiv: 2105.14491 [cs.LG]. URL:

[9] Bronstein, M. M. et al. "Geometric Deep Learning: Going beyond Euclidean data". In: *IEEE Signal Processing Magazine* 34.4 (July 2017), pp. 18–42. ISSN: 1558-0792. DOI: 10.1109/msp.2017.2693418. URL:

[10] Bronstein, M. M. et al. *Geometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges*. 2021. arXiv: 2104.13478 [cs.LG]. URL:

[11] Bruna, J. et al. *Spectral Networks and Locally Connected Networks on Graphs*. 2014. arXiv: 1312.6203 [cs.LG]. URL:

[12] Cangea, C. et al. *Towards Sparse Hierarchical Graph Classifiers*. 2018. arXiv: 1811.01287 [stat.ML]. URL:

[13] Conte, D., Foggia, P., and Vento, M. "Challenging Complexity of Maximum Common Subgraph Detection Algorithms: A Performance Analysis of Three Algorithms on a Wide Database of Graphs". In: *Journal of Graph Algorithms and Applications* 11.1 (Jan. 2007), pp. 99–143. DOI: 10.7155/jgaa.00139. URL:

[14] Dai, E. and Wang, S. *Towards Prototype-Based Self-Explainable Graph Neural Network*. 2022. arXiv: 2210.01974 [cs.LG]. URL:

[15] Debnath, A. K. et al. "Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. Correlation with molecular orbital energies and hydrophobicity." In: *Journal of medicinal chemistry* 34 2 (1991), pp. 786–97. URL:

[16] Defferrard, M., Bresson, X., and Vandergheynst, P. *Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering*. 2017. arXiv: 1606.09375 [cs.LG]. URL:

[17] Dimitriou, A. et al. *Graph Edits for Counterfactual Explanations: A comparative study*. 2024. arXiv: 2401.11609 [cs.LG]. URL:

[18] Dimitriou, A. et al. *Structure Your Data: Towards Semantic Graph Counterfactuals*. 2024. arXiv: 2403.06514 [cs.CV]. URL:
[19] Dong, X. et al. "Graph Signal Processing for Machine Learning: A Review and New Perspectives". In: *IEEE Signal Processing Magazine* 37.6 (Nov. 2020), pp. 117–127. ISSN: 1558-0792. DOI: 10.1109/msp.2020.3014591. URL:
[20] Douglas, B. L. *The Weisfeiler-Lehman Method and Graph Isomorphism Testing*. 2011. arXiv: 1101.5211 [math.CO]. URL:
[21] Duvenaud, D. et al. *Convolutional Networks on Graphs for Learning Molecular Fingerprints*. 2015. arXiv: 1509.09292 [cs.LG]. URL:
[22] Ehrlich, H.-C. and Rarey, M. "Maximum common subgraph isomorphism algorithms and their applications in molecular science: a review". In: *WIREs Computational Molecular Science* 1.1 (2011), pp. 68–79. DOI: https://doi.org/10.1002/wcms.5. eprint: URL:
[23] Erdös, P. and Rényi, A. "On Random Graphs I". In: *Publicationes Mathematicae Debrecen* 6 (1959), p. 290.
[24] Ester, M. et al. "A density-based algorithm for discovering clusters in large spatial databases with noise". In: *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*. KDD'96. Portland, Oregon: AAAI Press, 1996, pp. 226–231.
[25] Foggia, P., Sansone, C., and Vento, M. "An Improved Algorithm for Matching Large Graphs". In: Jan. 2001.
[26] Gao, H. and Ji, S. *Graph U-Nets*. 2019. arXiv: 1905.05178 [cs.LG]. URL:
[27] Garey, M. R. and Johnson, D. S. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. USA: W. H. Freeman & Co., 1990. ISBN: 0716710455.
[28] Gärtner, T., Flach, P. A., and Wrobel, S. "On Graph Kernels: Hardness Results and Efficient Alternatives". In: *Annual Conference Computational Learning Theory*. 2003. URL:
[29] Gilmer, J. et al. *Neural Message Passing for Quantum Chemistry*. 2017. arXiv: 1704.01212 [cs.LG]. URL:
[30] Grattarola, D. et al. "Understanding Pooling in Graph Neural Networks". In: *IEEE Transactions on Neural Networks and Learning Systems* 35.2 (Feb. 2024), pp. 2708–2718. ISSN: 2162-2388. DOI: 10.1109/tnnls.2022.3190922. URL:
[31] Grover, A. and Leskovec, J. *node2vec: Scalable Feature Learning for Networks*. 2016. arXiv: 1607.00653 [cs.SI]. URL:
[32] Hamilton, W. L., Ying, R., and Leskovec, J. *Inductive Representation Learning on Large Graphs*. 2018. arXiv: 1706.02216 [cs.SI]. URL:
[33] Jin, D. et al. *CGMN: A Contrastive Graph Matching Network for Self-Supervised Graph Similarity Learning*. 2022. arXiv: 2205.15083 [cs.LG]. URL:
[34] Kann, V. "On the Approximability of the Maximum Common Subgraph Problem." In: Feb. 1992, pp. 377–388. ISBN: 978-3-540-55210-9. DOI: 10.1007/3-540-55210-3_198.
[35] Karp, R. M., Vazirani, U. V., and Vazirani, V. V. "An optimal algorithm for on-line bipartite matching". In: *Symposium on the Theory of Computing*. 1990. URL:
[36] Kernighan, B. W. and Lin, S. "An efficient heuristic procedure for partitioning graphs". In: *The Bell System Technical Journal* 49.2 (1970), pp. 291–307. DOI: 10.1002/j.1538-7305.1970.tb01770.x.
[37] Kingma, D. P. and Ba, J. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG]. URL:
[38] Kipf, T. N. and Welling, M. *Semi-Supervised Classification with Graph Convolutional Networks*. 2017. arXiv: 1609.02907 [cs.LG]. URL:
[39] Knyazev, B., Taylor, G. W., and Amer, M. R. *Understanding Attention and Generalization in Graph Neural Networks*. 2019. arXiv: 1905.02850 [cs.LG]. URL:
[40] Kreuzer, D. et al. *Rethinking Graph Transformers with Spectral Attention*. 2021. arXiv: 2106.03893 [cs.LG]. URL:
[41] Kriege, N. and Mutzel, P. *Subgraph Matching Kernels for Attributed Graphs*. 2012. arXiv: 1206.6483 [cs.LG]. URL:
[42] Kuhn, H. W. "The Hungarian method for the assignment problem". In: *Naval Research Logistics (NRL)* 52 (1955). URL:
[43] Kuramochi, M. and Karypis, G. "Frequent subgraph discovery". In: *Proceedings 2001 IEEE International Conference on Data Mining*. 2001, pp. 313–320. DOI: 10.1109/ICDM.2001.989534.

[44] LeCun, Y. et al. "Object Recognition with Gradient-Based Learning". In: *Shape, Contour and Grouping in Computer Vision*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 319–345. ISBN: 978-3-540-46805-9. DOI: 10.1007/3-540-46805-6_19. URL:

[45] Li, Y. et al. *Gated Graph Sequence Neural Networks*. 2017. arXiv: 1511.05493 [cs.LG]. URL:

[46] Li, Y. et al. *Graph Matching Networks for Learning the Similarity of Graph Structured Objects*. 2019. arXiv: 1904.12787 [cs.LG]. URL:

[47] Liu, C. et al. *Graph Pooling for Graph Neural Networks: Progress, Challenges, and Opportunities*. 2023. arXiv: 2204.07321 [cs.LG]. URL:

[48] Liu, Y. et al. *Graph Summarization Methods and Applications: A Survey*. 2018. arXiv: 1612.04883 [cs.IR]. URL:

[49] Luxburg, U. von. *A Tutorial on Spectral Clustering*. 2007. arXiv: 0711.0189 [cs.DS]. URL:

[50] Marcelli, A., Quer, S., and Squillero, G. "The Maximum Common Subgraph Problem: A Portfolio Approach". In: *ArXiv* abs/1908.06418 (2019). URL:

[51] McCreesh, C., Prosser, P., and Trimble, J. "A Partitioning Algorithm for Maximum Common Subgraph Problems". In: *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*. 2017, pp. 712–719. DOI: 10.24963/ijcai.2017/99. URL:

[52] McCreesh, C. et al. "Clique and Constraint Models for Maximum Common (Connected) Subgraph Problems". In: *International Conference on Principles and Practice of Constraint Programming*. 2016. URL:

[53] Neuhaus, M., Riesen, K., and Bunke, H. "Fast Suboptimal Algorithms for the Computation of Graph Edit Distance". In: *Structural, Syntactic, and Statistical Pattern Recognition*. Ed. by D.-Y. Yeung et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 163–172. ISBN: 978-3-540-37241-7.

[54] Ortega, A. et al. "Graph Signal Processing: Overview, Challenges, and Applications". In: *Proceedings of the IEEE* 106.5 (2018), pp. 808–828. DOI: 10.1109/JPROC.2018.2820126.

[55] Ren, Y. et al. "Incremental Graph Classification by Class Prototype Construction and Augmentation". In: *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management*. CIKM '23. Birmingham, United Kingdom: Association for Computing Machinery, 2023, pp. 2136–2145. ISBN: 9798400701245. DOI: 10.1145/3583780.3614932. URL:

[56] Rose, W. et al. "Promoting Access to White Rose Research Papers Maximum Common Subgraph Isomorphism Algorithms for the Matching of Chemical Structures". In: URL:

[57] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. "Learning representations by back-propagating errors". In: *Nature* 323 (1986), pp. 533–536. URL:

[58] Sanchez-Lengeling, B. et al. "Evaluating Attribution for Graph Neural Networks". In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle et al. Vol. 33. Curran Associates, Inc., 2020, pp. 5898–5910. URL:

[59] Sanfeliu, A. and Fu, K.-S. "A distance measure between attributed relational graphs for pattern recognition". In: *IEEE Transactions on Systems, Man, and Cybernetics* SMC-13.3 (1983), pp. 353–362. DOI: 10.1109/TSMC.1983.6313167.

[60] Shabani, N. et al. *A Comprehensive Survey on Graph Summarization with Graph Neural Networks*. 2024. arXiv: 2302.06114 [cs.LG]. URL:

[61] Shervashidze, N. et al. "Weisfeiler-Lehman Graph Kernels". In: *J. Mach. Learn. Res.* 12 (2011), pp. 2539–2561. URL:

[62] Shin, Y.-M., Kim, S.-W., and Shin, W.-Y. *PAGE: Prototype-Based Model-Level Explanations for Graph Neural Networks*. 2024. arXiv: 2210.17159 [cs.LG]. URL:

[63] Tsitsulin, A. et al. *Graph Clustering with Graph Neural Networks*. 2023. arXiv: 2006.16904 [cs.LG]. URL:

[64] Vaswani, A. et al. *Attention Is All You Need*. 2017. arXiv: 1706.03762 [cs.CL].

[65] Veličković, P. et al. *Graph Attention Networks*. 2018. arXiv: 1710.10903 [stat.ML]. URL:

[66] Xu, K. et al. *How Powerful are Graph Neural Networks?* 2019. arXiv: 1810.00826 [cs.LG]. URL:

[67] Yan, X. and Han, J. "gSpan: graph-based substructure pattern mining". In: *2002 IEEE International Conference on Data Mining, 2002. Proceedings*. 2002, pp. 721–724. DOI: 10.1109/ICDM.2002.1184038.

[68] Ying, R. et al. *GNNExplainer: Generating Explanations for Graph Neural Networks*. 2019. arXiv: 1903.03894 [cs.LG]. URL:

[69] Zepeda-Mendoza, M. L. and Resendis-Antonio, O. “Hierarchical Agglomerative Clustering”. In: *Encyclopedia of Systems Biology*. Ed. by W. Dubitzky et al. New York, NY: Springer New York, 2013, pp. 886–887. ISBN: 978-1-4419-9863-7. DOI: 10.1007/978-1-4419-9863-7_1371. URL:

[70] Zhang, Z. et al. *ProtGNN: Towards Self-Explaining Graph Neural Networks*. 2021. arXiv: 2112.00911 [cs.LG]. URL: