
Skolkovo Institute of Science and Technology

# Combinatorial and Neural Graph Vector Representations

*Doctoral Thesis*

by

SERGEY IVANOV

DOCTORAL PROGRAM IN
COMPUTATIONAL AND DATA SCIENCE
AND ENGINEERING

Supervisor
Associate Professor Evgeny Burnaev

Moscow - 2019

# Abstract

For centuries network science has been relentlessly providing humankind with the hard challenges and a new leap forward towards improving our abilities in working with graphs has recently been made in the intersection of machine learning, computer science and graph theory. In this domain of *computational graph theory* new models based on neural networks showed the dominance in a number of graph-related tasks in the presence of large amounts of data. However in the rush for better metrics in experiments our understanding why some architectures or approaches work is often lost, which comes at the cost of uncertainty in the generalization and robustness on the new tasks or the new data.

In this light, representation learning underlies model construction phase and allows us to leverage the knowledge of the topological properties of the network in order to predict the behavior of representations in downstream tasks. *Combinatorial* methods to obtain vector representations of networks have been a common element in graph kernel literature, while *neural* representations have appeared recently as a building block of neural networks on graphs. In this thesis, we argue that these two paradigms complement each other; while combinatorial embeddings have richer interpretability of the models, neural embeddings come with enhanced expressivity and trainability; and that in often cases it is possible to benefit from both.

Broadly this dissertation has two main parts that contribute to the computational graph theory. In the first we propose a new type of combinatorial and neural embeddings based on introduced graph substructure, *anonymous walks*. In the combinatorial case the embeddings are guaranteed to possess invariance property, which is novel to the domain of representation learning on graphs. For the neural embeddings we utilize machine learning models to discover relationships between different anonymous walks, which becomes practical in several applications. In the second part, we discuss new hard problems on graphs. In particular, we deal with the problems of recommendation of products or users in networks and show the complexity and approximability of these problems. We propose two complementary algorithms for these problems, based on

the greedy estimation of the value for recommendation that has limitation with the running time, and one based on the learned embeddings to augment the set of good recommendations found by the greedy algorithm.

# Publications

1. Ivanov S, Karras P. Harvester: Influence optimization in symmetric interaction networks. In2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA) pp. 61-70. IEEE, 2016.

2. Ivanov S, Theocharidis K, Terrovitis M, and Karras P. "Content recommendation for viral social influence." In Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 565-574. ACM, 2017.

3. Ivanov S, and Burnaev E. "Anonymous Walk Embeddings." In International Conference on Machine Learning, pp. 2191-2200, 2018.

4. Sharaev M, Artemov A, Kondrateva E, Ivanov S, Sushchinskaya S, Bernstein A, Cichocki A, Burnaev E. Learning connectivity patterns via graph kernels for fmri-based depression diagnostics. In 2018 IEEE International Conference on Data Mining Workshops (ICDMW) pp. 308-314. IEEE, 2018 .

5. Ivanov S, Durasov N, Burnaev E. Learning node embeddings for influence set completion. In2018 IEEE International Conference on Data Mining Workshops (ICDMW) pp. 1034-1037, IEEE, 2018.

# Acknowledgements

Firstly, I am thankful to my supervisor, Evgeny Burnaev, who gave me mentorship along this long journey. The freedom he gave me to pursue my research interests taught me to understand many decisions that one has to undertake to do good research. I'm also thankful to my committee and reviewers, Alexander Bernstein, Ivan Oseledets, Michael Bronstein, Matthew Blaschko, Maxim Panov, Alexander Panchenko, and Andrzej Cichocki, who gave valuable feedback to improve my thesis and guided my roadmap.

I also want to thank education office of Skoltech and all the people who helped students to succeed in their studies. Their will to solve the issues of students, to help them to grow professionally and always to back up in challenging situations is incredible and it led Skoltech to become one of the best schools in the world.

Being among the first students graduated at Skoltech is a big privilege and during this time I met amazing people who became my friends. I'm grateful that they found time to discuss my research and help me in need. It made my scientific voyage much more exciting.

Finally, my family who supported me all these years, staying around me and listening to my research ideas, I owe to say a big thank you!

# Contents

# Chapter 1

# Introduction

## 1.1 Motivation

History of graph theory accounts for almost 300 years of research highlighting some of the most beautiful and profound results in mathematics. Numerous applications in chemistry and physics, social sciences, navigation studies, biology, linguistics, knowledge capturing and other domains made graphs, not just a testbed of new machinery in engineering but a useful tool that could be applied to solve new problems. Besides the results obtained in mathematics using graphs such as the development of group theory and algebraic graph theory, graphs have been useful novel problems in engineering and other sciences. As such, first results in graph theory showed that no solution covers all of the seven bridges of Konigsberg [1], a result attributed to Euler who stands at the roots of algebraic topology, a field that is closely related to the graph theory. Applications of graph theory can be also found in the works of Gustav Kirchhoff [2], whose use of modern algebra originated from extensive development of topology by Whitney and others.

With the arrival of computers, many problems were formulated in terms of graphs and were possible to solve with the scale never possible before. To start one may want to decide the shortest path between two points in a graph that represents the different buildings in a map of a city. Canonical Dijkstra's algorithm is one way to solve the problem through the mean of a computer program and it is hard to underestimate the impact of this algorithm on our daily life. To name a few, from the GPS systems that navigate our cars during regular commutes to the communication of files on the Internet that reduces the transportation costs are all based on our understanding of how to find shortest or minimal paths in such abstract tool as a graph. Problems like Traveling Salesman Problem and Chinese Postman Problem [3] can be cast as finding a Hamiltonian cycle or an Euler tour on a graph, while a problem of creating a curriculum

for a school with $n$ subjects and $m$ classes is known as a timetabling problem and can be rephrased in terms of edge colorings. Undoubtedly, this highlights some of the pearls of computer science problems that are easy to formulate and explain through the means of a graph and at the same time have high practical value.

Recently a diverse, vast, and ever-growing set of problems on graphs has appeared that applies machine learning techniques to graph structures. Unsupervised, supervised, and reinforcement learning have been applied for the problems that could not be defined as traditional combinatorial problems. One example is a link prediction problem, i.e. prediction of new edges not present in the topology based on the edge relationships and, maybe, attributes on nodes or edges. While this problem can be posed with some optimization objective that could be solved via a combinatorial algorithm, it stems from the recommendation tasks in social networks or product catalogs, where the goal is to facilitate the search of novelties for an end-user based on historical observations and therefore an objective is typically posed as a differentiable function that could be solved with numerical methods. Link prediction is often viewed as a binary classification problem between a pair of entities, where the goal is to give a probability that there is an edge in-between. This problem is hard because most real-world networks are sparse and thus the prior probability of a link is quite small. This, in turn, leads to the problem of the evaluation of a model and its ability to make low-variance predictions. One proposed way to resolve these issues is to model network structure as a whole, taking jointly different links and labels over the entire graph, such as Markov Logic Networks [4]. More lately novel approaches based on neural network optimization demonstrate consistently high performance on this task by encompassing a wide range of existing heuristics [5]. This an example of a problem that illustrates that machine learning methods could be successfully applied in impactful scenarios where heuristic approaches perform sub-optimally.

Another example is the graph classification problem that requires a model to determine the class of a graph as a whole rather than its subparts. Traditionally graph kernels have been employed to solve the graph classification problem with a high success rate. In analogy to kernel functions in machine learning, i.e. a symmetric positive semidefinite function of two arguments, graph kernel takes two graphs as input and outputs a real number that indicates similarity between the graphs. One framework that designs graph kernels is a decomposition of each input graph into its subparts, from which it is easier to measure the similarity. For example, one may want to compute the histogram of degrees of each graph and then measure the similarity between the two distributions; however such approach is too simplistic and does not work well in practice as different graph topologies can have the same degree distribution. Instead, a good bulk of kernels were proposed based on manual engineering the features that may lead to better

performance in experiments. Examples include kernels based on shortest paths, motifs, subtree patterns, random walks, subgraph, directed acyclic subgraphs, neighborhood, and other substructures.

Driven mainly by the performance gains this approach requires a consensus of a kernel designer on the type of substructure in advance, which has strong and weak points. On the positive side, such kernels exhibit invariant similarity measure, i.e. graphs with the same topology would yield the same similarity measure, which is not the case for the neural and sampling approaches. Furthermore, a large body of work related to graph comparison and isomorphism problems was well adapted for graph kernel, demonstrating competitive results in this task. For example, graphlet kernel [6] is based on the assumption of correctness of reconstruction conjecture [7] that states that a graph can be fully reconstructed by a set of its smaller subgraphs. Analogously, Weisfeiler-Lehman kernel [8] uses canonical coloring procedure used in modern graph isomorphism algorithms to produce different colors of each node. On the other hand, to explore which type of kernel will be suitable for the task at hand one has to try a good pile of methods that would be optimal for available data and no graph kernel is a magic bullet.

Another part of graph classification is the choice of the algorithm that uses similarity scores or features of the graphs. Indeed in the case of graph kernels, many have experimented with SVM model that was applied on the kernel matrix of pairwise similarities between graphs, which exploits a kernel trick bypassing computation of graph features explicitly. However, this is a rather historical approach has been dawned lately by neural network models that optimize objective function defined over graphs topology have been proposed in the literature. For example, Deep Graph Kernels attempt to resolve diagonal dominance problem appearing in traditional kernels when the number of parameters is high. Diagonal dominance indicates very high similarity scores between identical graphs and very low values for all other pairs of graphs, which is not useful for the classification model. This happens because during the kernel computation a substructure of interest (random walk, graphlet, subtree, etc.) is appearing frequently in one graph and rarely in other graphs. So the way Deep Kernels resolve this issue is by learning additionally the matrix of weights for each dot product of identical substructures that would multiply histogram of substructures in each graph representations. To compute the weights one can generate a sentence corpus, where a sentence represents a sequence of substructures appeared together in the given graph, and then train a neural model with the Skipgram loss function.

To this end, these two parts of graph classification with machine learning, designing feature representations, also known as embeddings of graphs, and then applying a classification algorithm have not used labels of the graphs themselves. Contrary, graph

neural models for the graph classification problem are trained in supervised fashion using given labels of the graphs, a missing part in the graph kernel approach. One early example of is applying a convolution over the neighborhood of each node to compute each node representations that can later be pooled over the entire graph. Unlike images or sounds, graphs do not possess a clear grid structure requiring to adopt convolution and pooling operations over the nodes, but developments from the literature of graph kernels and graph isomorphisms are helpful to facilitate this problem. In particular, Weisfeiler-Lehman algorithm can be applied to determine efficiently the order of the nodes in the neighborhood for convolution mask. This is an illustrative use case when a theoretically backed development in one domain (here, graph kernels) can be used in another domain, for example as a graph representation or ranking mechanism in neural models.

Taking this into account one of the results that we present in this thesis is a new way of graph representation that has special properties that are not available in other graph representations and we show that such representations can achieve state-of-the-art results in graph classification. Particularly, our representations are based on a new form of random walks in a graph that we call anonymous walks. Anonymous walks are exactly random walks for which the node labels were replaced to reflect their first appearance in a random walk. Informally anonymous walk is a camouflaged cousin of a random walk, which has a property that one can compare statistics of anonymous walks for any different graphs, while random walks would require the same subset of labels. In this regard, anonymous walks stand along with such notions as degrees, motifs, or subtrees, allowing a researcher to analyze its combinatorial properties in different graphs. One crucial property with which anonymous walks shine is the graph invariance over its distribution, i.e. the pure histogram of all anonymous walks of a certain length that start from any node can reconstruct the topology of a graph exactly. As we show this implies that anonymous walks can be used as complete graph invariants and therefore capture the topology exactly for two graphs with unknown labels. To give a perspective, another candidate for the exact reconstruction of a graph, graphlet, still waits for the proof for more than 50 years [9], but showed impressive results in graph classification task [6]. While exact recovery is theoretically possible we show that the number of anonymous walks required in the general case can be factorial in the number of nodes and therefore not practical; yet, we piggyback on the analysis of graphlets to determine concentration inequality to trade off the computation complexity and approximation of the exact distribution. Our experiments show state-of-the-art results on the commonly used datasets in graph classification. The core property of the anonymous walk embeddings is their ability to capture the topology of a graph exactly, and hence the applications where such property is useful can benefit from usage of our

results. We verify this finding in other disciplines such as medical diagnostics, where a graph of different spots in a brain can be constructed by analyzing fMRI images.

Yet, more recently a new type of machine learning task has appeared, namely solving combinatorial problems on graphs with machine learning. Combinatorial problems have long in history in computer science in general and were one of the main drivers of many algorithmic gems and theoretical breakthroughs. For example, the development of the complexity theory has started with the realization that the satisfiability problem can be rephrased as an independent set problem formulated on graphs. One of such problems is influence maximization that concerns with the search of a set of nodes in a graph that would maximize its influence over other nodes. Influence function is dependent on the stochastic process and models the product adoption from one node to another and the goal is to maximize expected adoption. Unfortunately, the problem is NP-hard with a greedy approximation algorithm, which achieves at least $(1 - 1/e)$ proportion of the optimum [10], which is also NP-hard to approximate. In ablation studies, the greedy algorithm shows the top performance among all other algorithms but with the expense of the running time. Therefore much of the efforts have allocated to the design of fast algorithms that could creep up on the greedy solution. One of the presented results here is the design of an algorithm that we call Harvester for the case of undirected graphs with the advantage that it can likewise solve the reverse problem, seed minimization, that searches the minimal set of nodes that would reach a certain influence barrier. Our algorithm is based on the sampling of connected components that allows us to compute the influence score of each node more efficiently than greedy algorithm. We next use this understanding to reformulate influence maximization problem as a semi-supervised node ranking problem for which we use node representations obtained by neural network models including anonymous walk embeddings. We additionally formulate and propose a solution to another hard combinatorial problem of product recommendation in social networks that is arguably more plausible in real life than influence maximization, which we experiment using real social networks. While product recommendation is similar in its formulation to influence maximization, we show that it is provably harder than influence maximization as not only its NP-hard problem but also no approximation within a constant factor is achievable unless P = NP.

### 1.1.1 Goals and main results

Broadly our main goal was design and analysis of machine learning approaches that are applied to the problems where graph-structured data is given as input and various loss functions are required to be optimized. To achieve this goal the following tasks were considered:

- Development and analysis of graph embeddings that preserve isomorphism in an embedding space for graph classification problem;
- Evaluation of embeddings in several important applications such as a problem of medical diagnostics and protein function prediction;
- Development and analysis of node embeddings for the combinatorial problem of influence maximization;
- Theoretical analysis and design of algorithms for the problem of product recommendation in networks.

In this work, we build upon previous results in graph theory, neural network design, and optimization methods. Our code accompanies this thesis and is open-sourced to be used as an independent block of node or graph embeddings for other projects. Scientific novelty obtained in this thesis include:

- To the best of our knowledge, we are the first to propose graph embeddings that provably have complete invariance property, which as we show play an important role in many applications.
- Formulated a new problem of product recommendation in graphs, closely related to influence maximization problem. We prove the hardness results for the problem and propose a more efficient algorithm that is a strong baseline for this problem.
- We build several models suitable for several applications and achieve state-of-the-art performance. In particular, we have designed algorithms for graph classification, medical diagnostics and user recommendation on graphs.

The theoretical novelty of this thesis includes several proves on the analysis of new graph embeddings, their invariance, the running time analysis, and approximation bounds. Given their theoretical guarantees, proposed graph embeddings show state-of-the-art performance in many tasks beyond those requiring the invariance, which is verified in the experimental setting. Furthermore, we provide a proof of NP-hardness and inapproximability results of product recommendation problems in networks.

The motivation of the approach and correctness of the proposed methods have been supported by the double-blind reviews of the obtained results in top international conferences; by presentations and seminars at various academic and research venues; by the experimental studies on the computational tasks such as graph classification and graph isomorphisms. Results of this thesis have been published in the proceedings of the international conferences in machine learning and computer science and include:

1. Ivanov S, Karras P. Harvester: Influence optimization in symmetric interaction networks. In2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA) pp. 61-70. IEEE, 2016.

2. Ivanov S, Theocharidis K, Terrovitis M, and Karras P. "Content recommendation for viral social influence." In Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 565-574. ACM, 2017.

3. Ivanov S, and Burnaev E. "Anonymous Walk Embeddings." In International Conference on Machine Learning, pp. 2191-2200, 2018.

4. Sharaev M, Artemov A, Kondrateva E, Ivanov S, Sushchinskaya S, Bernstein A, Cichocki A, Burnaev E. Learning connectivity patterns via graph kernels for fmri-based depression diagnostics. In 2018 IEEE International Conference on Data Mining Workshops (ICDMW) pp. 308-314. IEEE, 2018 .

5. Ivanov S, Durasov N, Burnaev E. Learning node embeddings for influence set completion. In2018 IEEE International Conference on Data Mining Workshops (ICDMW) pp. 1034-1037, IEEE, 2018.

### 1.1.2 Thesis organization

In Chapter 1 we continue with the introduction of the common notation to the graph theory and introducing important concepts that are later used in the thesis. We also make a short overview of the graph isomorphism results that include the theoretical complexity of the problem, approaches to solve the problem and particularly hard instances for graph isomorphism. This overview aims to introduce the reader to the problem for which many theoretical and practical results exist and can enhance our understanding of graph representations.

Chapter 2 presents a new graph representation method and discusses several applications. We start by defining combinatorial and neural embeddings for graphs. Our graph representation method is based on a graph-structured object, anonymous walks, that has a theoretical guarantee on graph invariance. We utilize this and propose two types of embeddings, combinatorial and neural. Our applications include graph classification and medical diagnostics. In Chapter 3 we take a fresh look at modern combinatorial problems on graphs. In particular, we formulate a problem of product recommendation in social networks and prove its complexity. Our algorithm is based on the greedy algorithm but has a few tweaks that make it run an order of magnitude faster. We then discuss the well-studied problem of influencer recommendation, for which we show that

we can adapt graph embeddings to augment existing recommendations with new users based on our proposed graph embeddings. We conclude the thesis and discuss future directions in the Chapter 4.

## 1.2 Graph theory

### 1.2.1 Definitions and examples

Graphs are a way to represent a specific type of data, namely the data that defines relationships between the elements of a set. For example, cities connected by transportation routes or atoms linked by chemical bonds can be deemed as graphs. From this perspective, graphs are just another way to look at data at hand, similar to images, audio, or text. Then a principal interest in graphs comes from the problems that can be formulated with and solved by the means of graphs and that indeed has a rich history called graph theory.

Graph $G$ is a pair $(V, E)$, where $V$ is a set of *vertices* or *nodes* and a set $E \subset V \times V$ is a set of *edges*. Hence, each edge is associated with a pair of vertices $(i, j)$ of set $V$, in which case we say that $i$ is *adjacent* to $j$ and is a *neighbor* of $j$. A graph is called *undirected* if for any edge $(i, j) \in E$, an edge $(j, i)$ also belongs to $E$; otherwise, it is *directed* graph. Some problems in this thesis are defined specifically for undirected case, while others are indifferent to the type of graphs, and we will specify the graph type explicitly for each type of problem.

For undirected graph the set of all neighbors of node $v$ is denoted $N_v$ and the cardinality of $N_v$ is called *degree* of $v$. For directed graph there are two sets for each node $v$: one for the nodes to which $v$ is adjacent to (also called *out-neighbors*), which is denoted $N_v^{out}$, and one for the nodes which are adjacent to $v$ (also called *in-neighbors*), which is denoted $N_v^{in}$. The numbers of in-neighbors $d_v^{in}$ and of out-neighbors $d_v^{out}$ called *in-degree* and *out-degree* of $v$. We say that a graph is *simple* if it does not contain *loops*, i.e. edges of type $(i, i)$, and does not contain *parallel* edges, i.e. each $(i, j) \in E$ has a single copy in a graph. Graphs that have edges between each pair of vertices are called *complete*.

Additionally, a graph can have human-defined labels on nodes, edges, or global topology. We call a graph $G = (V, E, l)$ *element-labeled* if a graph $G = (V, E)$ has a function $l : X \mapsto \Sigma$, which assigns a label from an alphabet $\Sigma$ to each element of set $X$, which may include a set of nodes $V$, edges $E$, a global graph information. A particular case of an element-labeled graph is a *colored* graph when each vertex is colored, i.e. has a single label that is called a color of a node. Different nodes can have the same color.

For algorithms that work with colored graphs only, it is common to give some default coloring, for example, based on the degree of the vertices. A graph is called *weighted*, if there is a function $f_w : e \mapsto \mathbb{R}$ that maps each edge to the associated weight number. Note that these labels refer to extra information about the graph, for example, the types of bonds in a molecule or the domain expertise of authors in citation networks, which is different to the labels that are used to represent one or the other graph (see next section on graph isomorphism).

### 1.2.2 Walks, paths, and cycles

A *walk* is a sequence of vertices $(v_1, \ldots, v_l)$ such that any consecutive pair of vertices $(v_i, v_{i+1})$ in this sequence is an edge of a graph, i.e. $(v_i, v_{i+1}) \in E$ for all $i = \overline{1, l-1}$. A walk where nodes are not repeated is called a *path*, and a walk where edges are not repeated is called a *trail*. A walk that starts and ends at the same vertex is called a *closed walk* or *circuit*, or *cycle*.

In an undirected case, if between all pairs of vertices there exists a path that connects them, then a graph is called *connected*, and *disconnected*, otherwise. A disconnected graph can be viewed as a collection of connected graphs (possibly of one vertex) called connected components. In the directed case, a graph is called *strongly connected* if there is a directed path between each pair of vertices (equivalent to the connectedness of undirected graph). A graph is called *weakly connected* if there is a path between each pair of vertices if edges are assumed to be undirected. A graph $\overline{G}$ is *complementary* (or complement) to $G$ if $(u, v) \in E(\overline{G})$ if and only if $(u, v) \notin E(G)$.

A graph is called *acyclic* there are no cycles for any walk in a graph. It follows that undirected graphs are not acyclic, while directed graphs with no cycles are called *directed acyclic graphs* (DAGs). A graph is called *bipartite* if every cycle has an even length or equivalently to partition a vertex set $V$ into two non-overlapping sets $V_1$ and $V_2$ such that any edge has one endpoint in $V_1$ and another endpoint in $V_2$.

A path that contains each edge of a graph exactly once is called *Euler path*. If Euler path starts and ends at the same vertex it is called *Euler circuit*. A path that visits each vertex in a graph exactly once is called *Hamiltonian path*; and if a Hamiltonian path can be closed to a starting vertex, then it is called a *Hamiltonian circuit*.

The problems associated with finding paths and circuits that satisfy some criteria are of great importance both for practical and theoretical reasons, so it is worth discussing the complexity of some of those that are relevant to this thesis.

The shortest/longest path problem asks to find a path of minimum/maximum length in a graph that connects two nodes $s$ and $t$. The shortest path problem has many efficient solutions including classic algorithms of Dijkstra, Bellman-Ford, and Floyd-Warshall. For an overview of these problems refer to [11]. On the other hand, the longest path problem has no known polynomial solutions and is $\mathbb{NP}$-complete. Hence, solutions come in the form of approximation algorithms [12], heuristics, or are suitable only for special classes of graphs such as directed acyclic graphs. An Euler circuit/path problem asks to present an Euler circuit/path in a graph or prove that it is not possible to construct such a path. It is arguably one of the first problems in graph theory solved famously by Euler. The theorem states that: Euler circuit exists in a connected graph if and only if each of its vertices has even degree. Euler path exists in a connected graph if and only if it has exactly zero or two vertices that have odd degree.

Contrary, the Hamiltonian path problem that asks to present a Hamiltonian path or prove its absence is known to be $\mathbb{NP}$-complete. A few sufficient conditions exist that guarantees the existence of a Hamiltonian path such as If for any pair of non-adjacent vertices $u$ and $v$ the sum of their degrees is not less than the number of vertices in a graph, i.e. $deg(u) + deg(v) \geq |V|$, then the graph has Hamiltonian circuit. Note that the longest path problem and Hamiltonian circuit problem are reducible to each other given the solution to one of the problems. Another notable example of a combinatorial problem, Travelling Salesman Problem (TSP), asks to find a circuit of the minimum length in a complete graph. TSP is known to be $\mathbb{NP}$-complete as it is a special case of Hamiltonian path problem.

### 1.2.3 Basic representations

Basic representation of a graph include adjacency matrix and incidence matrix. *Adjacency matrix* $A$ of graph $G$ defines a bijection between vertices $v_1$, $v_2$, ..., $v_n$ and the corresponding integer set 1, 2, ..., $n$. Then a cell $A_{ij} = 1$ if $(v_i, v_j) \in E$ and 0, otherwise. Because bijection is arbitrary, there are $n!$ possible adjacency matrices for a graph with $n$ vertices. Alternatively incidence matrix defines a matrix of order $|E| \times |V|$, with an element $Q_{ev}$ for an edge $e$ and a vertex $v$ given by:

$$Q_{ev} = \begin{cases} 1, & \text{if } v = i. \\ -1, & \text{if } v = j. \\ 0, & \text{otherwise.} \end{cases}$$

The area that studies algebraic properties of adjacency and incidence matrices of a graph is called algebraic graph theory and has discovered numerous results in graph theory [13, 14]. For example, $\chi(G) = \det(\lambda I - A)$ a characteristic polynomial of a graph $G$ and can be written as:

$$\chi(G) = \lambda^n + c_1\lambda^{n-1} + \ldots + c_n \tag{1.1}$$

Then:

1. $c_1 = 0$
2. $-c_2$ is the number of edges
3. $-c_3$ is twice the number of triangles

Similarly, the incidence matrix $Q$ can be used to represent *Laplacian* of the graph, which is popularly used for eigenvalue decomposition.

$$L = QQ^T = \Delta - A, \tag{1.2}$$

where $\Delta$ is a diagonal matrix of degrees of nodes.

## 1.3 Graph isomorphism

Two popular complexity classes of problems are the class $\mathbb{P}$ of polynomial-time solvable problems and the class $\mathbb{NP}$ of problems that can be verified in polynomial-time given a proposed solution. Class $\mathbb{P}$ includes problems of computing all-pairs shortest path, minimum spanning tree, or maximum flow; while, $\mathbb{NP}$ class subsumes $\mathbb{P}$ class and exposes other hard interesting problems, such as computing longest path or Hamiltonian cycle [15], for which no known polynomial-time solution exists to date. Somewhere in between the classes class $\mathbb{P}$ and $\mathbb{NP}$-hard problems lie the problems of unknown complexities with the most notable instance of graph isomorphism (GI) problem, for which no theoretical proof of being in class $\mathbb{P}$ has yet been established.

Isomorphism between two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ is a bijective function $\phi : V_1 \mapsto V_2$ such that any edge $(u, v) \in E_1$ if and only if $(\phi(u), \phi(v)) \in E_2$. Obviously, there are pairs of graphs that don't possess such function $\phi$ (e.g. graphs on different number of vertices) and pairs of graphs for which such function exists (e.g. pair of

identical graphs). The question of graphs isomorphism asks whether two graphs $G_1$ and $G_2$ have isomorphism function $\phi$. The problem has efficient algorithms in $\mathbb{P}$ for certain classes of graphs such as planar or bounded-degree graphs [16, 17], but in the general case admits only quasi-polynomial algorithm [18].

In practice many GI solvers are based on individualization-refinement paradigm [19], which identifies a *complete graph invariant* for each graph, i.e. a function on a graph $\chi_G$ such that $G_1 \cong G_2 \iff \chi_{G_1} = \chi_{G_2}$. In particular, these solvers compute a canonical labeling of a graph:

**Definition 1.1** (**Canonical labeling**)**.** A canonical labeling of a graph $G$ is a graph $C(G)$ such that:

1. $G \cong C(G)$,
2. $C(G') = C(G)$ for any isomorphic graphs $G$ and $G'$.

In other words, a canonical labeling represents the whole group of isomorphic graphs. Thus, one can think that canonical graph $C(G)$ is defined on vertices $S_{|V|} = \{1, 2, \ldots, |V|\}$ and that for any isomorphic graph $G = (V, E)$ there exists a function $\phi_G : V \mapsto S_{|V|}$, which preserves edge connectivity. If an algorithm for canonical labeling is given, one can easily report if two graphs are isomorphic. In particular, for two graphs $G_1$ and $G_2$ let $\psi : \phi_{G_1}^{-1}(i) \mapsto \phi_{G_2}^{-1}(i)$ be a bijection for any $i \in S_{|V|}$. Therefore if edges are preserved under bijection $\psi$ then it's an isomorphism, otherwise, graphs are not isomorphic. Importantly, while finding canonical labeling of a graph is at least as hard as solving GI problem, solvers tackle the majority of pairs of graphs efficiently, only taking exponential time on the specific hard instances of graphs that possess highly symmetrical structures [20].

Two graphs $G$ and $H$ are *identical* if $V(G) = V(H)$ and $E(G) = E(H)$, i.e. the sets of $V$ and $E$ are equal. However, it is also possible that two graphs have essentially the same structure (also called topology) but they are either depend on different universe of elements $V_1$ and $V_2$ *or* the set of edges $E_1$ and $E_2$ defines relationships between nodes in different ways. To understand this consider an example in Figure 1.1.

Obviously, two graphs have the same topology; however, nodes are defined using a different set of elements. For the first graph $V_1 = \{a, b, c, d, e, f\}$ and for the second it is $V_1 = \{\kappa_1, \kappa_2, \kappa_3, \kappa_4, \kappa_5, \kappa_6\}$. This situation is prevalent in practice when there is no explicit labels are given for the vertex set and one has to define the node elements.

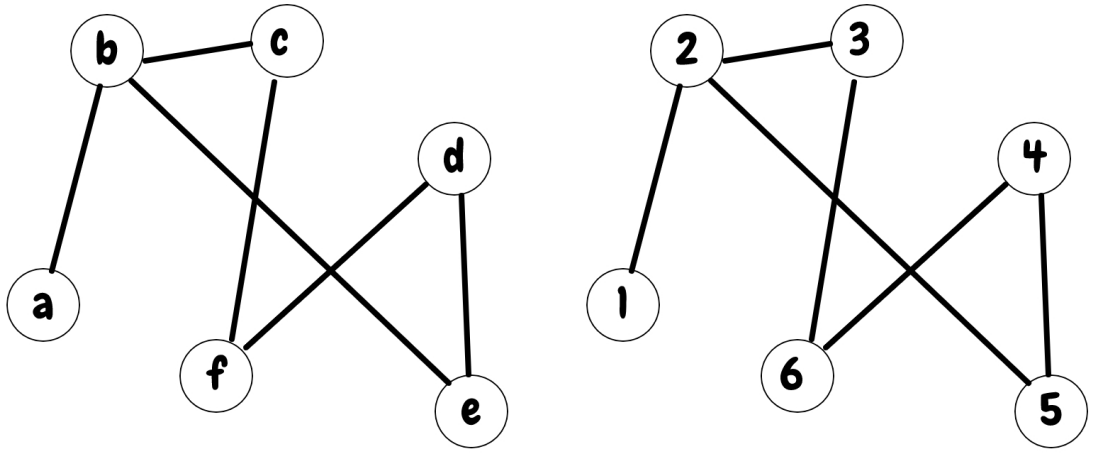



FIGURE 1.1: Isomorphic graphs with different node labels.

In the case of simple graphs, *isomorphism* between two graphs $G$ and $H$ is a bijection $\theta : V(G) \mapsto V(H)$ such that adjacency is preserved, i.e. $(u, v) \in E(G)$ if and only if $(\theta(u), \theta(v)) \in E(H)$.

**Definition 1.2** (Graph Isomorphism problem $\mathbb{ISO}$)**.** Given two undirected graphs $G$ and $H$ return an isomorphism function or report that it does not exist.

Graphs for which there exists an isomorphism are called *isomorhic* and are denoted $G \cong H$. Figure 1.2 presents three graphs, where left and top-right are topologically isomorphic, while the bottom-right is not isomorphic, despite having the same number of nodes or edges.

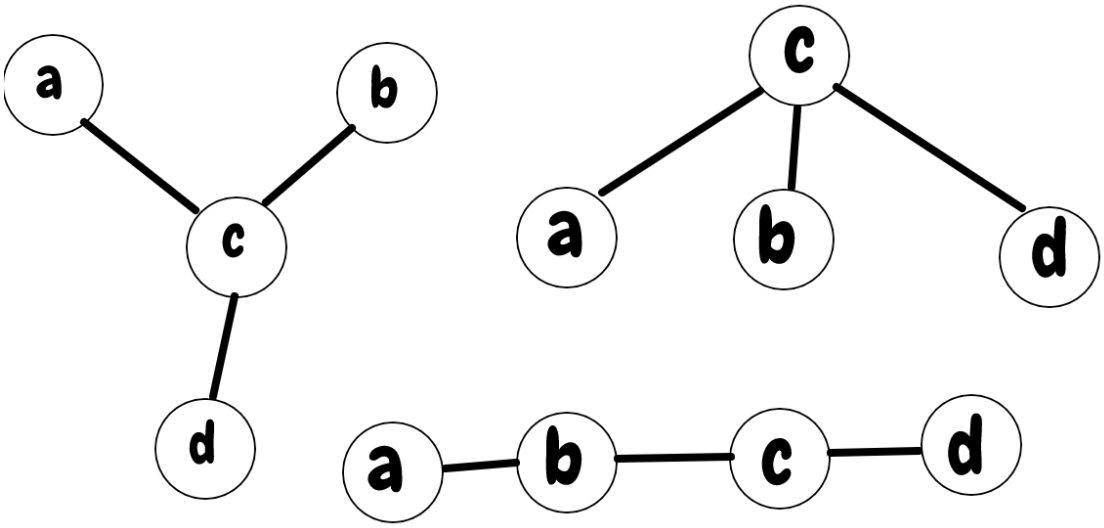



FIGURE 1.2: Examples of isomorphic and non-isomorphic graphs. Despite having the same number of nodes and edges, graph on the bottom-right is non-isomorphic.

Automorphism is the second case when two graphs with the same topology are not identical. In particular, *automorphism* $\alpha$ of a graph $G$ is an isomorphism of the graph

to itself, i.e. $(u, v) \in E(G)$ if and only if $(\alpha(u), \alpha(v)) \in E(G')$ and $V(G') = V(G)$. In other words, automorphism is an isomorphism between a graph $G$ and another graph $G'$ with the same vertex set. An automorphism of a graph determines similar vertices in a graph, for which permutation among themselves does not change the graph. A problem that is polynomially-equivalent to the problem $\mathbb{ISO}$

**Definition 1.3** (Automorphism Count problem $\mathbb{ACOUNT}$)**.** Given an undirected graph, $G$ find the cardinality of a set of all automorphisms for graph $G$.

Example in Figure 1.3 shows an original graph and its automorphic graph. Graph for which all vertices are similar, such as the complete graph, are called *vertex-transitive*. If a graph has only identical automorphism then it is called *assymetric*.

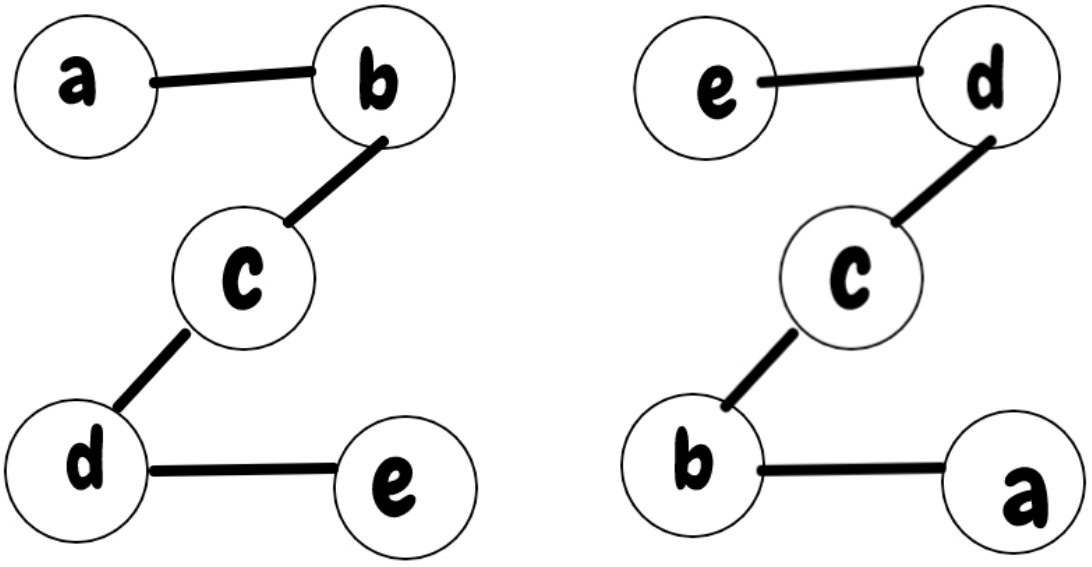



FIGURE 1.3: Example of automorphism. Graphs are defined on the same node labels and are isomorphic. Automorphism is defined by permutation ($a$, $e$) and ($b$, $d$)

Normally, one is interested in the structural properties of a graph, rather than the labelings that the graph incurs. Therefore, when referring to a graph we will imply an *unlabelled graph* that serves as a representative of all isomorphic graphs to the structure it represents and we say that all properties are the same *up to isomorphism* of this graph.

### 1.3.1 Complexity of graph isomorphism

Graph isomorphism problem is one of the problems for which no known polynomial-time algorithms exist in the general case but the performance of the best theoretical algorithms is better than for $\mathbb{NP}$-complete problems [21]. It is clear that if isomorphism

function $\theta$ is given, then one can test for isomorphism between two graphs in polynomial-time, but simply checking each of $\binom{n}{2}$ of pairs of vertices on the adjacency, so the problem is in $\mathbb{NP}$. On the other hand, to find such an isomorphism is a challenge.

A trivial way to find an isomorphism is to enumerate all possible permutations of vertices as a bijection between two graphs. As there are $n! \sim \exp(\mathcal{O}(n \log n)))$ permutations this algorithm is exponential. Then, an algorithm by [22] achieves $\exp(\mathcal{O}(n)))$ running time by applying branch and bound method. At the same year, by considering classification of finite simple groups (CFSG) [23] provided an algorithm of $\exp(\mathcal{O}(\sqrt{n \log n})))$, which is considered to be the best accepted running time for this problem. Result by [18] runs in quasi-polynomial time $\exp(\log n^{\mathcal{O}(1)}))$ by also considering CFSG and tackling barrier configurations of the previous algorithm; yet, the review of the algorithm is still in progress.

When the structure of the graph is restricted in some way, then it is possible to derive a polynomial-time solution.

Special classes of graphs such as of bounded degree [17], bounded color size [24], bounded genus [25], and planar graphs [16] are solved in polynomial time, while for random graphs is tested in expected polynomial time [26].

A *class* of graphs $\kappa$ is a collection of graphs that is closed under isomorphism, i.e. if $G \in \kappa$ and $G' \cong G$, then $G' \in \kappa$. It is known that some classes of graphs are equivalent for graph isomorphism problem for undirected graphs $\mathbb{ISO}$. We say that a problem $P_1$ is *reduced to* a problem $P_2$, $P_1 \propto P_2$, if from the polynomial-time algorithm for the problem $P_2$, $P_2 \in \mathbb{P}$, we can use the algorithm to show that $P_1$ is also polynomial-solvable, $P_1 \in \mathbb{P}$. Two problems are called *equivalent*, $P_1 \approx P_2$, if $P_1 \propto P_2$ and $P_2 \propto P_1$. A class $\kappa$ is Graph Isomorphic complete ($\mathbb{GI}$-*complete*), if the problem of solving graph isomorphism for class $\kappa$ is equivalent to solving graph isomorphism for class of all undirected graphs $\mathbb{ISO}$.

Many classes are known to be GI-complete [21] and we show one such reduction as an example.

**Theorem 1.4.** *Classes of connected graphs and bipartite graphs are* $\mathbb{GI}$*-complete.*

*Proof.* As reduction from a particular class of graphs $\mathbb{ISO}_\kappa$ to a class of all graphs is always valid, the only remaining part is to show that the problem $\mathbb{ISO}$ is reduced to $\mathbb{ISO}_\kappa$.

To show that any undirected graph $G$ can be solved by an algorithm for the class of connected graphs $\mathbb{ISO}_{Con}$, consider a graph $G'$:

$$\text{G'}=\begin{cases} G, & \text{if } G \text{ is connected.} \\ \overline{G}, & \text{otherwise.} \end{cases}$$

The graph $G'$ belongs to the class of connected graphs and hence has a solution to $\mathbb{ISO}_{Con}$. Note that knowing isomorphism for a complementary graph can be used to check isomorphism of the original graph as isomorphism is determined on the same vertex set $V$. Therefore $\mathbb{ISO}_{Con} \cong \mathbb{ISO}$.

To show that any undirected graph $G$ can be solved by an algorithm for the class of bipartite graphs $\mathbb{ISO}_{Bip}$, consider a graph $G'$ for which the vertex and edge sets are defined as follows:

$$\text{V(G')} = \text{V(G')} \cup W \cup \{w\},$$

$$\text{E(G')} = \{(\text{w, x}): \text{ x} \in W\} \cup \{(w_{uv}, u), (w_{uv}, v) : u, v \in V(G), (u, v) \in E(G)\}.$$

That is for every edge $(u, v)$ in graph $G$ we create a new vertex $w_{uv} \in W$ and connect it to $u$ and $v$ and remove the original edge $(u, v)$. Also we add additional vertex $w$ that is connected to all vertices in $W$.

Now considering degree of a vertex $u$ in a new graph $G'$:

$$\text{d}_{G'}(u) = \begin{cases} |E(G)|, & \text{if } u = w, \\ 3, & \text{if } u \in W, \\ d_G(u), & \text{otherwise.} \end{cases}$$

(1.3)

We can consider the case when $G'$ is connected bipartite graph as previously it was shown that isomorphism problem of disconnected graphs is equivalent to the isomorphism problem of connected graphs.

Let $\gamma : \mathbb{ISO}_{Bip}$ be an isomorphism for a bipartite graph $G$. Then it should preserve degrees of that graph, i.e. $d_{G'}(u) = d_{G'}(\gamma(u))$. Note that all vertices are one of the three types according to the equation (1.3.1). We can assume that $|E(G)| > d_G(u)$ (because otherwise the graph $G$ is a tree and hence a bipartite graph). Then the vertex $w$ in graph $G'$ is mapped to itself under isomorphism $\gamma$ (because it is the only vertex of such degree). Then all of its neighbors $u \in W$ must be mapped to the vertices of the same set, i.e. if $u \in W$, then $\gamma(u) \in W$. Hence all vertices in $u \in V(G)$ are also mapped to the same vertex set $\gamma(u) \in V(G)$. As $\gamma$ is an isomorphism then all edges between $V(G)$ and $W$ should be preserved under $\gamma$. As a pair $(u, v) \in E(G)$ if an only if there is a vertex $k \in W$ such that $(u, k) \in E(G')$ and $(v, k) \in E(G')$, then an isomorphism of all undirected graphs can be solved by using an isomorphism for bipartite graphs on the graph $G'$.

□

There are other graph families such as directed, regular, directed acyclic graphs and many other types of graphs that can be solved efficiently for graph isomorphism problem. [27] presents some of the proves for such families.

### 1.3.2 Practical graph isomorphism

While from a theoretical standpoint of view the GI problem is interesting, in practice many algorithms exist that solve the problem efficiently for reasonable sizes of graphs (up to 10000 nodes). However, there are still instances of highly-regular graphs for which these solvers take exponential time and the question of scaling solvers to larger graphs is open. In what follows we present a famous Weisfeiler-Lehman algorithm that is commonly used in practice as part of the color refinement block in the solvers, but that still fails for regular graphs, for example, [1].

Next, we describe the main ideas of Brendan McKay's algorithm that was one of the first successful attempts to solve GI in practice and that largely influenced its descendants.

[1] We provide a history behind the algorithm and its authors in the Appendix A

In the end, I briefly describe works that are related to finding hard instances for the solvers used in practice.

Weisfeiler-Lehman (WL) algorithm is one of the first practical algorithms for graph isomorphism problem and has been recently applied in machine learning community to the problem of molecule classification with graph kernels [28] and neural networks[29]. The algorithm has been extensively studied theoretically, drawing a relationship with logic [30] and showing there are always graphs for which the algorithm fails [20].

We first describe a 1-dimensional version of the WL algorithm. Given a colored graph $G = (V, E, C)$, the algorithm proceeds iteratively such that in every new iteration a new coloring is established. In the case of unlabeled graphs, one can assume uniform coloring for all nodes. At each iteration $i$, we first apply some arbitrary ordering for the node colors $C = (c_1, c_2, \dots c_k)$. The coloring of a node $v$ is a tuple at the iteration $i$:

$$
\begin{aligned}
&\mathrm{c}_i(v) = (c_{i-1}(v), |\{w, |w \in N_v \text{ and } c_{i-1}(w) = c_1\}|,\\
&|\{w, |w \in N_v \text{ and } c_{i-1}(w) = c_2\}|,\\
&\dots,\\
&|\{w, |w \in N_v \text{ and } c_{i-1}(w) = c_k\}|)
\end{aligned}
$$

That is each node gets a new color that consists of its color on the previous iteration, and the sizes of colors of each type for the neighborhood $N_v$ such that the ordering of colors at each iteration is predetermined and the same for all nodes in the graph. This procedure of getting new colors from the old ones is called a *naive vertex refinement* or *1-dimensional Weisfeiler-Lehman refinement*. Note that an initial color from any vertex to any other vertex will propagate at most in $|V|$ iterations, and in fact after at most $\mathcal{O}(|V|)$ the colors of nodes will stabilize in the sense that colorings $C_i$ and $C_{i-1}$ are the same up to permutation of colors. The running time of this algorithm is $\mathcal{O}((|V| + |E|) \log(|V|))$ [31, 32] and the results by [33] shows that graph isomorphism can be tested in liner average time for most graphs, which makes WL algorithm to be very good first candidate for the practical graph isomorphism.

Intuitively, 1-dim. WL algorithm updates each vertex based on the colors of its neighbors and two vertices will have the same color after one color refinement if and only if they have the same colors in the neighborhoods. A $k$-dimensional variant of WL is based on

the tuples of size $k$ of vertices, instead of the neighborhoods. Let $G$ be a graph and $k \geq 2$. In each iteration, the algorithm will compute a set of colors $C_i^k : V^k(G) \mapsto C$, where $C$ is a set of colors at this iteration. In the first iteration, we give two tuples $t_1$ and $t_2$ of vertices (not necessarily the same) of size $k$ the same color if the induced subgraphs of the vertices in the tuples $t_1$ and $t_2$ are isomorphic. Note that the nodes in the tuple are considered to be ordered. Let also $r(t, w, i)$ be a tuple of the same size as $t$ with the $i$-th vertex in $t$ replace by a node $w \in V$. Then a new color is obtained for the tuple $t$ as follows:

$$
\begin{aligned}
&\mathrm{C}_i^k(t) = (C_{i-1}^k(v), \{C_{i-1}^k(r(t, w, 1)),\\
&C_{i-1}^k(r(t, w, 2)),\\
&\ldots,\\
&C_{i-1}^k(r(t, w, k)), |w \in V\})
\end{aligned}
$$

In $k$-dim. WL algorithm in every iteration every tuple of vertices $t = (v_1, \ldots v_k)$ is given a new color according to its previous color and the colors of tuples, when one vertex of $t$ is replaced with any other vertex in $V$. It is a generalization of 1-dim. WL, where a tuple is a single vertex and similarly it stabilizes after $\mathcal{O}(n^k)$ rounds. Note that isomorphic graphs will get the same stable colorings and therefore the algorithm can be used for $\mathbb{ISO}$ testing.

A brute force approach fora single iteration takes $\mathcal{O}(kn^{k+1})$ time and the overall $k$-dim. WL algorithm can be computed in $\mathcal{O}(k^2 n^{k+1} \log n)$ [30]. It has been noticed that k-dim. WL for sufficiently large $k$ subsumes many other combinatorial algorithms in their ability to recognize some statistics on graphs, for example, number of cycles [34].

Despite lacking a polynomial-time algorithm, there exists a few software tools that can solve graph isomorphism for all graphs very efficiently. To name a few the solvers are nauty/traces [35], bliss [36], conauto[37], saucy[38]. These state-of-the-art solutions are fast and can solve an abundance of graph pair instances within seconds and even hard graphs with thousands of vertices take hours to resolve. The underlying idea of these methods is in the individualization-refinement paradigm that is described below, while the difference is in how exactly the search tree traversal is performed and how automorphism of different tree nodes is detected.

In the WL algorithm, we already used a refinement routine that updates the colors of the nodes according to the colors of the neighbors (in the case of 1-dim. refinement). This part remains the same for the solvers. What is added is the second part called *individualization*. In individualization we have a graph $G = (V, E, c)$ and a vertex $v$ and we get a new graph $G_v = (V, E, c')$, where $c' = c$ for all vertices but $v$, for which it gets a new color $c_v \notin c$. Thus a single vertex gets a new color.

Then an algorithm repeats in iteration, by first applying refinement and then individualization procedures, until each vertex has its own separate color. In the first iteration a naively refined vertex set is given and one *target cell*, i.e. a class of the same color, is selected and individualized for each vertex of the cell, creating children of the first refinement. Then, refinement-individualization is applied again until all nodes have their own classes for all individualized partitions. In the end, there will be many possible discrete partitions and one is selected according to a predetermined rule (e.g. lexicographical order of the paths to the leaves) to be a representative of the whole graph. Then two graphs are isomorphic if and only if their representatives are the same. This can already be used as a test for the isomorphism but it would take exponential running time for graphs that have many vertices in the same partition such as a complete graph. So in addition to refinement-individualization, the solvers also employ special techniques to reduce the running time. In particular, some tree nodes in the search space are equivalent and hence it is not necessary to individualize them all but rather do it only once. Also if some paths are already not "promising" compared to the predetermined order, then these paths can be discarded as well. A more in-depth explanation can be found for example in [3, 35, 39, 40].

Alongside development of advanced theory of solving graph isomorphism, there is an attempt to counteract proposed algorithms by new constructions of the graphs for which the running time becomes exponential in the number of vertices. [20] presented a family of graphs such that for any $k$ the $k$-dim. WL algorithm cannot distinguish the graphs, even for the graphs of small degree. Their construction is based on Fürer gadgets that replace each vertex with a new subgraph and each edge $(v, u) \in E$, gadgets are connected by a pair of parallel edges. The main idea is to start with a 3-regular graph for which two new non-isomorphic graphs $G_1$ and $G_2$ are made, not distinguishable by Weisfeiler-Lehman algorithm. Fürer gadget is defined as follows:

- For each edge in the original graph $G$, $(s, t) \in E$, we create four distinct vertices $a_{s,t}$, $a_{t,s}$, $b_{s,t}$, $b_{t,s}$;

- For each vertex $v \in V$, we create a vertex $c_{v,S}$ for each odd-cardinality subset $S \subset N_v$. So in the example of 3-regular graph, each vertex $v$ has three neighbors $x$, $y$, $z$ and we create four additional vertices $c_{v,\{x\}}$, $c_{v,\{y\}}$, $c_{v,\{z\}}$, $c_{v,\{x,y,z\}}$;
- We also add two edges, one between $a_{s,t}$ and $a_{t,s}$, and one between $b_{s,t}$ and $b_{t,s}$;
- We add edges $(a_{s,t}, c_{s,S})$ for each $S$ such that $t \in S$;
- We add edges $(b_{s,t}, c_{s,S})$ for each $S$ such that $t \notin S$.

As $G$ is undirected, we add Fürer construction twice for every edge. The resulted graph $G_{||}$ is a combination of Fürer gadgets by parallel edges.

Then, let $G_{s,t}$ be the graph made from $G_{||}$ by deleting the edges $(a_{s,t}, a_{t,s})$ and $(b_{s,t}, b_{t,s})$ and adding two cross edges $(a_{s,t}, b_{t,s})$ and $(b_{s,t}, a_{t,s})$. A key observation is that for any pair of edges $(s_1, t_1)$ and $(s_2, t_2)$ graphs obtained by adding the cross edges are isomorphic.

**Lemma 1.5.** *For any edges $(s_1, t_1)$, $(s_2, t_2) \in E$, the graphs $G_{s_1,t_1}$ and $G_{s_2,t_2}$ are isomorphic.*

*Proof.* You can move the cross edge around the graph in the following sense. For any pair of edges $(i, j), (j, k) \in E$, there's an isomorphism of $G_{(i,j)}$ that exchanges $a_{(j,i)}$ and $b_{(j,i)}$ and exchanges $a_{(j,k)}$ and $b_{(j,k)}$. This isomorphism is a permutation of the vertices $c_{(j,S)}$, and it fixes every other vertex in $G_{(i,j)}$. If you draw the fragment of the graph corresponding to vertex $j$ and its incident edges, the isomorphism should be pretty obvious.

This means that $G_{(i,j)} \simeq G_{(j,k)}$, because the isomorphism "uncrosses" the edge $(i, j)$ and "crosses" the edge $(j, k)$. Since $G$ is connected, it contains a path whose first edge is $(u, v)$ and whose last edge is $(x, y)$, and we can use the isomorphisms corresponding to each adjacent pair of edges in this path to move the cross in $G_{(u,v)}$ to $(x, y)$.

□

From Lemma 1.5 it follows that we can take any edge $(s, t)$ and denote $G_\times = G_{s,t}$. It is easy to see that graphs $G_{||}$ and $G_\times$ are non-isomorphic as the first does not have cross edges and the second has it. Intuitively these two graphs are very similar to each other with a single twist introduced for a pair of edges, which in the case of graphs with big separators make it is hard to distinguish by the algorithm. In their seminal work Cai, Fürer , and Immerman [20] showed that $d$-dimensional Weisfeiler-Lehman algorithm can be expressed in so-called fixed-point logic with counting and that any formula that uses

$k$ distinct variables cannot distinguish $G_{||}$ and $G_{\times}$, where $G$ has no separator of size less than $2k$. As there is a direct relationship between WL algorithm and the formula in the first-order logic, for any $d$-dimensional Weisfeiler-Lehman algorithm there is a graph $G$ with separator of size less than about $2k$ such that the algorithm cannot tell the difference between $G_{||}$ and $G_{\times}$.

Later Miyazaki [41] used the same construction to design a family of graphs to a by-then state-of-the-art algorithm nauty [39]. To design the family of graphs, he takes 3-regular graphs of the form of a path, with self-loops at the endpoints and double edges between each even pair of points inside the path and applies Fürer gadgets to it. The resulted graph is a simple graph and with certain coloring, he shows that nauty takes exponential time on this family of graphs.

More recently a new upgrade to the nauty solver has been made and the resulted algorithm is called Traces[35]. It is improved over nauty by the fact that it detects automorphisms while refining the vertices and it significantly reduces the search space. This means that the running time is divided by the size of the automorphism group of the graph and Miyazaki graphs are indeed not as difficult as for nauty. Again for this new solver, a new family of graphs has been proposed that is challenging and requires exponential running time. In particular, [42] present graphs that do not have non-trivial automorphism and have linear Weisfeiler-Lehman dimension, which leads to exponential lower bound on the search tree size under any target cell selection strategy.

As of the beginning of 2019, the current state-of-the-art in hard instances for $\mathbb{ISO}$ problem are the graphs due to [43]. Their construction also relies on the adaptation of Fürer gadgets and graphs with no non-trivial automorphisms. To preserve the graphs as small as possible some techniques preserve vertices without changing the local automorphism structure of the gadgets. The resulted graphs are of bounded degree and therefore can be solved in polynomial-time theoretically [17]; however, they pose the challenges to current individualization-refinement algorithms. Additionally, similar construction can be used to prove theoretically an exponential lower bound for such algorithms [44].

# Chapter 2

# Anonymous Walk Embeddings

## 2.1 Motivation

Combinatorial optimization problems are widely used in real-world applications spanning many different industries, including but not limited to energy, medicine, finance, communication, and transportation. In this thesis, we will discuss some of such problems (e.g. graph isomorphism) which are known to be problems for which no known-polynomial time algorithm exists. Yet, these problems appear ubiquitous in different applications and therefore are of paramount importance to researchers. Besides such problems have been a driver for novel ideas in different areas of mathematics and natural sciences (e.g. development of group theory [14]).

Machine Learning (ML), on the other hand, deals with the tasks with the presence of real data, that often comes with noise and in a limited amount. One can say that ML is a new epoch for statistical studies that has centuries of years of development, featuring already classical results such as Bayes rule and Monte-Carlo methods [45]. ML has gained a lot of traction recently due to the two factors, which reinforce each other: accumulated data sources and increased computational capabilities. Indeed, nowadays the data come from different places with ease for a computer to perform computational operations on these data such as when processing our photos and videos, text that we write online, music that we listen to. Such data don't have a known true distribution and therefore does not bear exact mathematical formulation for which combinatorial algorithms can be applied. Rather data have patterns and it is our goal to discover such patterns in the presence of noise and scarcity of data. A sub-field of ML called Deep Learning (DL) recently has beaten other ML algorithms such as Random Forests and Logistic Regression in many domains such as Computer Vision and Natural Language Processing, while understanding of its success is still very limited and it is largely attributed to the

massive number of parameters (often in the scale of billions of parameters per model) and clever prior (e.g. convolutional masks) and regularization (e.g batch normalization) techniques [45] that experimentally have shown increased performance of the models.

It already can be noted that combinatorial optimization and machine learning are closely related, overlapping and complementing in different settings. It has been a recent trend to apply machine learning models to make decisions in highly structured, discrete settings of combinatorial problems. For example, taking a known Branch-and-Bound optimization algorithm for Mixed-Integer Linear Programs [46], one selects a variable on which to branch at any given step. The algorithm guarantees to converge to the optimal solution, but the number of variables to branch on and hence the running time of the algorithm highly depends on the choice at each step. ML can help in this setting to make an efficient selection at each step that would minimize the number of branches. This one example illustrates that many of the existing combinatorial problems can be facilitated by an ML model trained on the many instances of the problem that could be generated easily. In another example, ML can learn a solution to a combinatorial problem from scratch without the use of any expert combinatorial algorithm. For instance, if we are seeking the longest path in a graph (NP-hard problem), we may start exploring the neighbors randomly, to the point when we do not satisfy the conditions of the problem anymore (no available neighbors) at which point we get a reward for the selections that we have made. If we parametrize our selection policy, we can update the weights according to the reward function, an approach known as policy gradient in Reinforcement Learning (RL) [47]. And yet, in a reverse manner, combinatorial optimization can be helpful for machine learning to decompose a problem into smaller subparts or by detecting sub-patterns that are later used by an ML method. In this thesis, we highlight this relationship between combinatorial optimization and machine learning in the case of problems formulated on graphs.

In its turn graphs share many similarities with images and text (e.g. its discrete nature) and one could hope to carry over successful techniques from these domains to graph theory. Indeed, a standard definition of a graph requires specifying the relationship between discrete objects, nodes, which is not too far from the image representation, where one can see nodes as pixels positioned in a specific order in an image and connected to the neighbors. Likewise, words that appear in a text can be seen as nodes having a relationship between each other if they are located with a context window in a text. An early attempt of applying deep learning to graphs derived from Convolutional Neural Networks (CNN) in computer vision [45], which extracts multi-scale localized spatial features to construct representations of the graph. Figure 2.1 shows challenges of using CNN directly on graphs. An image of a cat shows that each pixel has the same number of neighbors and therefore we can apply the same filter matrix to each of the pixels. On the

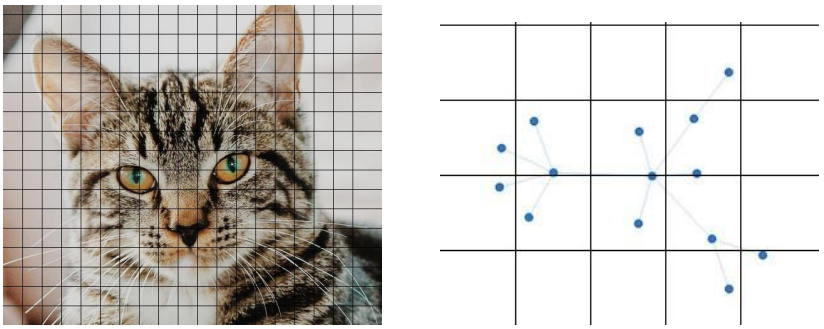


FIGURE 2.1: Comparison of application of CNN in Computer Vision and Graph Theory domains. On the left, the image that can be represented by a grid which are amenable to convolutional mask. Alternatively, CNN can operate on graphs if we define a unique weight matrix for all possible neighborhoods of each node.

other hand, nodes in a graph have a different number of neighbors and applying CNN to graph neighborhoods require some tricks. In particular, modern CNN learns the function over aggregated neighbors, rather than applying a weight transformation for each pixel directly. From this example, it is clear that there are two prerequisites to apply neural networks successfully on graphs. First is to define an aggregation function over nodes so that each node has exactly one local neighbor, i.e. aggregated form of its neighbors, to which a filter mask can be applied. Second is a transformation from categorical discrete objects, nodes, to continuous space on which deep learning models can operate. This second prerequisite is a known area of research called Representation Learning which concerns with the representation of objects in a vector space a.k.a. embedding space. In this thesis, we aim to develop models of machine learning to combinatorial optimization on graphs, a new, promising direction in graph theory research.

To summarize our motivation:

1. Combinatorial optimization problems on graphs have a long list of applications in the real world and therefore important to be solved quickly and efficiently;

2. Machine Learning operates on the settings, where a task is defined over available data and requires to find patterns in these data;

3. Combinatorial optimization and machine learning on graphs share many similarities and each area can benefit from another;

4. Machine learning has been successful in many domains such as image and text processing;

5. Machine learning has not been widely used in graph settings, where irregular data present challenges of a straightforward application of known deep learning methods to graphs;

6. Machine learning can be applied successfully to graphs if we determine a continuous representation of a graph.

## 2.2 Our approach

As discussed before a successful application of deep learning on graphs requires two steps, determining vector representation of the input data and designing machine learning models that operate on these vectors. In this chapter, we are concerned with the representation of the graphs, while we discuss the second in the applications of these embeddings and in the following sections.

An ease of representing data with graphs makes them very valuable asset in any data mining toolbox; however, the complexity of working with graphs led researchers to seek for new ways of representing and analyzing graphs, of which network embeddings have become broadly popular due to their success in several machine learning areas such as graph classification [48], visualization [49], and pattern recognition [50]. Essentially, network embeddings are vector representations of graphs that capture local and global traits and, as a consequence, are more suitable for standard machine learning techniques such as SVM that works on numerical vectors rather than graph structures. In the example of graph classification, one existing approach that has been applied successfully is based on kernel methods. Informally, a kernel is a function of two objects that quantifies their similarity and mathematically corresponds to the inner product of latent representations of the two objects in a Reproducing Kernel Hilbert Space (RKHS) [51]:

$$f(G_1, G_2) : G \times G \mapsto \mathcal{R} \tag{2.1}$$

Therefore once embeddings are obtained, one can construct a kernel matrix, where entries are kernel values applied to embeddings and then use it as a feature set for classification algorithms such as SVM. An important reason why kernels have become popular in machine learning community is that although latent representations could be high- or even infinite-dimensional the inner product can be calculated efficiently without performing any operations in RKHS. Ideally, a practitioner would like to have a *polynomial*-time algorithm that can convert different graphs into different feature vectors. However, such an algorithm would be capable of deciding whether two graphs are isomorphic [52], for which currently only quasipolynomial-time algorithm exists [53]. Hence, there are fundamental challenges in the design of the polynomial-time algorithm for network-to-vector conversion. Instead, a lot of research was devoted to the question

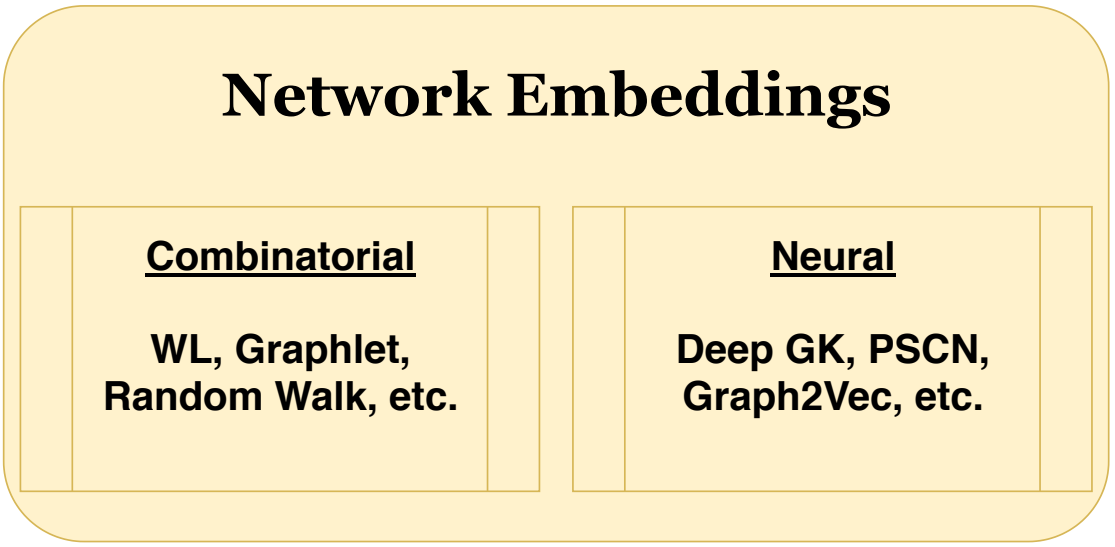



FIGURE 2.2: High-level overview of network embeddings. Combinatorial embeddings are obtained by computation of the combinatorial object in a graph. Neural embeddings are obtained by training a neural network that optimizes a function defined over the graph.

of designing network embedding models that are computationally efficient *and* preserve similarity between graphs.

Broadly speaking[1], network embeddings come from one of the two buckets, either based on the combinations of the statistics in a graph or that are optimized in a neural network. We call the first one ***combinatorial embeddings*** and the second ***neural embeddings***. Combinatorial embeddings traditionally appeared in graph kernel setting [54], where each graph is decomposed into discrete components, distribution of which is used as a vector representation of a graph [55]. Importantly, the general concept of combinatorial methods implies ad-hoc knowledge about the data at hand. For example, Random Walk kernel [54] assumes that graph realization originates from the types of random walks a graph has, whereas for Weisfeiler-Lehman (WL) kernel [8] the insight is in subtree patterns of a graph. For high-dimensional graph embeddings, combinatorial methods produce a sparse solution as only few substructures are common across graphs. This is known as *diagonal dominance* [56], a situation when a graph representation is only similar to itself, but not to any other graph.

On the other hand, a neural approach learns network embeddings by optimizing some form of objective function defined on graph data. Deep Graph Kernels (DGK) [56], for example, learns a positive semidefinite matrix that weights the relationship between

[1]Extensive overview of network embeddings is discussed in 2.3

graph substructures, while Patchy-San (PSCN) [57] constructs locally connected neighborhoods for training a convolutional neural network on. Neural approach implies learning *distributed* graph representations that have demonstrated promising classification results [57, 58].

**Our approach.** To this end, it must be clear that in graph classification settings *no free lunch theorem* still holds and there is no reason to believe that any set of embeddings is *universally* better than any other set of embeddings, i.e. performance averaged over all possible data-generating distributions is the same of any embeddings. However, at least we are interested in the case of data-generating processes that assign the same labels to topologically same graphs, in which case our embeddings should be the same for isomorphic graphs. This leads to the idea that vectors that can recover isomorphism of two graphs are powerful enough to work well in many practical applications, of which graph classification is a prominent example. We propose to use a natural graph object named *anonymous walk* as a base for learning combinatorial and neural network embeddings. Recent discovery [59] has shown that anonymous walks provide characteristic graph traits and are capable to reconstruct network proximity of a node *exactly*. In particular, distribution of anonymous walks starting at node $u$ is sufficient for reconstruction of a subgraph induced by all vertices within a fixed distance from $u$; and such distribution uniquely determines underlying Markov processes from $u$, i.e. no two different subgraphs exist having the same distribution of anonymous walks. This implies that two graphs with similar distributions of anonymous walks should be topologically similar. We, therefore, define combinatorial network embeddings on the distribution of anonymous walks and show an efficient sampling approach that approximates distributions for large networks.

To overcome sparsity of combinatorial methods, we design a neural approach that learns distributed representations on the generated corpus of anonymous walks via backpropagation, in the same vein as neural models in NLP [60, 61]. Considering anonymous walks for the same source node as co-occurring words in the sentence and graph as a collection of such sentences, the hope is that by predicting a target word in a given context of words and a document, the proposed algorithm learns the semantic meaning of words and a document.

The following applications when anonymous walk embeddings can be preferable to other existing network embeddings:

1. **Unsupervised learning.** Unlike other solutions that require labels during training, anonymous walk embeddings are learned in an unsupervised manner and thus network representations can be utilized in various ML tasks.

2. **Substructure embeddings.** While there are numerous approaches exist to learn embeddings of graph structures such as nodes [62] or subgraphs [63], we note that anonymous walks provide a unified corpus that serves to learn embeddings of nodes, subgraphs, and entire graphs at the same time.

3. **Complex graph structure.** Unlike approaches that are based on simple substructures such as random walks or shortest paths, our embeddings are based on a topologically-preserving graph object, anonymous walk. It has been shown that such substructures achieve higher classification results than linear kernels [8].

In the context of learning network representations we highlight the following contributions:

- Based on the notion of an anonymous walk that recovers graph structure exactly, we propose combinatorial network embeddings, for which we describe a sampling procedure to alleviate the time complexity of exact computation.
- By maximizing the likelihood of preserving network proximity of anonymous walks, we propose a scalable algorithm to learn neural network embeddings.
- On widely-used real datasets, we demonstrate that our network embeddings achieve state-of-the-art performance in comparison with other graph kernels and neural networks in graph classification task.

To summarize our approach:

1. Combinatorial embeddings decompose a graph into a set of combinatorial objects, which later are used to aggregate into a single vector,
2. Neural embeddings are obtained by training a neural network that optimizes a function over a given graph,
3. Both approaches have been successful in graph classification settings, with neural approaches taking an ever-increasing presence among embedding methods,
4. Combinatorial embeddings are efficient on its own right while becoming more and more present as a part in neural embedding approaches,
5. We motivate the use of anonymous walks as the main brick for our embeddings, motivated by a recent theorem that assures graph recovery property for anonymous walks,

6. We study the properties of the anonymous walks, its strong and weak properties, and propose approximations to overcome computational challenges,

7. Finally, we propose and compare experimentally combinatorial and neural embeddings based on anonymous walks.

## 2.3 Review of combinatorial and neural embeddings

Before we propose a new set of embeddings we discuss existing types of embeddings in detail. We start by formalizing graph embeddings (a.k.a. network embeddings or graph vector representations)[2].

**Definition 2.1** (**Graph embedding**)**.** Let function $\phi$ be a function such that for any graph $G$ it maps to a vector in $d$-dimensional space

$$\phi : G \mapsto \mathbb{R}^d. \tag{2.2}$$

Then $\phi(G)$ is called *graph embedding*.

Graph embeddings are useful on their own, without any application to downstream tasks. For example, if a function $\phi$ gives a bijection between graphs and vectors, then embeddings of the graph can be used alternatively as a data storage format for graphs instead of a traditional way of storing it as adjacency matrix. However, more often, embeddings are fed into more complex machine learning algorithms that solve a particular problem. Hence, we define a more general definition of graph representation problem.

**Definition 2.2** (**Graph Representation Problem**)**.** Let $\{G_i\}_{i=1}^N$ be a set of graphs in a dataset, $\mathcal{P}$ be a downstream problem on $\{G_i\}_{i=1}^N$, and $\mathbb{m}$ be a performance metric for algorithm $\mathcal{A}$ in a problem $\mathcal{P}$. Then *problem of graph representation* is to define a function

$$\phi : G \mapsto \mathbb{R}^d, \tag{2.3}$$

such that an algorithm $\mathcal{A}$ that takes as input $d$-dimensional vector $\phi(G)$ and outputs a solution to the problem $\mathcal{P}$, maximizes performance metric $\mathbb{m}$.

In particular we will be interested in a *graph classification problem* $\mathcal{P}$ with *accuracy* as a performance metric.

[2]Historically, graph embeddings were popularized after node and edge embeddings and unfortunately, graph embeddings are often used to denote the embeddings for nodes or edges. The proper way to denote vectors for nodes would be node embeddings.

**Definition 2.3** (**Graph Classification Problem**). Let Train $= \{(G_i, y_i)\}_{i=1}^{N_{train}}$ and Test $= \{(G_i, y_i)\}_{i=1}^{N_{test}}$ be train and test graph datasets, where $G_i$ is a graph and $y_i \in \mathcal{Y}$ is a *label* of a graph. *Graph classification problem* asks to train a function

$$f : G_i \mapsto \mathcal{Y}_\rangle, \tag{2.4}$$

that takes as input a graph $G$ and outputs a label of that graph such that the accuracy on unseen graphs in a test set is maximized:

$$\max_{\text{Test}} \frac{1}{N_{test}} [\![ f(G_i) = y_i ]\!]. \tag{2.5}$$

One approach to solving the classification problem is based on graph matching, i.e. finding an optimal correspondence between vertices of the two graphs that would preserve the adjacency between nodes. Once such matching is found one can proceed with the application of kernel-based methods that takes similarity between two graphs as an input. While totally legit, this approach is computationally expensive and often requires a solution to NP-complete problems as intermediate steps [64].

Somewhat similar to graph matching, graph kernels also deal with pairs of graphs but instead, they measure the similarity between graphs based not on the node matchings but on the topologies of the graphs and ensuing combinatorial statistics over the graphs. Graph kernels stem out from the kernel function defined next:

**Definition 2.4** (**Graph Kernel**). Let $\{G_i\}_{i=1}^{N}$ be a set of graphs. We define a graph kernel function $\mathcal{K} : G_i \times G_j \mapsto \mathbb{R}$ to be a function that has two properties:

1. Symmetric: $\mathcal{K}(G_i, G_j) = \mathcal{K}(G_j, G_i)$,
2. Positive-Semidefinite: $\sum_{i=1}^{N} \sum_{j=1}^{N} c_i c_j \mathcal{K}(G_i, G_j) \geq 0$, for any $c_i \in \mathbb{R}$.

Kernel functions have been known in mathematics for very long time due to important result by Mercer [65], which states that for any kernel function $\mathcal{K}$ defined over non-empty space $\mathcal{X}$ there exists a Hilbert space known as Reproducing Kernel Hilbert Space (RKHS) such that there exists an embedding function $\phi : \mathcal{X} \mapsto \mathcal{H}$ for which kernel function is equivalent to inner product of the embeddings in that space: $\mathcal{K}(\mathcal{X}_i, \mathcal{X}_j) = \phi(\mathcal{X}_i) \cdot \phi(\mathcal{X}_j)$. What is interesting is that the kernel function does not need to define an embedding function $\phi$ to get an inner product value. Moreover, there are known kernel functions for which a corresponding RKHS is infinitely-dimensional (for example Gaussian kernel) and hence embeddings are infinitely-dimensionally too [66]. This property can become handy in machine learning problems where the data are often non-linearly separable and

increasing dimensionality of the data may be a quick win for different classifiers such as SVM [67]. Note however that except for very few approaches (for example, Random Walk Graph Kernel [54, 68]) majority of graph kernels first compute graph embeddings *explicitly* using combinatorial methods discussed next and then use these embeddings with kernel SVM to train and classify graphs.

The developments in graph kernels led researchers to seek new ways to obtain embeddings that can show state-of-the-art results in graph classification. This, in turn, opened a new era of graph embeddings that spotlights some of the beautiful results in the intersection of machine learning and graph theory. We discuss these ideas next.

### 2.3.1 Combinatorial graph embeddings

Combinatorial embeddings are a rich family of methods, each of which is associated with some combinatorial objects such as random walk or shortest path that decomposes a graph into a collection of such objects. One of the constraints on the combinatorial objects is that the resulted graph embedding will be graph invariant:

**Definition 2.5** (**Graph invariant**)**.** If for any isomorphic graphs $G_1$ and $G_2$ the embeddings are equivalent, i.e. $\phi(G_1)$ and $\phi(G_2)$, then embedding function $\phi$ is called *graph invariant*.

This definition requires a definition of isomorphism between two graphs; however, we postpone this definition until later, keeping in mind that isomorphic graphs are those with exactly the same topology. We denote isomorphism between two graphs as $G_1 \cong G_2$. In this case, for graph invariant it is valid:

$$G_1 \cong G_2 \implies \phi(G_1) = \phi(G_2), \tag{2.6}$$

for any two graphs $G_1$ and $G_2$.

One example of graph invariant is the number of vertices of a graph. While its obviously a graph invariant, its not a particularly useful graph invariant for graph classification, because there are $2^n$ graph with $n$ vertices, and the number of vertices gives little information about the class of the graph in most datasets.

A more useful example of graph invariant is a sorted degree sequence. In particular, let $\{G_i\}_{i=1}^N$ be a dataset with $N$ and $d$ be a maximum degree among all nodes in all graphs in this dataset. Then for any graph $G$ let $v_G \in R^d$ be an embedding of graph $G$ such that the position $i$ is occupied by a count of the nodes with degree equals $i$. This

drastically reduces the number of possibilities for graphs with an embedding $v$. Figure 2.3 illustrates two different regular graphs that obviously have the same embedding $v$.

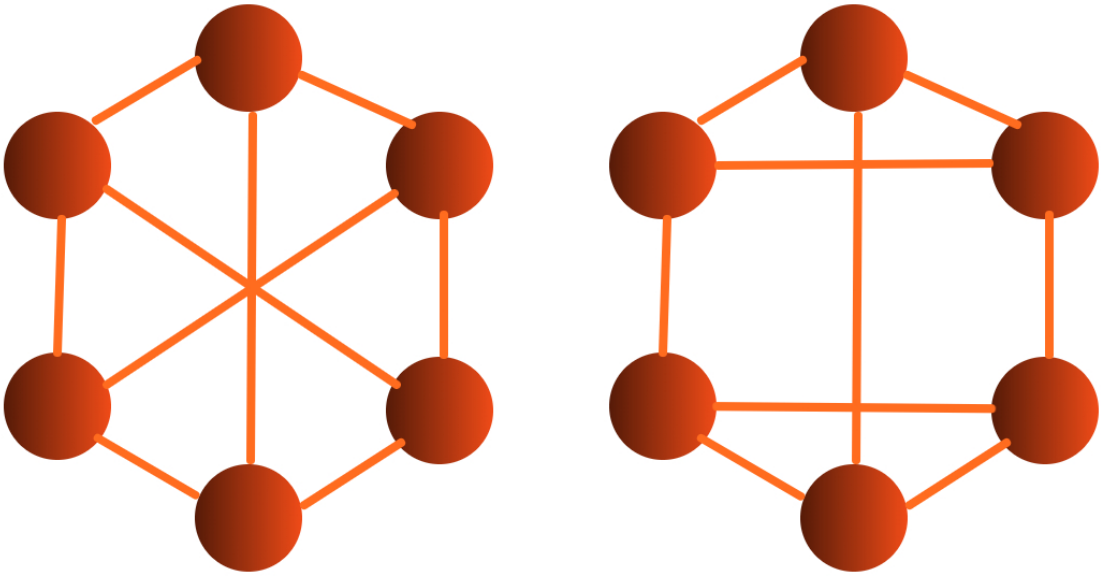


FIGURE 2.3: Example of two non-isomorphic graphs that have the same degree-based embedding $v = [0, 0, 6]$. The graph on the left has two triangles, while the right one does not have any triangles. Each cell in the embedding indicates the number of nodes with the corresponding degree. Graphs are 3-regular, hence only one value is non-zero.

An example of a function $\phi$ that is not graph invariant is the following. Let $A_G$ be an adjacency matrix of graph $G$. Let $v$ be a vectors composed of rows in $A_G$ such that the first entries are occupied by the first row of $A_G$, the second by the second row, and so on. It is easy to see that the resulted vector is not a graph invariant in the general case. Indeed, let's take a graph where there are two nodes with a different degree. For the first case, $A_{G_1}$ will put a larger degree node to the first node, while for the second case $A_{G_2}$ the smaller degree node will be the first node. Since there are different number of ones in rows for these two nodes, it's clear that the first entries of $v_{G_1}$ are different from the first entries of $v_{G_2}$ and hence the embeddings are not the same.

Note that there are different graphs for which we have the same graph invariant in general, in other words $G_1 \ncong G_2$ and $\phi(G_1) = \phi(G_2)$. The case for which equality of embeddings implies isomorphism of the graph is defined by complete graph invariant.

**Definition 2.6** (**Complete graph invariant**)**.** If for any pair of graphs $G_1$ and $G_2$ holds equation:

$$\phi(G_1) = \phi(G_2) \iff G_1 \cong G_2$$

then a function $\phi : G \mapsto R^d$ is called complete graph invariant.

Note that finding a complete graph invariant solves a problem of graph isomorphism if the dimension $d$ is relatively small compared to the size of the graphs. There are very few

known complete graph isomorphism known. A prominent example is a canonical form by graph isomorphism solvers such as Nauty [69–71]. In this thesis, we provide another complete graph invariant based on the new combinatorial object called anonymous walk.

Now, after we defined graph invariant we describe some of the proposed combinatorial embeddings based on them. Informally one can see the history of graph embeddings evolving from topological descriptors to graph kernels to neural networks.

The first combinatorial embeddings appeared in the domain of bio- and chemoinformatics, where the molecules are represented as a graph. A few of the so-called *topological descriptors*, which often is just a number that represents the whole graph, have been proposed to solve similarity tasks such as nearest neighbor search to the graph query. Wiener index [72, 73] is one of such topological descriptors, which equals to the sum of the shortest paths between all pairs in a graph:

$$W(G) = \sum_{v_i, v_j \in V} D_{ij}, \tag{2.7}$$

where $D_{ij}$ is the shortest path between vertex $v_i$ and $v_j$. Another example is Hosoya index [74], which is defined as the number of ways to select edges such that none of the selected is overlapping with other selected edges:

$$Z(G) = \sum_{k=0}^{M} a(G, k), \tag{2.8}$$

where $a(G, k)$ is the number of ways to select $k$ non-adjacent edges in a graph and $M$ is the number of edges in the graph. There are hundreds of proposed topological descriptors, which were shown to correlate with one property or another of the molecules. For an extensive overview of such descriptors, refer to the book [75]. One can see topological descriptors as first decomposing a graph into combinatorial objects and then aggregating these objects into a single value. While topological descriptors have a rich history of studies that show their relationship to some properties of chemoinformatics data, it is unlikely to see a single number working efficiently in a broad range of applications.

A broader framework for computing a combinatorial embedding is related to R-convolution framework [55]. The principal idea of R-convolution stems out of topological descriptors but introduces two distinctions. First, in R-convolution there is a procedure to distinguish the objects in which the graph is decomposed. For example, in the case of Wiener index (2.7), shortest paths of the same length are difficult to compare with each other: different paths contain different nodes and there is no notion of order between nodes,

hence one has to introduce some means of comparison between two paths (say the length or the number of specific node labels). The second distinction is the number of distinct objects of the decompositions. In the majority of cases, objects are represented in a great variety in the graph and therefore its combinatorial embedding is represented as an array of multiple numbers, each corresponding to its own objects. Often the length of the array is so big due to the exponential nature of the graph decomposition that one has to sacrifice the precision of the R-convolution to obtain the reasonable size of the array. This idea paved the way for the development of graph kernels, where a new R-convolution results in a new kernel.

For example, in graphlet kernel [6] embeddings are based on the decomposition with graphlets (a.k.a. motifs), i.e. all possible subgraphs of given size. Procedure to compute graphlet kernel is the following:

1. Select size of graphlet $k$,
2. Enumerate all possible graphlets of size $k$,
3. Compute the count of each graphlet in a graph,
4. Output a vector of counts in some predetermined order.

Since there are $\binom{n}{k}$ possible motifs, the total running time requires $\mathcal{O}(n^k)$ steps, with a constant running time to perform inner product between two graphs. This clearly prohibitive for large graphs and the authors proposed a concentration inequality that approximates the true distribution within $\varepsilon$ factor. One of the strong sides of the graphlet kernel is that it is based on the graph reconstruction conjecture [9, 76] that states that all possible subgraphs of size $n-1$ of the original graph with $n$ nodes are enough to reconstruct the original graph. Even though, a graphlet kernel is summarized version of the collection of $k$-size subgraphs, this conjecture, has been confirmed for different types of graphs [9] and for $n \leq 11$ is a strong indication that graphlet distributions capture well the topology of the graph. Our proposed solution based on anonymous walks is also based on the theorem that states that the distribution of the anonymous walks is enough to reconstruct the graph.

Another type of kernels is based on random walks in a cross product $A_\times$ between two graphs [77]. Cross product of two graphs is determined as following:

$$V_\times = \{(v_i^1, v_j^2) : v_i^1 \in G_1 \text{ and } v_j^2 \in G_2\},$$
$$E_\times = \{((v_i^1, v_j^2), (v_k^1, v_l^2)) : \text{ iff } (v_i^1, v_k^1) \in G_1, (v_i^2, v_k^2) \in G_2\}.$$

This creates a single graph with $\mathcal{O}(n^2)$ nodes and $\mathcal{O}(n^4)$ edges. A graph kernel is defined then as the number of paths of different lengths in a cross graph:

$$\mathcal{K}(G^1, G^2) = \sum_{k=0}^{\infty} \mu(k) q_{\times} A_{\times}^k p_{\times}, \tag{2.9}$$

where $\mu(k)$ is a coefficient that guarantees convergence of the sum, $q_{\times}$ and $p_{\times}$ are the initial and stopping probability distribution, and $A_{\times}$ is the adjacency matrix of a cross graph. Based on the coefficients $\mu(k)$ one can derive a closed-formula for the series (2.9) [52, 78, 79]. There are two peculiarities about this graph kernel. First is that it does not require the computation of explicit embeddings between two graphs. In fact, graphs $G_1$ and $G_2$ are never used in the definition of the kernel and all of the work is performed in the cross graph. The second trait of this kernel is that it implicitly compares random walks between two graphs, even though walks may have different node labels. This is possible because the random walks computed in the cross product, bypassing computation of distinct random walks in two graphs. Random walk kernel requires $\mathcal{O}(n^6)$ time in a naive implementation and can be computed in $\mathcal{O}(n^3)$ using Kronecker products [54, 68, 77]; however, it's still computationally expensive for large graphs.

One of the most both fast and performing graph kernels is based on the Weisfeiler-Lehman algorithm for isomorphism testing. Weisfeiler-Lehman (WL) kernel [8] computes explicit graph embeddings based on relabeling the nodes of a graph iteratively, based on the local neighborhoods. Initially, all labels of the nodes are the same. Then each iteration involves three steps:

1. For each node $v$ sort the labels of the neighbors and append its own label to obtain a list of labels $\mathcal{L}(v)$,
2. Hash each list of labels $\mathcal{L}(v)$ into a new label $l$,
3. Relabel a previous label of $v$ with a hashed label.

After initial labeling, in the first iteration, each node will get a new label that will correspond to the degree of the node. In the second iteration, each node will get a new label according to the distribution of degrees of its neighbors, and so on. Informally it means that each node propagates its local topology to other nodes in the network; hence after several iterations will know all the local topologies that exist in a graph. Then WL embedding of the graph is the distribution of labels after some number of iterations. The original algorithm by Weisfeiler-Lehman for graph isomorphism testing has the following

property [80]: if after $n$ iterations of Weisfeiler-Lehman procedure WL embeddings are the same for two graphs, then they are most likely to be isomorphic; while if at some iteration $i \leq n$ the distributions of labels is different, then the graphs definitely are not isomorphic. The total running of the Weisfeiler-Lehman procedure with $k$ iterations for graphs with $m$ edges is $\mathcal{O}(km)$, which it makes it very fast. Weisfeiler-Lehman kernel has shown competitive results in many graph classification datasets making it a strong baseline.

As could be noted from the above the crux of the design of combinatorial embedding lies at the selection of combinatorial graph object that sufficiently preserves information about the topology of the graph when the graph is decomposed by this object. There are kernels based on shortest paths [81, 82], cycles and trees [83–85], group theoretical invariants [86, 87], and many others [88–91]. For an extensive overview of the graph kernels and combinatorial embeddings one can refer to the surveys [77, 92, 93].

### 2.3.2 Neural graph embeddings

In the previous section, we saw that combinatorial embeddings are based on some type of combinatorial object in the graph: distribution of the types of this object or some transformation of it is what constitutes an embedding. In the neural embedding case, the situation is different in a radical way. Instead of defining an object and then operating on it, we randomly will initialize embeddings and then learn these embeddings according to the topology of the network. This end-to-end procedure replaces the choice of the combinatorial object with the choice of selecting the right training procedure (model architecture, loss function, hyperparameters, etc.).

One of the examples of learning neural embeddings is of *Deep Graph Kernels* [56]. As the name suggest it is still a graph kernel, but it adds an additional relationship factor that solves a particular challenge of graph kernels, namely *diagonal dominance*. Diagonal dominance is an empirical observation that when the size of the combinatorial object is large enough, then the number of such objects tends to zero for all graphs, except some single graph. For instance, in the case of graphlet kernels [6] if the size is big (e.g. $k \geq 10$) the number of graphlets can become very high ($\mathcal{O}(2^k)$) and computing the distribution of the objects in a graph will reveal that some specific graphlet appears more common than others. Moreover, other graphs will not have these graphlets often, preferring some other graphlets. This will lead to a situation when the similarity between the two graphs will be close to zero, even if the graphs are of the same class. To address this, the authors proposed to multiply graph kernel but a positive semidefinite matrix that encodes the relationship between the objects:

$$\mathcal{K}(G_1, G_2) = \phi(G_1)\mathcal{M}\phi(G_2). \tag{2.10}$$

To obtain the matrix $\mathcal{M}$ the authors consider co-occurrence between two types of objects if they appear at the same time during the computation of these objects. This corpus is considered similarly as the corpus of words to which we can apply a language model that learns embeddings for these objects, for example, the Skip-gram word2vec model [94]. Once the embeddings learned one can find similarity between different objects and compute the matrix $\mathcal{M}$. While the approach is useful for resolving diagonal dominance, large $k$ sizes are rarely used in practice because of the computation limits. Another approach *graph2vec* [95] based on also language modeling has been recently proposed that learns the graph embedding directly, without passing to graph kernel. In their case, they sample a corpus of rooted subgraphs (also used in WL graph kernel) that belong to a graph and then consider this corpus as a text on which one can learn embeddings. Their learnable model is doc2vec [60], which considers additionally an embedding of a document during the learning. In this case, not only embeddings of the rooted subgraphs are obtained, but also of the graph. In this thesis, we propose to have a similar model but which is applied to anonymous walks, which can be considered as sentences where nodes are the words.

A different approach which is not based on corpus generation but rather on direct learning of the node features is known as Graph Neural Networks (GNN). Informally, GNN is the application of deep learning models on the nodes neighborhood level and includes various forms of models such as RNN, CNN, Attention. In Figure 2.4 In the case of the CNN graph convolutional algorithm aggregates the information of the neighbors according to some weight functions, this mechanism known as message-passing neural network [96]. Graph convolutional networks in turn fall into two paradigms, *spectral-based* or *spatial-based methods*. In the first case, the embeddings are obtained from the perspective of noise removal in the graph. Its ideas originate from signal processing and attempt to update initial node embeddings by multiplying them by a learnable filter.

More formally, it is known that Laplacian matrix $L = I_n D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$ is real symmetric positive semidefinite that can be factored using SVD as such: $L = U\Lambda U^T$, where matrix of $U$ is the matrix of eigenvectors and $\Lambda$ is a diagonal matrix of eigenvalues. If we define a filter as $g \in \mathbb{R}^n$ and vector embedding $x$, then we can apply this filter to the original graph signal:

$$x \star g = U(U^T x \odot U^T g). \tag{2.11}$$

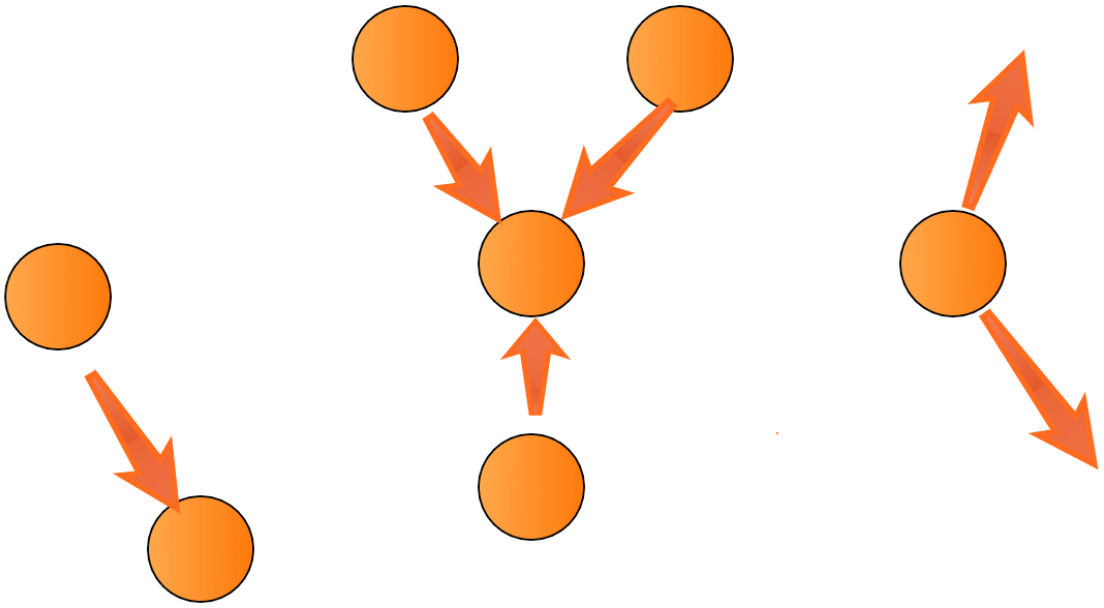


FIGURE 2.4: Message-passing algorithm propagates embeddings from its neighbors towards the node aggreating them to produce a new embeddings.

This corresponds to the Hadamard product between two Fourier transforms of the embeddings $x$ and $g$ and then applying reverse Fourier transform [97]. If we denote a filter as $g_\theta = diag(U^T g)$ then a graph convolution is defined by:

$$x \star g_\theta = U g_\theta U^T x. \tag{2.12}$$

Different definitions of the filter $g_\theta$ defines different GNN. For instance, Spectral Convolutional Neural Network [97] defines the filter $g_\theta = \Theta$, where $\Theta$ is learnable parameter matrix. In a *ChebNet* [98], the authors propose to approximate the filter $g_\theta$ by a Chebyshev polynomial, i.e. $g_\theta = \sum_{i=0}^{K-1} \theta_i T_k(\hat{\Lambda})$, where $\hat{\Lambda}$ is the normalized matrix $\Lambda \in [-1, 1]$, the Chebyshev polynomials are defined as $T_k(x) = 2xT_{k-1}(x)T_{k-2}(x)$ with $T_0(x) = 1$ and $T_1(x) = x$. A further approximation of ChebNet, called *GCN* [99] assuming that there is a single term in Chebyshev polynomial. In this case:

$$x \star g_\theta = \theta(I_n + D^{-\frac{1}{2}} A D^{\frac{1}{2}})x. \tag{2.13}$$

In the case of GCN, the convolutional function is propagated only within a first-level neighborhood and hence this type of spectral-based methods can also be seen as a spatial-based methods, that uses information only to its neighbors. In particular, the function for the update for the parameters of the node is:

$$h_v = f((\sum_{v \in N(v) \cup v} \hat{A}_{v,u} x_u)\Theta) \tag{2.14}$$

There are many other models that propose new graph filters [100, 101]; however, spatial-based methods that we discuss next offer more efficiency and flexibility. Indeed spectral methods require to do SVD which is costly and requires re-computation if the graph structure slightly changes due to the modification of the eigenbases. Also, spatial-based methods are required to work on undirected graphs due to the properties of the Laplacian, while spatial-based methods can work with any types of graphs (directed, dynamic, knowledge, etc.).

The spatial-based method update iteratively the node embeddings based on the neighbors embeddings at the current iteration. This draws inspiration from the convolutional mask is a shared filter applied to all the neighboring pixels of the current one. The distinction from the images though is that nodes may have a different number of neighbors and therefore one cannot have its own separate weight associated with each neighbor. Instead, spatial-based methods aggregate neighbors into a single embedding such that aggregation happen in order-invariant way. One of the first attempts NN4G [102] was using a linear transformation of the neighbors' embeddings to produce an updated embedding:

$$h_v^{(k)} = f(x_v W^{(k-1)} + \sum_{i=1}^{k-1} \sum_{u \in N(v)} h_u^{(k-1)} \Theta^{(k-1)}), \tag{2.15}$$

where $f$ is an activation function. In its form, the embeddings from the neighborhoods are summed with the same weights and then summed with the current (weighted) embedding of the node. Note also that in NN4G embeddings are dependent on the whole history of user embeddings, which limits the number of layers one can use in practice due to the volume of data.

In the Graph Isomorphism Network (GIN) [103], the authors propose a simple convolution that theoretically guarantees to have embeddings that as powerful as Weisfeiler-Lehman test of isomorphism. Their update function is the following:

$$h_v^{(k)} = \sigma(((1 + \varepsilon^k) h_v^{k-1} + \sum_{u \in N(v)} h_u^{(k-1)} W^{k-1}), \tag{2.16}$$

They showed that when activation and readout functions are injective (for example, sigmoid and sum respectively) then the models are as discriminative as WL test.

In Graph Attention Network [104], the weights are computed for every edge dictating the importance of each relationship of the neighbor to the update of the current node. The mechanism is similar to the language model Transformer [105], where multi-layer multi-head attention is applied to update the embeddings of the words. The update rule is the following:

$$h_v^{(k)} = \sigma\Big(\sum_{u \in N(v) \cup v} \alpha_{u,v} W^{(k-1)} h_u^{(k-1)}\Big), \tag{2.17}$$

where attention weights $\alpha_{u,v}$ sum up to one for $N(v) \cup v$.

For all Graph Neural Networks, the embeddings are defined on node-level, while the graph embedding is obtained with an additional graph pooling layer. A simple strategy for defining a pooling layer is to apply mean, max, or sum operation on the last layer embeddings across all nodes, for example:

$$h_G = \sum_{v \in V} h_v^{(K)}. \tag{2.18}$$

In [103], the authors show that the pooling layer with the sum has the most expressive power in comparison to mean and max aggregators. Moreover pooling methods such as based on attention [104] or LSTM [106] have been also proposed.

A somewhat different approach is based on generative models that have encoder-decoder structure [107–109]. For example, in [107] the authors propose to use Variational Auto-Encoder that has a two-layer GCN encoder and a simple sigmoid function over the inner product of two latent variables to decode latent distributions back to the graph (Figure 2.5). In particular, there are two GCNs that model $\mu_i$ and $\sigma_i$ respectively, which are used to draw a latent variable from the normal distribution:

$$\begin{aligned} q(z_i|X, A) &= \mathcal{N}(z_i|\mu_i, \mathrm{diag}(\sigma_i^2)), \text{where} \\ \boldsymbol{\mu} &= \mathrm{GCN}_\mu(X, A) \\ \log \boldsymbol{\sigma} &= \mathrm{GCN}_\sigma(A, X) \end{aligned} \tag{2.19}$$

In the equation above, GCN is a two-layer convolutional network that takes adjacency matrix $A$ and a feature matrix $X$ as an input and produces vectors $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$.

Generative model outputs a graph object given a latent variable $\boldsymbol{z}$:

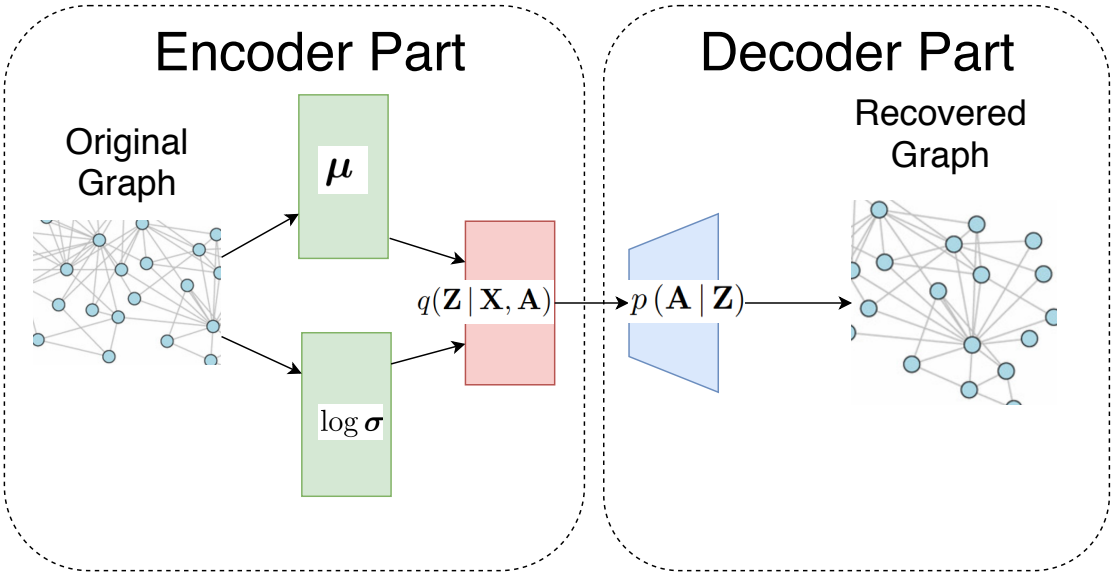



FIGURE 2.5: Graph Variational Auto-Encoder. A graph is encoded by two GCNs that produce two vectors $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$. Then a latent variable $z$ is drawn from a normal distribution $\mathcal{N}$. A latent variable is later decoded to a new recovered graph, where adjacency is determined by a dot product of two latent variables. The model is trained by maximizing log-likelihood of the recovered graph and minimizing the KL-divergence between a latent distribution $q$ and a prior normal distribution.

$$p(A_{ij} = 1|z_i, z_j) = \sigma(z_i \cdot z_j) \tag{2.20}$$

In this way, the adjacency matrix is determined by the inner product of two variables that correspond to nodes $i$ and $j$. This encoder-decoder process is trained over a graph that optimizes variational lower bound:

$$\mathcal{L} = \mathbb{E}_q[\log p(A|Z)] - \mathrm{KL}[q(Z|X, A)\|p(Z)] \tag{2.21}$$

This loss has two terms: the first is a reconstruction loss that indicates the error between the decoded graph and the original one; the second term is regularization that minimizes the error between two distributions, one encoded distribution of latent variables and one prior distribution which is Gaussian distribution. After the model is trained one can see $q(z|X, A)$ as a *probabilistic embedding* of a graph. Other types of probabilistic embeddings include probabilistic embeddings based on other generative models such as Generative Adversarial Networks [110, 111] and Auto-regressive models [112].

Graph Neural Networks is an active area of research due to its effectiveness, flexibility, and end-to-end nature. They are also quite well theoretically backed up, by the results from graph isomorphism literature [103] and universal approximation capabilities within

certain criteria [113–115]. For an extensive overview of the recent methods please refer to the surveys [116, 117].

## 2.4 Anonymous walks

Random walks are the sequences of nodes, where each new node is selected independently from the set of neighbors of the last node in the sequence. Normally states in a random walk correspond to a label or a global name of a node; however, for reasons described below such states could be unavailable. Yet, recently it has been shown that anonymized version of a random walk can provide a flexible way to reconstruct a network even when global names are absent [59]. We next define the notion of anonymous walk.

**Definition 2.7.** Let $s = (u_1, u_2, \ldots, u_k)$ be an ordered list of elements $u_i \in V$. We define the positional function $pos$: $(s, u_i) \mapsto q$ such that for any ordered list $s = (u_1, u_2, \ldots, u_k)$ and an element $u_i \in V$ it returns a list $q = (p_1, p_2, \ldots, p_l)$ of all positions $p_j \in \mathbb{N}$ of $u_i$ occurrences in a list $s$.

For example, if $s = (a, b, c, b, c)$, then $pos(s, a) = (1)$ as element $a$ appears only on the first position and $pos(s, b) = (2, 4)$.

**Definition 2.8 (Anonymous Walk).** If $w = (v_1, v_2, \ldots, v_k)$ is a random walk, then its corresponding *anonymous walk* is the sequence of integers $a = (f(v_1), f(v_2), \ldots, f(v_k))$, where integer $f(v_i) = \min pos(w, v_i)$.
We denote mapping of a random walk $w$ to anonymous walk $a$ by $w \mapsto a$.

For instance, in the graph of Fig. 2.6 a random walk $a \to b \to c \to b \to c$ matches anonymous walk $1 \to 2 \to 3 \to 2 \to 3$. Likewise, another random walk $c \to d \to b \to d \to b$ also corresponds to anonymous walk $1 \to 2 \to 3 \to 2 \to 3$. Conversely, another random walk $a \to b \to a \to b \to d$ corresponds to a different anonymous walk $1 \to 2 \to 1 \to 2 \to 3$.

Intuitively, states in anonymous walk correspond to the first position of the node in a random walk and their total number equals the number of distinct nodes in a random walk. Particular name of the state does not matter (so, for example, anonymous walk $1 \to 2 \to 3$ would be the same as anonymous walk $3 \to 1 \to 2$); however, by agreement, anonymous walks start from 1 and continue to name new states by incrementing the current maximum state in an anonymous walk.

**Rationale.** From the perspective of a single node, in the position of an observer, the global topology of the network may be hidden deliberately (e.g. social networks often

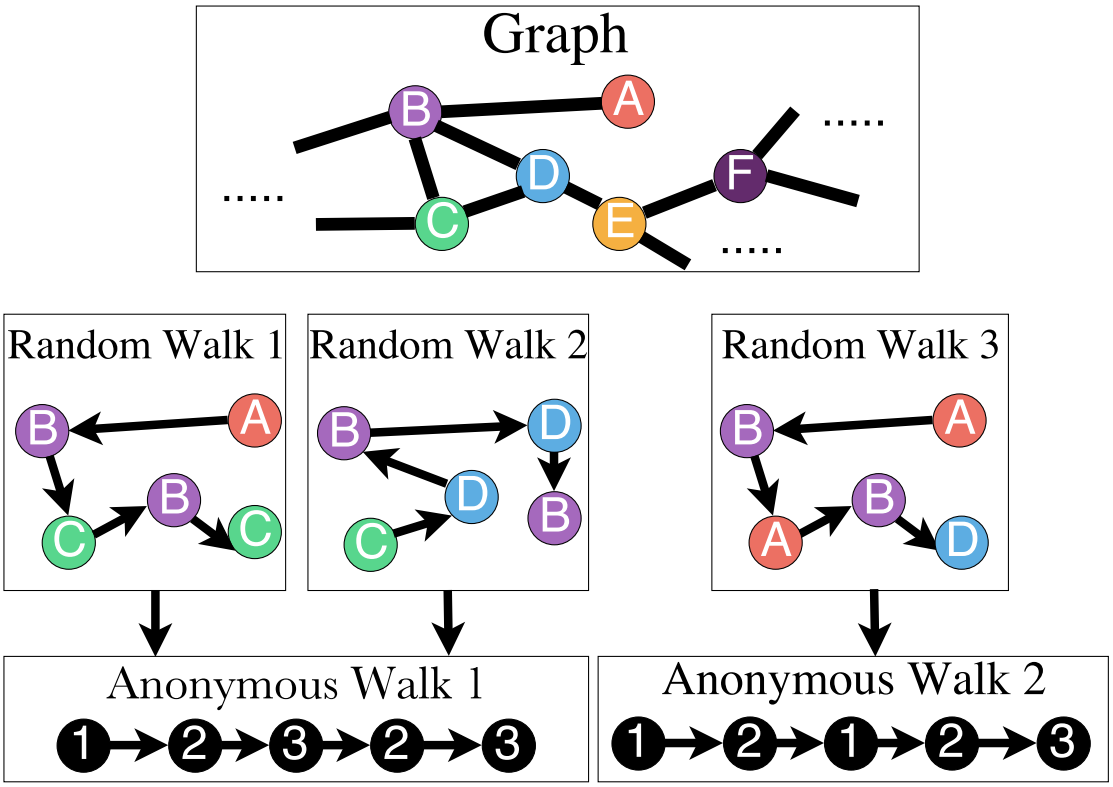



FIGURE 2.6: An example demonstrating the concept of an anonymous walk. Two different random walks 1 and 2 of the graph correspond to the *same* anonymous walk 1. A random walk 3 corresponds to *another* anonymous walk 2.

restrict outsiders to examine your friendships) or otherwise (e.g. newly created links in the world wide web may be yet unknown to the search engine). Nevertheless, an observer can, on his own, experiment with the network by starting a random walk from itself, passing the process to its neighbors and recording the observed states in a random walk. As global names of the nodes are not available to an observer, one way to record the states *anonymously* is by describing them by the first occurrence of a node in a random walk. Not only are such records succinct, but it is common to have privacy constraints [118] that would not allow recording a full description of nodes.

Somewhat remarkably, [59] show that for a single node $u$ in a graph $G$, a known distribution $\mathfrak{D}_l$ over anonymous walks of length $l$ is sufficient to reconstruct topology of the ball $B(u, r)$ with the center at $u$ and radius $r$, i.e. the subgraph of graph $G$ induced by all vertices distanced at most $r$ hops from $u$. For the task of learning embeddings, the topology of the network is available and thus the distribution of anonymous walks $\mathfrak{D}_l$ can be computed precisely. As no two different subgraphs can have the same distribution $\mathfrak{D}_l$, it is useful to generalize the distribution of anonymous walks from a single node to the whole network and use it as a feature representation of a graph. This idea paves the way to our feature-based network embeddings.

### 2.4.1 Graph isomorphism test

We focus on undirected connected graphs. It is well known that directed and/or disconnected graphs are equivalent to the class of undirected connected graphs in GI problem. Let $G = (V, E)$ be a graph on the set of $n$ vertices $V$ and the set of $m$ edges $E$. Let $w = (v_0, v_1, \ldots, v_l)$ be a random walk that has $l + 1$ vertices and $a$ be anonymous walk according to the definition 2.8.

In other words, anonymous walks can be seen as an anonymized version of random walks, where each label is replaced with its first occurrence. [119] shows that given only a set of all anonymous walks of long enough length for any source node is sufficient to reconstruct the exact topology of a graph surrounding that node. As we show next their algorithm returns an instance of anonymous covering walk, for which reconstruction property always holds.

Let $\mathcal{D}_{\leq l}$ be a set of all anonymous walks of length up to $l$ in graph $G$, which we call *support set* and its elements as *support anonymous walks* (or just support if it's clear from the context). Given anonymous walk $\alpha$ that has $k$ distinct values, we define its *supporting graph* $G_\alpha = (V, E)$, where $V = \{1, 2, \ldots, k\}$ and $(a, b) \in E$ if and only if a consecutive pair $(a, b)$ appears in anonymous walk $\alpha$. Let $w$ be a random walk that traverses each edge at least once, then the corresponding anonymous walk $\beta$ is called *covering anonymous walk* or cover. Finally, a property $p$ of graph such that $p(G) = p(G') \Rightarrow G = G'$ is called a *complementary graph invariant.*

We observe the following result that represents new canonical labeling of the graph.

**Theorem 2.9.** *Let $S = \bigcup_{v \in V} \beta$ be a union of all covers $\beta$ of length $l$ in graph $G$ for all vertices $v \in V$. Then $S$ is a complete graph invariant and for any $\beta \in S$ graph $G_\beta$ is a canonical labeling of $G$, i.e. $G_\beta \cong G$.*

*Proof.* Consider a covering anonymous walk $\beta$, i.e. an anonymous walk that traverses each edge at least once. As $\beta$ exists in graph $G$, let $w$ be any corresponding random walk in $G$. Note that each vertex $v$ in $G$ is present in a random walk $w$ and it corresponds to some element $i$. Then a function $\phi(v) = i$ is a bijection between vertices and nodes in $\beta$.

Note that $(u, v)$ is an edge of graph $G$ if and only if a pair $(\phi(u), \phi(v))$ is an edge in graph $G_\beta$. Hence, function $\phi$ is an isomorphism for $G$ and $G_\beta$.

If vertices $v_1 \in V_1$ and $v_2 \in V_2$ of graphs $G_1$ and $G_2$ have the same cover $\beta$, then the graphs are isomorphic, i.e. $G_1 \cong G_2 \cong G_\beta$. Therefore, all graphs that have a cover $\beta$ are isomorphic to $G_\beta$ and by definition, $G_\beta$ is a canonical labeling.

**Algorithm 1** Test of Graph Isomorphism

1: **Input:** graphs $G_1$, graphs $G_2$
2: **Output:** True/False if graphs are isomorphic
3: Compute $S_1 = \bigcup \beta$ for graph $G_1$
4: Compute $S_2 = \bigcup \beta$ for graph $G_2$
5: **if** $S_1 \bigcap S_2 = \emptyset$ **then**
6: **return** False
7: **else**
8: **return** True
9: **end if**

Let $S_v = \bigcup \beta_v$ be all covers of length $l$ that start from the vertex $v$. If two vertices $v_1 \in V_1$ and $v_2 \in V_2$ have the same $S_{v_1} = S_{v_2} = S_v$, then any $\beta \in S_v$ is sufficient to obtain canonical labeling $G_\beta$. On the other hand, if two graphs are not isomorphic, there is no $\beta$ that is the same for two vertices (otherwise, it will become a canonical labeling). Finally, obtaining the union of all covers $\beta$ across all nodes guarantees that the sets $S$ are the same for isomorphic graphs and non-overlapping for non-isomorphic graphs.

□

Essentially if one obtains two sets of all covering anonymous walks, $S(G)$ and $S(G')$, of length $l$ for two graphs $G$ and $G'$, then an intersection between the two sets reveals if graphs are isomorphic. Moreover, the intersection is either the full set $S = S(G) = S(G')$ in the case of isomorphism or it is empty set in the case that two graphs are non-isomorphic. If two graphs are isomorphic then taking any cover $\beta$ gives canonical labeling of a graph.

Algorithm 1 tests two graphs for isomorphism. It first computes two complete graph invariants $S(G_1)$ and $S(G_2)$ and then compares their intersection. By Theorem 2.9 $S_1$ and $S_2$ are complete graph invariants and therefore the Algorithm 1 always produces correct answer. Note that the number of covers $\beta$ in the set $S$ can be extremely large. The following theorem establishes an upper bound of elements in $D_l$.

**Theorem 2.10.** *Let $D_l$ be a set of all anonymous walks with $l$ nodes in a complete graph $K_l$ with $l$ vertices. Then $|D_l| = B_{l-1}$, where $B_{l-1}$ is the $(l-1)$-th Bell number.*

*Proof.* Note that a complete graph $K_l$ with $l$ vertices contains all possible anonymous walk sequences with $l$ nodes, because any vertex $u$ is connected with $v$ and hence any node in anonymous walk $i$ is connected with any node $j$. An anonymous walk can be seen equally as coloring of a map consisting of a row of $l$ elements, where adjacent regions cannot have the same color. The number of such non-isomorphic colorings is known to be $B_{l-1}$ [120, 121]. □

**Algorithm 2** Maximal Covering Anonymous Walk

**Input:** graph $G$, vertex $v$
**Output:** cover $\beta$
Initialize stack $S = [1]$, $\beta = [1]$, $m = \#$ edges
**while** $S \neq \emptyset$ **do**
  last $= S[-1]$
  $x = \max(\beta) + 1$
  # phase 1: explore not visited nodes
  $\beta_{try} = replace(\beta, last, x)$
  **if** $\beta_{try}$ exists in $G$ **then**
    $\beta = \beta_{try}$
    $S.append(x)$
  **else**
    $S.pop()$
  **end if**
  # phase 2: interconnect existing elements in $\beta$
  $U = [x-1, x-2, \ldots, last+1]$
  **for all** $u \in U$ **do**
    $\beta_{try} = replace(\beta, last, u)$
    **if** $\beta_{try}$ exists in $G$ **then**
      $\beta = \beta_{try}$
    **end if**
  **end for**
**end while**
**return** $\beta$

The Bell number $B_n$ counts possible non-empty partitions of the set of $n$ numbers and has been linked to many areas in combinatorics [122]. Due to [123] the Bell number has upper bound:

$$B_n < (\frac{0.792n}{\ln n + 1})^n, \tag{2.22}$$

which, while smaller than $n!$, grows extremely fast.

The length of covering anonymous walk $\beta$ is at least $m$ and therefore a cardinality of a set $S = \bigcup \beta$ of length $l = \mathcal{O}(m)$ for a single vertex in a graph with $m$ edges is $\mathcal{O}(B_{m-1})$.

Instead of learning the whole set $S$ for the graph across all vertices, one may compute a certain "predetermined" $\beta \in S$, which would be also enough to use as a canonical labeling due to Theorem 2.9. For example, one can compute the lexicographically maximal cover $\beta \in S$ without requiring access to the full set $S$. Algorithm 2 searches the lexicographically maximal cover $\beta$ in a graph $G$ that traverses each edge exactly once in each direction.

In a nutshell, the Algorithm 2 builds a cover $\beta$ by iteratively exploring the neighborhood of each node, adding a new edge in a depth-first search manner. If a new unvisited edge exists, the algorithm adds a corresponding edge (in both direction), until no unvisited edges exist.

Prior to the explanation of the algorithm, let $replace(\beta, v, w)$ be a function that replaces last occurrence of $v$ in $\beta$ with $(v, w, v)$ triplet. So for example, $replace((1, 2, 3, 2, 1), 2, 4) = (1, 2, 3, 2, 4, 2, 1)$. Given these definitions we present an algorithm that returns a canonical labeling for a given graph $G$ and source vertex $v$.

The Algorithm 2 takes a graph $G$ and a source vertex $v$ as input and outputs a covering anonymous walk $\beta$ with $2m + 1$ elements such that each edge in $G$ is traversed once in each direction. The algorithm first initializes a stack $S$ of nodes to check for connectivity and current cover $\beta$. In lines 7-14, the algorithm adds a triplet between last element in the stack and a new yet-unvisited element $x$, which is higher by one for the current maximum of $\beta$. In lines 16-22, the algorithm attempts to connect already added elements in $\beta$ with each other. The algorithm terminates when there is no element in the stack $S$, which attained when all possible edges have been verified.

As the algorithm checks connectivity between each pair of vertices exactly once, in total there are $\mathcal{O}(n^2)$ requests to verify if $\beta$ exists in $G$ (lines 9 and 19). One way to compute a request $\beta \in G$ is to add the next node in a random walk while in parallel verifying that it does adhere to the next node in $\beta$. Each edge in the original graph $G$ appears two times (one for each direction) in the final support $\beta$ and therefore final $\beta$ has $l = 2m + 1$ elements. The following proposition establishes the connection between the original graph $G$ and the supporting graph $G_\alpha$ and therefore sufficient to test graph isomorphism by replacing computation of the whole set $S$ with the maximal $\beta$ from the Algorithm 2.

**Proposition 2.11.** *Algorithm 2 returns lexicographically maximal covering anonymous walk starting from vertex $v$ in graph $G$ such that each edge in $G$ is traversed once in each direction and it makes $\mathcal{O}(n^2)$ requests to graph $G$.*

*Proof.* We split our proof into two parts. In the first part, we prove that the final $\beta$ traverses each edge and in the second part we prove that its maximal.

Suppose that there exists an edge that is missing from the $\beta$, it means that both of the vertices are missing from the corresponding walk $w$. Let $(u_1, u_2) \in E$ be an edge that is missing from $\beta$. Let's focus on a vertex $u_1$. Since graph $G$ is undirected connected it means there exists a path from $u_1$ to some vertex $v$ in a walk $w$ that has a corresponding element $j$ in $\beta$. If $(u_1, v) \in E$, then in the Algorithm 2 there will be a step when $v$ will

check for connectivity with $u_1$ (either in line 9 or line 19), therefore $u_1$ will be added. In case if $(u_1, v) \notin E$, the argument can be repeated for the vertex $v$ and due to connectivity of all vertices in graph $G$, all nodes in $G$ will be added to the final cover $\beta$. A similar argument can be used to show that all edges also belong to $\beta$. Therefore, $\beta$ is a covering anonymous walk.

To prove that $\beta$ is maximal lexicographically among all covers that traverse each edge once in each direction, note that the Algorithm first attempts to connect a current node with a new maximum element in $\beta$ (line 6). If such an element exists then it adds this node and continues from it, until no new node can be added to the last element in $\beta$, in which case it backtracks to the preceding element. For the preceding element it first attempts to connect a node that is not present yet in $\beta$ and after this stage it connects itself with all elements that are already present in the reversed order (line 16), which guarantees that the new edges are added from the highest elements to the lowest elements in $\beta$. Since $\beta$ is built deterministically (each request in lines 9 and 19 happens among all possible random walks), the final $\beta$ will be maximal among all covers that traverse each edge at least once.

□

The running time of the Algorithm 2 is $\mathcal{O}(n^2 t)$, where $\mathcal{O}(t)$ is the time to answer request if anonymous walk $\beta_{try}$ exists in $G$, which depends on the number of anonymous walks in a graph. Note that the final cover $\beta$ is lexicographically maximal cover and therefore its first elements subsume the sequence that corresponds to the longest path in the graph $G$. Indeed, if $(1, 2, \ldots, k, k+1)$ are the first $k+1$ elements in $\beta$, then the longest path that exists in the graph $G$ has $k$ nodes; otherwise, the Algorithm 2 would attempt to add a new element in the sequence. Since the $k+1$ element in $\beta$ is smaller than $k$, then there is no path of length $k+1$ in graph $G$. On the other hand, there is a path of length $k$ in graph $G$ that corresponds to the first $k$ elements in $\beta$. Therefore, on the $k+1$ request of the Algorithm 2 one may answer a decision problem of the longest path problem, which is known to be $\mathbb{NP}$-complete. Hence, computationally the Algorithm 2 is at least as hard as solving the longest path problem. Hence the following theorem holds:

**Theorem 2.12.** *Finding maximal cover $\beta$ is $\mathbb{NP}$-hard.*

By Theorem 2.12 there is no known polynomial-time algorithm to find exactly the maximal anonymous cover in graph $G$. Instead one may hope to approximate the solution by having a policy of selecting the next nodes in a random walk. The general procedure of building an arbitrary cover of length $2m+1$ is akin to the Algorithm 2, with a

**Algorithm 3** Building Covering Anonymous Walk

**Input:** graph $G$, vertex $v$, policy $\pi$
**Output:** cover $\beta$
Initialize stack $S = [1]$, $\beta = [1]$, $m = \#$ edges
Initialize set $Q = (v)$ of visited nodes
Initialize map $M(1) = v$ between elements in $\beta$ and nodes
**while** $S \neq \emptyset$ **do**
    last $= S[-1]$
    $x = \max(\beta) + 1$
    # phase 1: explore not visited nodes
    $node = M[last]$
    $\eta = \pi(G, node, Q)$
    **if** $\eta \neq Null$ **then**
        $\beta = replace(\beta, last, x)$
        $S.append(x)$
        $M[x] = \eta$
        add $\eta$ to $Q$
    **else**
        $S.pop()$
    **end if**
    # phase 2: interconnect existing elements in $\beta$
    $U = [x - 1, x - 2, \ldots, last + 1]$
    **for all** $u \in U$ **do**
        $t_1 = M[u]$, $t_2 = M[last]$
        **if** $(t_1, t_2)$ edge exists in $G$ **then**
            $\beta = replace(\beta, last, u)$
        **end if**
    **end for**
**end while**
**return** $\beta$

change of building explicit random walk instead of making requests if $\beta$ exists in graph $G$. Specifically, the Algorithm 3 has two phases.

In phase 1, the algorithm selects a neighbor *node* according to the input policy for the current node. The policy $\pi$ must return some neighbor $\eta$ in graph $G$ that has not been added to the visited set $Q$ yet; or otherwise, return $Null$. If such neighbor exists then it replaces a corresponding last element in $\beta$ with a triplet $(last, x, last)$, where $x$ is a new element in $\beta$, and adds $x$ to the stack $S$ and the map $M$ and mark a corresponding node $\eta$ as visited. If all of the neighbors have been explored in $\beta$, then we delete the last element in the stack $S$.

In phase 2, the algorithm selects a set of elements $U$, which indicates all possible elements to which $last$ element in $\beta$ can be connected. It contains elements $[x-1, x-2, \ldots, last+1]$, all of which are already present in $\beta$, and therefore by checking if the corresponding nodes have an edge in the graph $G$, the algorithm interconnects existing elements in $\beta$.

**Algorithm 4** Random policy $\pi_{rand}$

1: **Input:** graph $G$, vertex $v$, visited set $Q$
2: **Output:** neighbor $u$ or $Null$
3: Initialize set of neighbors $N_v$
4: **if** $N_v \setminus Q = \emptyset$ **then**
5: **return** $Null$
6: **else**
7: **return** random node from $N_v \setminus Q$
8: **end if**

Given a policy $\pi$, one can generate a sample of $\tau$ covering anonymous walks $\beta$, $S = \bigcup_{1...\tau} \beta$ and then use it along with the Algorithm 1 to test for isomorphism of two graphs. Note that if $S(G_1)$ and $S(G_2)$ are two generated samples for graphs $G_1$ and $G_2$ and $S(G_1) \bigcap S(G_2) \neq \emptyset$, then there exists at least one $\beta$ such that $G_1 \cong G_\beta \cong G_2$ by Theorem 2.9. Hence, a set $S$ is a *complementary* graph invariant. Whether, a set $S$ is a complete graph invariant depends on the selected policy $\pi$ and the choice of the policy is critical for the efficiency of the graph isomorphism test.

For example, a random policy $\pi_{rand}$ in the Algorithm 4 selects a random not-visited neighbor or returns $Null$. As the returned neighbors appear in random order with the policy $\pi_{rand}$, the complementary graph invariant $S = \bigcup_{1...\tau} \beta$ can be different for two isomorphic graphs if the number of samples $\tau$ is much smaller than the total number of possible covers in these graphs. As the space of all possible covers in a graph is extremely large, the random policy is not efficient for graphs with many vertices.

There are open questions that have not been yet answered and that we believe may significantly improve the presented algorithm in their ability on finding a solution to the longest path and graph isomorphism problems.

1. In Theorem 2.10 we showed that an upper bound for the number of all anonymous walks with $n$ nodes in a complete graph $K_n$ equals to $B_{n-1}$. The set of all covering anonymous walks is contained in this set and represents the search space for the agent's policy, therefore finding stricter bounds on the set of all covers will shed the light on the efficiency of the Algorithm 2 and the complexity of the problem of finding the maximal cover in a graph.

2. One way to reduce the cardinality of the search space for the agent is to reduce the length of the anonymous walk. In this work the length of the cover is $l = 2m + 1$ as the Algorithm 3 guarantees to find a cover of length $l$. The walk of a minimal length that traverses each edge at least once (i.e. a cover) is a variant of a Chinese Postman Problem [124], where starting and end points are not necessarily the same. Finding an efficient algorithm (e.g. linear time in the number of edges) that

returns a cover of a length less than $2m+1$ has significant practical improvements due to the vast reduction of the search space.

## 2.5 Algorithms

### 2.5.1 Combinatorial model

By definition, a weighted directed graph is a tuple $G = (V, E, \Omega)$, where $V = \{v_1, v_2, \ldots, v_n\}$ is a set of $n$ vertices, $E \subseteq V \times V$ is a set of edges, and $\Omega \subset \mathbb{R}$ is a set of edge weights. Given graph $G$ we construct a *random walk graph* $R = (V, E, P)$ such that every edge $e = (u, v)$ has a weight $p_e$ equals to $\omega_e / \sum_{v \in N_{out}(u)} \omega_{(u,v)}$, where $N_{out}(u)$ is the set of out-neighbors of $u$ and $\omega_e \in \Omega$. A random walk $w$ with length $l$ on graph $R$ is a sequence of nodes $u_1, u_2, \ldots, u_{l+1}$, where $u_i \in V$, such that a pair $(u_i, u_{i+1})$ is selected with a probability $p_{(u_i,u_{i+1})}$ in a random walk graph $R$. A probability $p(w)$ of having a random walk $w$ is the total probability of choosing the edges in a random walk, i.e. $p(w) = \prod_{e \in w} p_e$.

According to the Definition 2.8, anonymous walk is a random walk, where each state is recorded by its first occurrence index in the random walk. The number of all possible anonymous walks of length $l$ in an arbitrary graph grows exponentially with $l$ (Figure 2.7). Consider an initial node $u$ and a set of all different random walks $W_l^u$ that start from $u$ and have length $l$. These random walks correspond to a set of $\eta$ different anonymous walks $\mathcal{A}_l^u = (a_1^u, a_2^u, \ldots, a_\eta^u)$. A probability of seeing anonymous walk $a_i^u$ of length $l$ for a node $u$ is $p(a_i^u) = \sum_{\substack{w \in W_l^u \\ w \mapsto a_i}} p(w)$. Aggregating probabilities across all vertices in a graph and normalizing them by the total number of nodes $N$, we get the probability of choosing anonymous walk $a_i$ in graph $G$:

$$p(a_i) = \frac{1}{N} \sum_{u \in G} p(a_i^u) = \frac{1}{N} \sum_{u \in G} \sum_{\substack{w \in W_l^u \\ w \mapsto a_i}} p(w).$$

We are now ready to define network embeddings that we name combinatorial anonymous walk embeddings (AWE).

**Definition 2.13 (Combinatorial AWE).** Let $\mathcal{A}_l = (a_1, a_2, \ldots, a_\eta)$ be the set of all possible anonymous walks of length $l$. *Anonymous walk embedding* of a graph $G$ is the vector $f_G$ of size $\eta$, whose $i$-th component corresponds to a probability $p(a_i)$, of having anonymous walk $a_i$ in a graph $G$:

$$f_G = (p(a_1), p(a_2), \ldots, p(a_\eta)). \tag{2.23}$$

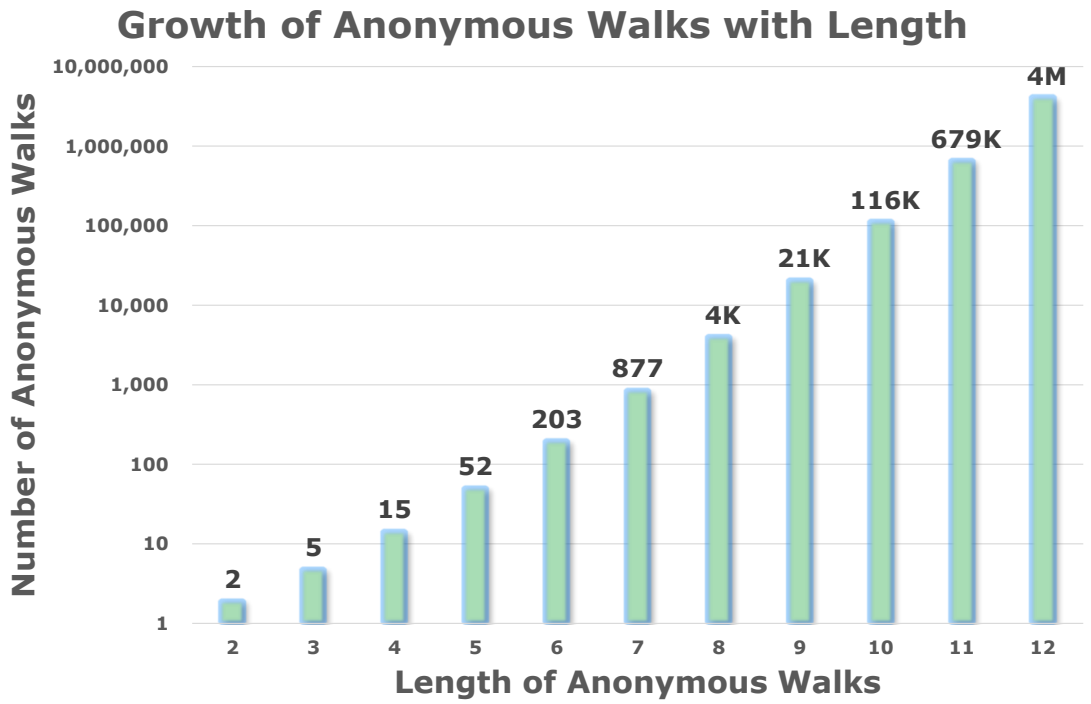



FIGURE 2.7: The number of different anonymous walks increases exponentially with length of walks $l$. $Y$-axis is in log scale.

Direct computation of AWE relies on the enumeration of all different random walks in graph $G$, which is shown below to grow exponentially with the number of steps $l$.

**Theorem 2.14.** *The running time of Anonymous Walk Embeddings (eq. 2.23) is* $\mathcal{O}(nl(d_{in}^{max}(v) \cdot d_{out}^{max}(v))^{l/2})$, *where* $d_{in/out}^{max}$ *is the maximum in/out degree in graph* $G$ *with* $n$ *vertices.*

*Proof.* Let $k_l$ be the number of random walks of length $l$ in a directed graph. According to [125] $k_l$ can be bounded by the powers of in- and out-degrees of nodes in $G$:

$$k_l^2 \leq (\sum_{v \in G} d_{in}^l(v))(\sum_{v \in G} d_{out}^l(v)).$$

Hence, the number of random walks in a graph is at most $n(d_{in}^{max}(v) \cdot d_{out}^{max}(v))^{l/2}$, where $d_{in/out}^{max}$ is the maximum in/out degree. As it requires $\mathcal{O}(l)$ operations to map one random walk of length $l$ to anonymous walk, the theorem follows. □

**Sampling.** As complete counting of all anonymous walks in a large graph may be infeasible, we describe a sampling approach to approximate the true distribution. In this fashion, we draw independently a set of $m$ random walks and calculate its corresponding empirical distribution of anonymous walks. To guarantee that empirical and actual

distributions are close with a given confidence, we set the number $m$ of random walks sufficiently large.

More formally, let $\mathcal{A}_l = (a_1, a_2, \ldots, a_\eta)$ be the set of all possible anonymous walks of length $l$. For two discrete probability distributions $P$ and $Q$ on set $\mathcal{A}_l$, define $L_1$ distance as:

$$\|P - Q\|_1 = \sum_{a_i \in \mathcal{A}} |P(a_i) - Q(a_i)|$$

For a graph $G$ let $\mathfrak{D}_l$ be the actual distribution of anonymous walks $\mathcal{A}_l$ of length $l$ and let $X^m = (X_1, X_2, \ldots, X_m)$ be i.i.d. random variables drawn from $\mathfrak{D}_l$. The empirical distribution $\mathfrak{D}^m$ of the original distribution $\mathfrak{D}_l$ is defined as:

$$\mathfrak{D}^m(i) = \frac{1}{m} \sum_{X_j \in X^m} [\![X_j = a_i]\!],$$

where $[\![x]\!] = 1$ if $x$ is true and 0 otherwise.

Then, for all $\varepsilon > 0$ and $\delta \in [0, 1]$ the number of samples $m$ to satisfy $P\{\|\mathfrak{D}^m - \mathfrak{D}\|_1 \geq \varepsilon\} \leq \delta$ equals to (from [6]):

$$m = \left\lceil \frac{2}{\varepsilon^2}(\log(2^\eta - 2) - \log(\delta)) \right\rceil. \tag{2.24}$$

For example, there are $\eta = 877$ possible anonymous walks with length $l = 7$ (Figure 2.7). If we set $\varepsilon = 0.5$ and $\delta = 0.05$, then $m = 4888$. If we decrease $\varepsilon = 0.1$ and $\delta = 0.01$, then the number of samples will increase to 122500.

As transition probabilities for random walks can be preprocessed, sampling of a node in a random walk of length $l$ can be done in $\mathcal{O}(1)$ via alias method. Hence, the overall running time of the sampling approach to compute combinatorial anonymous walk embeddings is $\mathcal{O}(ml)$.

Our experimental study shows state-of-the-art classification accuracy of combinatorial AWE on real datasets. We continue to design a neural approach that eliminates the sparsity of combinatorial embeddings.

### 2.5.2 Neural model

Our approach for learning network embeddings is analogous to methods for learning paragraph vectors in a text corpus [60]. In our case, an anonymous walk is a word,

a randomly sampled set of anonymous walks starting from the same node is a set of co-occurring words, and a graph is a document.

**Neighborhoods of anonymous walks.** To leverage the analogy from NLP, we first need to generate a corpus of co-occurring anonymous walks in a graph $G$. We define a neighborhood between two anonymous walks of length $l$ if they share the same source node. We note that unlike NLP, such neighborhood does not define any order, but it still gives the context of which anonymous walks are present in a graph, and hence will try to maximize the similarity between graph embedding and embeddings of occuring anonymous walks. This is similar to other methods such as shortest-paths co-occurrence in DGK [56] and rooted subgraphs neighborhood in graph2vec [95], which proved to be successful in empirical studies. Therefore, we iterate over each vertex $u$ in a graph $G$, sampling $T$ random walks $(w_1^u, w_2^u, \ldots, w_T^u)$ that start at node $u$ and map to a sequence of co-occurred anonymous walks $s^u = (a_1^u, a_2^u, \ldots, a_T^u)$, i.e. $w_i^u \mapsto a_i^u$. A collection of all $s^u$ for all vertices $u \in G$ is a corpus of co-occurred anonymous walks in a graph and is analogous to a collection of sentences in a document.

**Training.** In this framework, we learn representation vector d of a graph and anonymous walks matrix W (see Figure 2.8). Vector d has $1 \times d_g$ size, where $d_g$ is embedding size of a graph. The matrix W has $\eta \times d_a$ size, where $\eta$ is the number of all possible anonymous walks of length $l$ and $d_a$ is embedding size of anonymous walk. For convenience, we call d as a document vector and W as a word matrix. Each graph corresponds to its vector d and an anonymous walk corresponds to a row in a matrix W. The model tries to predict a target anonymous walk given co-occurring context anonymous walks and a graph.

Formally, a sequence of co-occurred anonymous walks $s = (a_1, a_2, \ldots, a_T)$ corresponds to vectors $\mathrm{w}_1, \mathrm{w}_2, \ldots, \mathrm{w}_T$ of matrix W, and a graph $G$ corresponds to vector d. We aim to maximize the average log probability:

$$\frac{1}{T} \sum_{t=\Delta}^{T-\Delta} \log p(\mathrm{w}_t | \mathrm{w}_{t-\Delta}, \ldots, \mathrm{w}_{t+\Delta}, \mathrm{d}), \tag{2.25}$$

where $\Delta$ is a window size, i.e. number of context words for each target word. Probability in objective (2.25) is defined via softmax function:

$$p(\mathrm{w}_t | \mathrm{w}_{t-\Delta}, \ldots, \mathrm{w}_{t+\Delta}, \mathrm{d}) = \frac{e^{y(\mathrm{w}_t)}}{\sum_{i=1}^{\eta} e^{y(\mathrm{w}_i)}} \tag{2.26}$$

Each $y(\mathrm{w}_t)$ is unnormalized log probability for output word $i$:

$$y(\mathrm{w}_t) = b + Uh(\mathrm{w}_{t-\Delta}, \ldots, \mathrm{w}_{t+\Delta}, \mathrm{d})$$

where $b \in \mathbb{R}$ and $U \in \mathbb{R}^{d_a+d_g}$ are softmax parameters. Vector $h$ is constructed by first averaging walk vectors $\mathrm{w}_{t-\Delta}, \ldots, \mathrm{w}_{t+\Delta}$ and then concatenating with a graph vector d. The reason is that since anonymous walks are randomly sampled, we average vectors $\mathrm{w}_{t-\Delta}, \ldots, \mathrm{w}_{t+\Delta}$ to compensate for the lack of knowledge on the order of walks; and at the same time, the graph vector $d$ is shared among multiple (context, target) pairs.

To avoid computation of the sum in softmax equation (2.26), which becomes impractical for large sets of anonymous walks, one can use Hierarchical softmax [126] or NCE loss functions [127] to speed up training. In our work, we use sampled softmax [128] that for each training example picks only a fraction of vocabulary according to a chosen sampling function. One can measure the distribution of anonymous walks in a graph via means of definition 2.23 and decide on a corresponding sampling function.

At every step of the model, we sample context and target anonymous walks from a graph and compute the gradient error from a prediction of target walk and update vectors of context walks and a graph via gradient backpropagation. When given several networks to embed, one can reuse word matrix W across graphs, thereby sharing previously learned embeddings of walks.

Summarizing, after initialization of matrix W for all anonymous walks of length $l$ and a graph vector d, the model repeats the following two steps for all nodes in a graph: 1) for sampled co-occurred anonymous walks the model calculates a loss (Eq. 2.25) of predicting a target walk (one of the sampled anonymous walks) by considering all context walks and a graph; 2) the model updates the vectors of context walks in matrix W and graph vector d via gradient backpropagation. One step of the model is depicted in Figure 2.8. After using up all sampled corpus, a learned graph vector d is called *anonymous walk embedding*.

**Definition 2.15** (**Neural AWE**). *Anonymous walk embedding* of a graph $G$ is a vector representation d learned on a corpus of sampled anonymous walks from a graph $G$.

So even though graph and walk vectors are initialized randomly, as an indirect result of predicting a walk in the context of other walks and a graph the model also learns feature representations of networks. Intuitively, a graph vector can be thought of as a word with a special meaning: it serves as an overall summary for all anonymous walks in the graph.

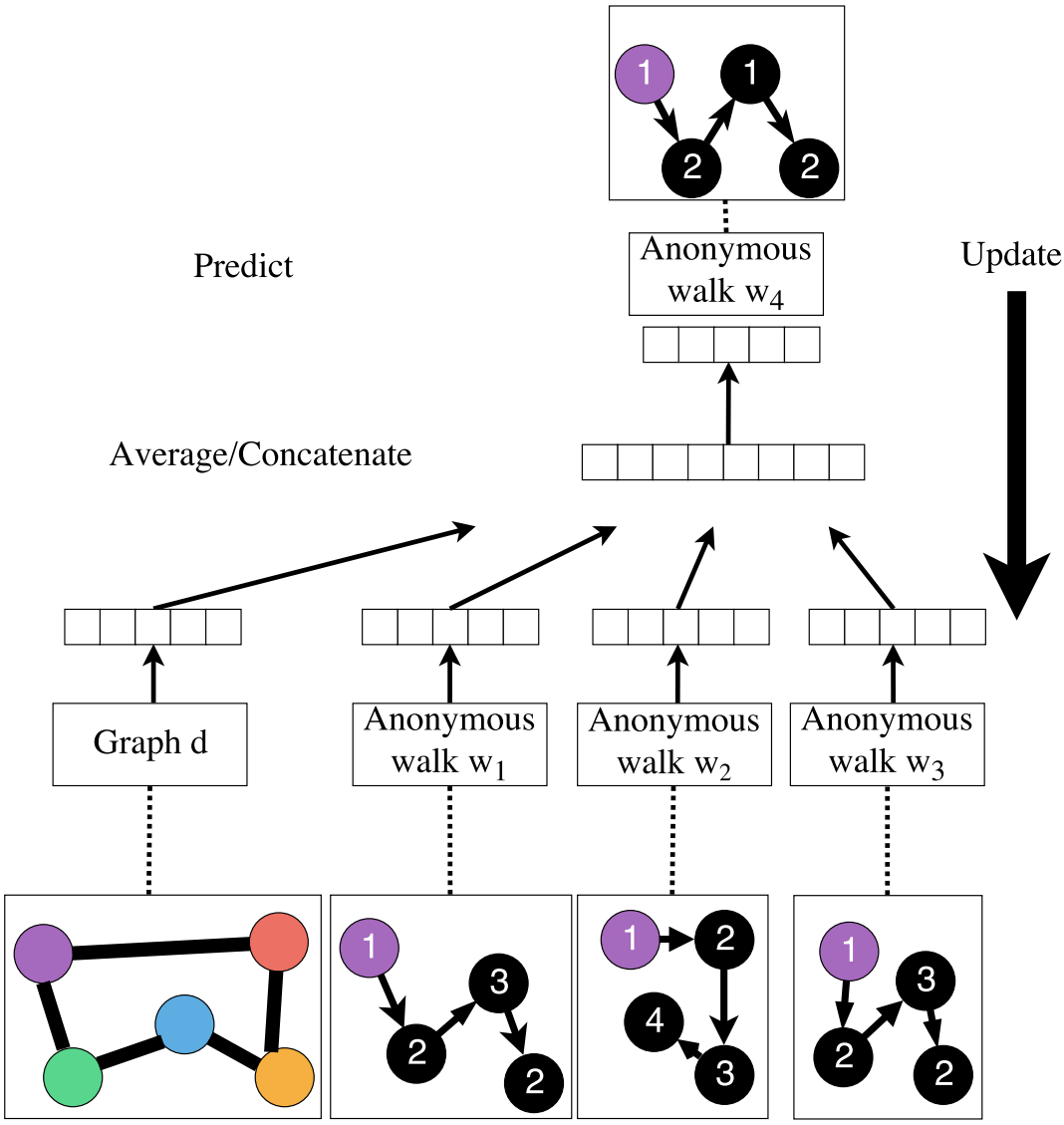



FIGURE 2.8: A framework for learning neural anonymous walk embeddings. The graph is represented by a vector $d$ and anonymous walks are represented by rows of matrix W. All co-occurring anonymous walks start from the same node in a graph. The goal is to predict a target walk $w_4$ by its surrounding context walks $(w_1, w_2, w_3)$ and a graph vector d. We average embeddings of context walks and then concatenate with a graph vector to predict a target vector. Vectors are updated using stochastic gradient descent on a corpus of sampled anonymous walks.

In our experiments, we show how anonymous walk network embeddings can be used in the graph classification problem, demonstrating state-of-the-art performance in classification accuracy.

## 2.6 Application to graph classification

Graph classification is a task to predict a class label of a whole graph and it has found applications in bioinformatics [129] and malware detection [95]. In this task, given a

series of $N$ graphs $\{G_i\}_{i=1}^N$ and their corresponding labels $\{L_i\}_{i=1}^N$, we are asked to train a model m: $G \mapsto L$ that would efficiently classify new graphs. The formal formulation of the problem can be found in 2.3.

Two typical approaches to graph classification problem are (1) supervised learning classification algorithms such as PSCN algorithm [57] and (2) graph kernel methods such as WL kernel [8]. As we are interested in designing task-agnostic network embeddings that do not require labeled data during training, we show how to use anonymous walk embeddings in conjunction with kernel methods to perform classification of new graphs. For this, we define a kernel function on two graphs.

When $X \subseteq \mathbb{R}^n$, several popular choices of kernel exist [51]:

- **Inner product** $k(x, y) = \langle x, y \rangle$, $\forall x, y \in \mathbb{R}^n$,
- **Polynomial** $k(x, y) = (\langle x, y \rangle + c)^d$, $\forall x, y \in \mathbb{R}^n$,
- **RBF** $k(x, y) = \exp(-\frac{\|x - y\|_2^2}{2\sigma^2})$, $\forall x, y \in \mathbb{R}^n$.

With network embeddings, it is then easy to define a kernel function on two graphs:

$$K(G_1, G_2) = k(f(G_1), f(G_2)), \tag{2.27}$$

where $f(G_i)$ is an embedding of a graph $G_i$ and $k$: $(x, y) \mapsto \mathbb{R}^n$ is a kernel function.

To train a graph classifier m one can then construct a square kernel matrix $\mathcal{K}$ for training data $G_1, G_2, \ldots, G_N$ and feed this matrix to a kernelized algorithm such as SVM. Every element of kernel matrix equals to: $\mathcal{K}_{ij} = K(G_i, G_j)$. For classifying new test instance $G_\tau$, one would first compute graph kernels with training instances $(K(G_1, G_\tau), K(G_2, G_\tau), \ldots, K(G_N, G_\tau))$ and provide it to a trained classifier m.

In our experiments, we use anonymous walk embeddings to compute kernel matrices and show that kernelized SVM classifier achieves top performance comparing to more complex state-of-the-art models.

### 2.6.1 Experimental evaluation

We evaluate our embeddings on the task of graph classification for a variety of widely-used datasets.

**Datasets.** We evaluate performance on two sets of graphs. One set contains *unlabeled* graph data and is related to social networks [56]. Another set contains graphs with

labels on nodes and/or edges and originates from bioinformatics [8]. Statistics of these ten graph datasets presented in Table 2.1.

**Evaluation.** We train a multiclass SVM classifier with a one-vs-one scheme. We perform a 10-fold cross-validation and for each fold we estimate SVM parameter $C$ from the range [0.001, 0.01, 0.1, 1, 10] using validation set. This process is repeated 10 times and average accuracy is reported, i.e. the average number of correctly classified test graphs.

TABLE 2.1: Graph datasets used in classification experiments. The columns are: Name of dataset, Number of graphs, Number of classes (maximum number of graphs in a class), Average number of nodes/edges.

| Dataset | Source | Graphs | Classes (Max) | Nodes Avg. | Edges Avg. |
|---|---|---|---|---|---|
| COLLAB | Social | 5000 | 3 (2600) | 74.49 | 4914.99 |
| IMDB-B | Social | 1000 | 2 (500) | 19.77 | 193.06 |
| IMDB-M | Social | 1500 | 3 (500) | 13 | 131.87 |
| RE-B | Social | 2000 | 2 (1000) | 429.61 | 995.50 |
| RE-M5K | Social | 4999 | 5 (1000) | 508.5 | 1189.74 |
| RE-M12K | Social | 12000 | 11 (2592) | 391.4 | 913.78 |
| Enzymes | Bio | 600 | 6 (100) | 32.6 | 124.3 |
| DD | Bio | 1178 | 2 (691) | 284.31 | 715.65 |
| Mutag | Bio | 188 | 2 (125) | 17.93 | 19.79 |

**Competitors.** GK is a graphlet kernel [6] and DGK is a deep graphlet kernel [56] with graphlet size equals to 7. We note that DGK is a neural network model that learns a matrix of similarity between substructures, similarly to our NN approach. WL is Weisfeiler-Lehman graph kernel algorithm [8] with a height of subtree pattern equals to 7. WL proved consistently strong results comparing to other graph kernels and supervised algorithms, presenting state-of-the-art model for graph classification problem. ER is exponential random walk kernel [52] with exponent equals to 0.5 and kR is $k$-step random walk kernel with $k = 3$ [130].

**Setup.** For combinatorial anonymous walk embeddings (Def. 2.23), we choose length $l$ of walks from the range $[2, 3, \ldots, 10]$ and approximate actual distribution of anonymous walks using sampling equation (2.24) with $\varepsilon = 0.1$ and $\delta = 0.05$.

For neural anonymous walk embeddings (Def. 2.15), we set length of walks $l = 10$ to generate a corpus of co-occurred anonymous walks. We run gradient descent with 100 iterations for 100 epochs with batch size that we vary from the range [100, 500, 1000, 5000, 10000]. Context walks are drawn from a window, which size varies in the range [2,

4, 8, 16]. The embedding size of walks and graphs $d_a$ and $d_g$ equals to 128. Finally, the candidate sampling function for softmax equation (2.26) chooses uniform or loguniform distribution of sampled classes.

To perform classification, we compute a kernel matrix, where Inner product, Polynomial, and RBF kernels are tested. For RBF kernel function we choose parameter $\sigma$ from the range $[10^{-5}, 10^{-4}, \ldots, 1, 10]$; for Polynomial function we set $c = 0$ and $d = 2$. We run the experiments on a machine with Intel(R) Xeon(R) CPU E5-2680 v4 @ 2.40GHz and 32GB RAM[3]. We refer to our algorithms as AWE (NN) and AWE (GK) for neural and combinatorial approaches correspondingly.

**Classification results.** Table 2.3 presents results on classification accuracy for Social unlabeled datasets. AWE approaches are consistently at the top, sharing top-2 results for all six social datasets, despite being an unsupervised approach, unlike PSCN. At the same time, Table 2.5 shows accuracy results for labeled bio datasets. Note that AWE are learned using only the topology of the network and not node/edge labels. In this setting, embeddings obtained by AWE (GK) approach achieve competitive performance for the labeled datasets.

TABLE 2.2: Comparison of classification accuracy (mean ± std., %) in Social datasets. Top-2 results are in **bold**. OOM is out-of-memory.

| | Algorithm | IMDB-M | IMDB-B | COLLAB |
|---|---|---|---|---|
| Neural | DGK | 44.55 ± 0.52 | 66.96 ± 0.56 | 73.09 ± 0.25 |
| Kernel | WL | 49.33 ± 4.75 | **73.4 ± 4.63** | **79.02 ± 1.77** |
| | GK | 43.89 ± 0.38 | 65.87 ± 0.98 | 72.84 ± 0.28 |
| | ER | OOM | 64.00 ± 4.93 | OOM |
| | kR | 34.47 ± 2.42 | 45.8 ± 3.45 | OOM |
| Ours | AWE (NN) | **51.54 ± 3.61** | **74.45 ± 5.83** | **73.93 ± 1.94** |
| | AWE (GK) | **51.58 ± 4.66** | 73.13 ± 3.28 | 70.99 ± 1.49 |

TABLE 2.3: Comparison of classification accuracy (mean ± std., %) in Social datasets (continued). Top-2 results are in **bold**. OOM is out-of-memory.

| | Algorithm | RE-B | RE-M5K | RE-M12K |
|---|---|---|---|---|
| Neural | DGK | 78.04 ± 0.39 | 41.27 ± 0.18 | 32.22 ± 0.10 |
| Kernel | WL | 81.1 ± 1.9 | 49.44 ± 2.36 | 38.18 ± 1.3 |
| | GK | 65.87 ± 0.98 | 41.01 ± 0.17 | 31.82 ± 0.08 |
| | ER | OOM | OOM | OOM |
| | kR | OOM | OOM | OOM |
| Ours | AWE (NN) | **87.89 ± 2.53** | **50.46 ± 1.91** | **39.20 ± 2.09** |
| | AWE (GK) | **82.97 ± 2.86** | **54.74 ± 2.93** | **41.51 ± 1.98** |

[3]Code can be found at https://github.com/nd7141/AWE

**Overall observations.**

- Tables 2.3 and 2.5 demonstrate that AWE is competitive to supervised state-of-the-art solutions in graph classification task. Importantly, even with simple classifiers such as SVM, AWE increases classification accuracy compared to other more complex neural network models. Likewise, just comparing graph kernels, we can see that anonymous walks are at the top with traditional graph objects such as graphlets (GK kernel) or subtree patterns (WL kernel).

- While combinatorial and neural approaches are different in nature, the resulted classification accuracy is close across many datasets. As such, only on RE-B dataset neural approach has more than 5% increase in the accuracy. In practice, we found that using a combinatorial approach for small length $l$ (e.g. $\leq 10$) produces competitive results, while neural approach works best for a large number of iterations and length $l$.

- Polynomial and RBF kernel functions bring non-linearity to the classification algorithm and are able to learn more complex classification boundaries. Table 2.4 shows that RBF and Polynomial kernels are well suited for combinatorial and neural models respectively.

TABLE 2.4: Kernel function comparison in classification task (%).

| Algorithm | IMDB-M | COLLAB | RE-B |
|---|---|---|---|
| AWE (NN) *RBF* | 50.73 | **73.93** | **87.89** |
| AWE (NN) *Inner* | **51.54** | 73.77 | 84.82 |
| AWE (NN) *Poly* | 45.32 | 70.45 | 79.35 |
| AWE (GK) *RBF* | **51.58** | **70.99** | **82.97** |
| AWE (GK) *Inner* | 46.45 | 69.60 | 76.83 |
| AWE (GK) *Poly* | 46.57 | 64.3 | 67.22 |

TABLE 2.5: Classification accuracy (%) in labeled Bio datasets. Top-2 results are in **bold**. OOM is out-of-memory.

| | Algorithm | Enzymes | DD | Mutag |
|---|---|---|---|---|
| Neural | DGK | 27.08 ± 0.79 | – | 82.66 ± 1.45 |
| Kernel | WL | **53.15 ± 1.14** | **77.95 ± 0.70** | 80.72 ± 3.00 |
| | GK | 32.70 ± 1.20 | **78.45 ± 0.26** | 81.58 ± 2.11 |
| | ER | 14.97 ± 0.28 | OOM | 71.89 ± 0.66 |
| | kR | 30.01 ± 1.01 | OOM | 80.05 ± 1.64 |
| Ours | AWE (GK) | **35.77 ± 5.93** | 71.51 ± 4.02 | **87.87 ± 9.76** |

**Scalability.** To test for scalability, we learn network representations using AWE (NN) algorithm for Erdos-Renyi graphs with increasing sizes from [10, $10^1$, $10^2$, $10^3$, $10^4$, $3 \cdot 10^4$]. For each size, we construct 10 Erdos-Renyi graphs with $\mu = np \in [2, 3, 4, 5]$, where $n$ is the number of nodes and $p$ is the probability of having an edge between two arbitrary nodes. In that case, a graph has $m \propto \mu n$ edges. We average time to train AWE (NN) embeddings across 10 graphs for every $n$ and $\mu$. Our setup: the size of embeddings equals to 128, batch size equals to 100, window size equals to 100. We run AWE (NN) model for 100 iterations in one epoch. In Figure 2.9, we empirically observe that the model to learn AWE (NN) network representations scales to networks with tens of thousands of nodes and edges and requires no more than a few seconds to map a graph to a vector. We note that this experiments measures pure efficiency without consideration of classification accuracy.

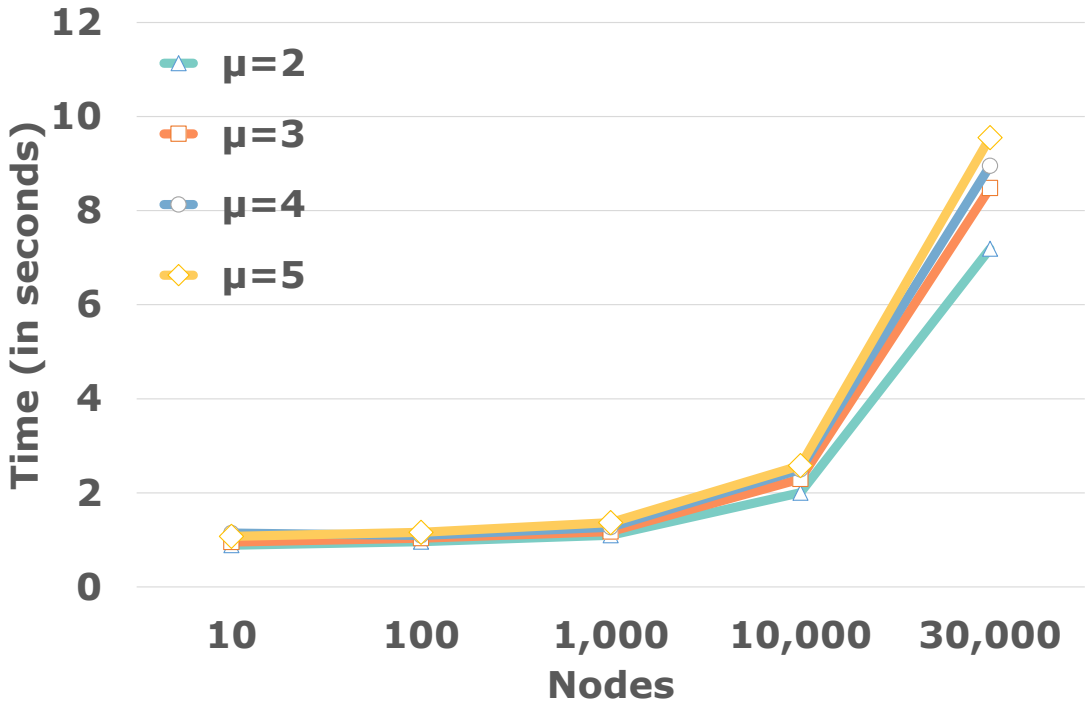



FIGURE 2.9: Average running time to generate anonymous walk embedding for Erdos-Renyi graphs, with $\mu = np \in [2, 3, 4, 5]$ where $n$ is the number of nodes and $p$ is probability parameter of Erdos-Renyi model. $X$-axis is in log scale.

**Intuition behind the performance.** There is a couple of factors that leads anonymous walk embeddings to state-of-the-art performance in the graph classification task. First, the use of anonymous walks is backed up by a recent discovery that, under certain conditions, the distribution of anonymous walks of a single node is sufficient to reconstruct a topology of the ball around a node. Hence, at least on a level of a single node, the distribution of anonymous walk serves as a unique representation of subgraphs in a network. Second, the neural approach reuses hitherto learned embeddings matrix W in previous iterations for learning embeddings of new graph instances. Therefore

one can think of anonymous walks as words that have semantic meaning unified across all graphs. While learning graph embeddings, we simultaneously learn the meaning of different anonymous walks, which provides extra information for our model.

## 2.7 Application to medical diagnostics

In the above, we have seen that AWE embeddings can work well with standard graph datasets in classification tasks. In what follows we describe the framework to analyze real data associated with medical diagnostics of the patients. To do so, we first obtain the magnetic resonance imaging data (MRI), process it to derive the graph structures, and then apply graph embeddings as a solution to the graph classification problem 2.3.

Our motivation comes from the fact that identifying correct diagnostics is a huge problem in medical science, which costs billions of dollars[4] and more human importantly lives. Graph embeddings are a relatively new approach to this problem but as we show can bring incremental values to already known solutions. In particular, among the known psychiatric disorders, depression is believed to be among the most prevalent psychiatric illnesses, with an estimated 300 million people affected in all age groups. Depression studies have identified that in 75% [131] of cases patients experience more than one depression act, which has a profound negative impact on the well-being and professional life [132, 133]. Moreover, some of the forms for depression such as major depressive disorder or bipolar depression amount to a large portion of disabilities around the world. Hence, a better understanding of the treatment methods and diagnostics of depression has far-reaching consequences.

It has been found [134–136] that epilepsy and depression have a strong association, with the one often causing another. Epileptic patients often have an additional risk of being affected by depression and moreover makes treatment more difficult: there is less effect of medication on the patients, more challenging prescription of the correct treatment due to the presence of two conditions, which leads to the development of by-products of the antiepileptic medication.

There are many factors that affect the development of depression and many studies have confirmed that corruptions in the topological and functional properties of the neural brain networks are the underlying majority of mental and social symptoms of mood diseases [137]. Analysis of MRI scans of the brain identified a hopeful set of features [138, 139] that have a strong correlation with diagnosing depression. As such imagery

[4] http://www.who.int/news-room/fact-sheets/detail/depression

may be useful to provide additional insights on pathophysiological instruments of mood diseases [140, 141].

In the following, we propose to compose the graphs based on dynamic MRI images and then apply graph embeddings to facilitate the diagnostics task. We experiment with two sets of embeddings based on anonymous walks by neural model AWE and based on subtree patterns of WL graph kernel. Designing graphs from the MRI images is known to be difficult due to the noise coming from human scanning, hence we extensively describe our preprocessing, cleaning, and graph construction pipeline. We formulate the problem in terms of classification of the diagnosis of the patients and use graph embeddings along with other features to assess the discriminative power of all features. The patient data contain four different classes, each associated with the diagnosis of the patients: (1) patients with epilepsy, (2) patients with depression, (3) patients with epilepsy and depression, and (4) healthy volunteers. All classes are non-overlapping, so for example class (1) and (3) contain different patients.

1. Depression and Health patients (DvsH): we design graph-based embeddings to classify depression from healthy patients and evaluate several baselines available in the literature [142].

2. Epilepsy and Depression-Epilepsy patients (EvsDE): we propose several markers that discriminate depression versus patients with both conditions, epilepsy, and depression.

### 2.7.1 fMRI data and pipeline

Magnetic resonance imaging (MRI) is a popular scan of the human body, when strong magnetic fields created within a body part of the interest, produce images of the human tissues. Function MRI (fMRI) is a dynamical variant of MRI, where the physiological changes of the tissues are measured. The quality of the images depends on several factors such as the noise of the environment and the resolution of the camera. The latter presents a big computational challenge as it significantly increases the amount of data a practitioner needs to operate on. For example, a standard voxel size of one cubic millimeter in the case of fMRI imaging may result in $10^5$ for a typical brain volume, which also depends on the scanning parameters, duration of the scanning session and scanners detection coil. Having hundreds of sequential scans, one fMRI procedure may take at least $10^7$ dimensions. Hence dimensionality reduction techniques may be necessary to proceed with ML models.

One way to cope with such high dimensions of the data is to represent functional images as the correlation images, i.e. a matrix of correlation values where pairwise similarities between the regions of the brain are measured. After, a graph representation of that matrix can be composed by picking the right threshold or maximizing the likelihood between the predicted values and observed data. As such patterns of anomalous functional behavior can be used as an indicator of dysfunction. Previous methods of functional connectivity include probabilistic graphical models [143] and matrix decomposition of functional brain networks (FBN) [144]. We construct a graph that represents interconnectivity between known brain atlas of different regions [144] and extract graph based-features to provide as an input to the ML models. Graph-based features can include combinatorial statistics over graphs such as size and order of graphs, degree sequence, betweenness and centrality measures, or alternatively it can be embeddings that capture global and local properties of a graph. After a set of features is constructed, a classification model such as SVM or neural network can be trained to predict the disorder label of the subject. This approach circumvents extreme amounts of data that traditional approaches use and it leads to high accuracy compared to several baselines.

A recent approach of fMRI scans has demonstrated the effectiveness of using ML models with connectivity patterns to classify patients into four groups [139]. In particular, the authors established four different subtypes of connectivity among patients with depression. For that, they derived 33,154 connectivity features from 258 functional network nodes that include all brain regions via estimating correlation among all pairs of the regions. After they identified two types of features that correlate with different symptoms of the patients. SVM classifier was used to predict the diagnosis of different subtypes purely based on these sets of features, leading to the accuracy of 89.2%. In cross-validation, a control group was accurately predicted the diagnosis with a sensitivity of 84.1% to 90.9% and specificity of 84.1% to 92.5%. This demonstrates that ML models based on fMRI images can be an efficient tool in medical diagnostics. We take this approach to analyze if recent advancements of graph representation can be integrated into this pipeline and bring additional value.

We experiments with two types of features: simple graph features and graph embeddings. We compute common graph metrics that we describe as a baseline for the problem as well as test WL graph kernel and combinatorial AWE embeddings. We describe these features in more detail next.

Given the functional connectivity matrices or graphs for patients and healthy controls, the diagnostic task could be formulated as follows. In $k$ indirect connectivity graphs $(G_1, G_2, ..., G_k)$ with labels $l_1, l_2, ..., l_k$, $l_i \in \{1, 0\}$, indicating diagnosis, we need to predict labels for new graphs – to diagnose new patients $(G_{k+1}, G_{k+2}, ..., G_{k+n})$.

For this purpose, graph features describing each node – brain region separately and the whole graph – brain connectome could be constructed. Entire brain connectivity shows how efficiently distinct brain regions exchange the information and is described by average efficiency, global efficiency and local efficiency.

Average efficiency of a connectivity graph G is defined as

$$E(G) = \frac{1}{n(n-1)} \sum_{i<j \in G} \frac{1}{d(i,j)},$$

where $n$ is number of nodes – regions in a graph, and $d(i,j)$ is the length of the shortest path between nodes $i$ and $j$.

Global efficiency of a network is

$$E_{glob}(G) = \frac{E(G)}{E(G^{ideal})},$$

where $G^{ideal}$ is the fully connected graph.

Local efficiency of a node $i$ shows how well its neighbors communicate when node $i$ is formally removed, i.e.

$$E_{loc}(G) = \frac{1}{n} \sum_{i \in G} E(G_i),$$

where $G_i$ is the subgraph containing just neighbors of node $i$, without the node $i$ itself.

To describe each brain region its degree, betweenness centrality, average neighbor degree, clustering coefficient, degree centrality, closeness centrality, path length, eigenvector centrality could be calculated.

Degree is the number of edges incident to the vertex. It shows the importance of a brain region. Average neighbor degree is the average degree of the neighborhood of each node. Betweenness centrality makes it possible to rank vertices in terms of their importance depending on the number of shortest paths passing through them. Betweenness centrality of a node $v$ is the sum of the fraction of all-pairs shortest paths that pass through $v$:

$$c_B(v) = \sum_{s,t \in V} \frac{\sigma(s,t|v)}{\sigma(s,t)},$$

where $V$ is the set of nodes, $\sigma(s,t)$ is the number of shortest paths between $s$ and $t$, and $\sigma(s,t|v)$ is the number of those paths passing through some node $v$.

Clustering coefficient is defined as

$$c_u = \frac{2T(u)}{deg(u)(deg(u)-1)},$$

where $T(u)$ is the number of triangles through node $u$ and $deg(u)$ is the degree of $u$. Degree centrality of a node $v$ is the fraction of nodes it is connected to. Closeness centrality of a node $u$ is defined as

$$C(u) = \frac{n-1}{\sum_{v=1}^{n-1} d(v,u)},$$

where $d(v,u)$ is the shortest-path distance between $v$ and $u$, and $n$ is the number of nodes in the graph.

Gains in performance for graph classification tasks when using AWE-based kernels stem from the fact that the distribution of subtree patterns in the case of WL kernel or of anonymous walks in the case of AW kernel uniquely determines the topology of a graph and therefore is suitable for graph isomorphism problem. While WL test of isomorphism fails in some unique cases [145], it works efficiently in polynomial time. Contrary, it has been shown [146, 147] that for random walks long enough, the distribution of anonymous walks uniquely determines a graph $G$; however, the computational resources grow exponentially with the length of a walk and the size of a graph. In addition, the performance of the algorithms will also depend on the dataset and the diversity of computed statistics. For example, if the variance of the distribution of anonymous walks is quite high, the classification accuracy of the anonymous walk kernel will be high as well. Finally, in practice, it may be important to check the individual contribution of each node to the overall classification score (for instance, the areas in the brain that impacts the most the disease). As anonymous walk distribution can be computed for each vertex, it can express the importance of each node on the overall classification.

Moreover, simple graph features listed above have one strict assumption, which could limit their ability to detect distributed disruption patterns specific to psychiatric diseases in different subjects. The assumption is that similar functional disruptions take place within the same nodes — anatomical regions: local established features are calculated for each particular node as well as the concept of random walks used in WL kernels implies that node labels are known. On the other hand, depression has a complex structure of disruptions in brain function [137] which may affect different areas and brain subnetworks in different subjects. Hence, when comparing different correlation matrices one is not sure that problematic regions are the same across subjects – the problem could be in similar connectivity disruption pattern and not in a particular brain area. The concept of anonymous walks could be useful here because it is constructed in an assumption that nodes' labels are not known [146].

In what follows we describe our highly-automated fMRI processing pipeline. We assume that the input to our pipeline is the raw MRI images and the output is our predictions

that vary upon the task. For example, when we classify depression against health group, the output of the pipeline is the set of probabilities for each patient in the control group. Our pipeline is fully modular, allowing one to implement their own modules of preprocessing and analysis as long as it respects the input and output for each module. One of the crucial factors in the design of the modules is its effectiveness as depending on the processing procedures the runtime can vary from minutes to several hours. In what follows we describe our pipeline.

Our pipeline contains several modules and takes as input a series of MRI scans. In the first step, we apply a standard preprocessing also called low-level MRI handling stage. This involves time correction of each sequence, the correctness of motion, different filtering procedures and anatomical alignment for each patient [148]. The next two modules perform a correlation matrix construction from which a graph for each patient is constructed and representation of the graphs as numerical embedding. At the final module, we train a ML classifier that can be from the linear approaches such as SVM [149, 150], non-linear neural networks [149, 150], or gradient boosted trees that process particularly well heterogeneous data [151]. We note that for our pipeline we selected each step such that the most informative features of the models can be highlighted and to be able to pinpoint classification outliers and evaluate precision and recall metrics, often used in the medical applications.

After the first module of standard preprocessing, the volume of a brain is separated into 117 anatomic regions according to the AAL atlas. In the next module, we assign a correlation between time-series scans of one region with another region, i.e. the similarity between corresponding dynamics of each region is measured. These correlation coefficients comprise $117 \times 117$ binary adjacency matrix, where one is obtained if a corresponding correlation value is greater than a predetermined threshold $h$ and otherwise it is zero. We used`python` library `networkX` to perform basic graph operations. In the evaluation, we tried several values of the threshold $h$ and select the one that has substantial correlation values (p=0.05, FDR-corrected [152]).

We selected 11 graph-based features from the constructed correlation graph that were appended to the final embedding of a graph. In particular, for each patient data, we have 116*8 measures and additional three global features, which amounts to 931-dimensional feature vector.

For the extraction of the WL kernels, we used Graph Kernels[5], Eigen[6], and igraph[7] libraries. The input for the Weisfeiler-Lehman subtree kernel algorithm is a list of

[5] https://github.com/BorgwardtLab/graph-kernels
[6] http://eigen.tuxfamily.org/index.php?title=Main_Page
[7] http://igraph.org/c/

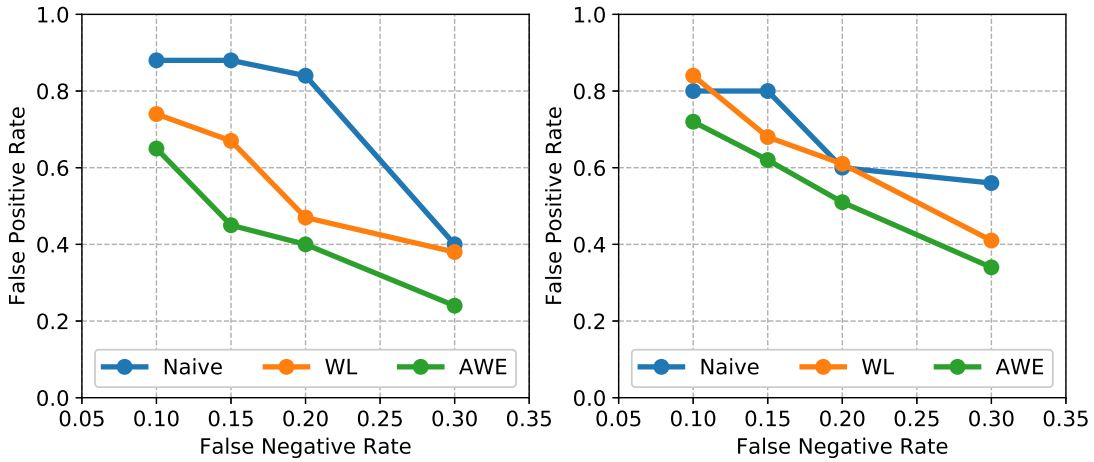



FIGURE 2.10: Classification results on *DvsH* and *DvsED* tasks in terms of False Negative Rate vs. False Positive Rate performance curves. Left image: *DvsH* task. Right image: *DvsED* task. Note how the gap between AWE and naïve graph-based features is larger for the relatively simpler *DvsH* task, indicating a clear advantage of learned features.

graphs of length 50 and parameter $\rho$ (subtrees height) and the output is $50 \times 50$ kernel matrix.

To compute AWE features, we used AWE[8] library [146]. In our experiments, we set the length of the random walk equal to 5. The output corresponds to the sample histogram of random walks.

After feature extraction, the following part of the pipeline was organized as follows:

- We tried two geometrical dimensionality reduction techniques. (1) Local Linear Embedding (LLE) and (2) Principal Component Analysis (PCA) with the obtained correlation weights [153].
- We tried two feature selection algorithms. (1) SelectKBest function is a selection technique based on Pearsons chi-squared test and ANOVA test. (2) Select top-$k$ features based on importance weights from the machine learning model such as Logistic Regression, $k$-Nearest Neighbor or a Random Forest Classifier.
- In cross-validation we selected the best parameters for a number of selected parameters among 10, 20, 50, 100 and for a number of components in the dimensionality reduction techniques 5, 10, 15, 20.

We separated training and testing datasets before feature selection and dimensionality reduction [154]. Grid search was performed using $k$-fold cross-validation with $k = 10$ folds and stratification. The best model was verified using leave-one-out cross-validation,

[8] https://github.com/nd7141/AWE

hence the results are comparable to those found in the literature. Also, the decision rule is selected with the grid search to prevent overfitting on the small data [155].

### 2.7.2 Experimental evaluation

In the experiments, our goal is to find discriminative features of the patients cohort with depression and also in combination with epilepsy. We collected a set of fMRI data at 1.5T EXCEL ART VantageAtlas-X Toshiba scanner at Z.P. Solovyev Research and Clinical Center for Neuropsychiatry. As was described before, the data were split into four nonoverlapping groups, for those with depression, with epilepsy, healthy, and both epilepsy and depression condition, such that each group is composed of 25 members. We consider two classification problems, one for distinguishing depression from the healthy control group and one from distinguishing depression from patients with double conditions of depression and epilepsy.

Functional MRI data were preprocessed according to the proposed pipeline. Features were extracted from the connectivity graph (correlation matrix) in three different ways: 1) simple graph features (denoted Naïve approach in Table 2.6) 2) WL kernel-based features (WL) 3) Anonymous Walks Embedding based features (AWE). Extracted features were passed to four classifiers as described in the pipeline. Classification results are in Table 2.6.

As all data preprocessing and classification steps were the same, so the difference in classification performance could be only due to the different quality of extracted features.

First of all, it can be seen that in EDvsE classification task all feature extraction methods demonstrate worse performance than in DvsH task. This result is obvious because, as discussed earlier, epilepsy could affect the same structures (and their function) as depression thus making it harder to distinguish between these two groups than between Depression group and Healthy controls.

Next, Naïve features performance is substantially worse for both tasks in terms of Accuracy, FPR and FNR in comparison to kernel methods, especially in EDvsE task with high allowed FPR. This could be due to subtle differences in connectivity graph structures, which cannot be captured with Naïve features.

WL features performance though in most cases better than Naïve performance, is worse than AWE performance (for any fixed FPR), although the variance is quite high to have statistical significant results. The average FNR difference between WL and AWE features performance in DvsH task is 0.13 and in EDvsE task is 0.09. It is hard to judge how significant are these results due to the small sample size, but they are stable for all

TABLE 2.6: Summary of classification results in terms of prediction accuracy for the three competing graph-based features.

| Task | Naïve | WL | AWE |
|---|---|---|---|
| DvsH | $73 \pm 15\%$ | $78 \pm 15\%$ | $\mathbf{80 \pm 12}\%$ |
| EvsDE | $67 \pm 15\%$ | $75 \pm 14\%$ | $\mathbf{76 \pm 16}\%$ |

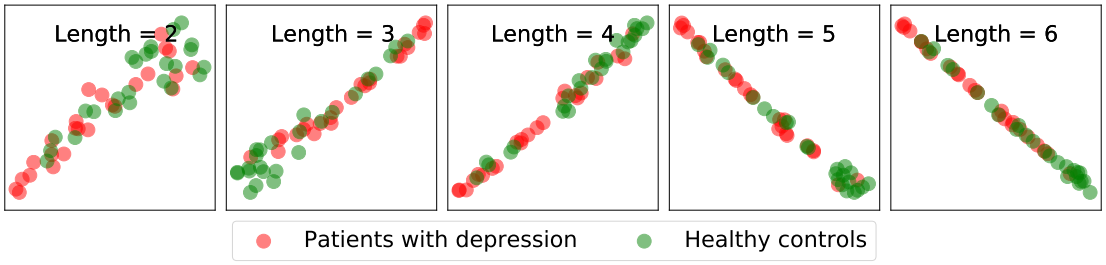



FIGURE 2.11: Visualization of two-dimensional projections of graph embeddings learned by AWE for different lengths of the random walk ($\eta = 2, \ldots, 6$). Note how embeddings for healthy controls (green circles) tend to cluster together in all cases, indicating a normal connectivity pattern. All figures were obtained with t-SNE algorithm.

selected FPR values. As previously discussed, the reason for better performance could lie in the nature of depression disease: depression has a complex structure of disruptions in brain function [137] which may affect different areas and brain subnetworks in different subjects. Whereas the disruption pattern could be similar, the damaged areas could be different, making it impossible to compare connectivity region by region or in other words to put labels on graph nodes. This finding supports the idea of the usefulness of Anonymous Walks concept and AWE framework in case of complex and subtle graph differences, which could arise in considered diagnostic tasks.

To visually inspect the reasons of the increased performance enjoyed by AWE, we have computed two-dimensional projections of our high-dimensional AWE embeddings using a popular t-SNE technique [156]. We investigate the effect of the length of anonymous random walk $\eta$ and the binarization threshold $h$ used for the correlation matrix on the learned representations. Figure 2.11 shows that longer random walks provide a larger variability of the distribution of anonymous walks, fostering the approximation of the original graph and boosting the classification accuracy. The corresponding low-dimensional embeddings tend to cluster together more tightly. On the other hand, as depicted in Figure 2.12, stronger thresholding diminishes the benefits of using AWE by generating sparser graphs that are initially easier to approximate but carry less important information. We hypothesize that AWE experiences larger performance gains for more complex networks.

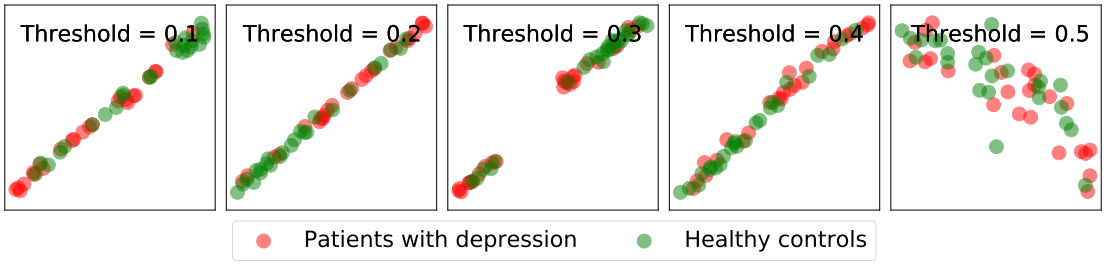



FIGURE 2.12: Visualization of the effect of threshold value used for binarization of the correlation matrix. Plotted are two-dimensional projections of graph embeddings learned by AWE for different values of the threshold ($h \in \{0.1, 0.2, 0.3, 0.4, 0.5\}$). The more sparse the original connectivity graph (larger $h$), the less do the embeddings tend to cluster, as there is less variability among the generated random walks. All figures were obtained with t-SNE algorithm.

### 2.7.3 Final remarks

In this application we analyzed how graph kernel methods are useful for fMRI data classification in neuroimaging-based psychiatric research, namely detecting depression patterns in subjects. For the sake of reproducibility we introduced a data preprocessing, cleaning, and graph extraction pipeline as well as described three approaches for graph features extraction: simple graph features, a conventional WL-based kernel, and learnable AWE features. For both Depression versus Healthy controls and Epilepsy with Depression versus Epilepsy classification tasks kernel-based methods outperformed conventional graph features; AWE approach performed slightly better than WL-based approach in terms of sensitivity and specificity. AWE visualization reveals that network embeddings derived from Healthy controls tend to group together, for a large variety of algorithm parameters, which might indicate the ability of AWE to capture subtle differences between fMRI connectivity graphs. This is the first time we evaluated WL-based features and AWE graph embeddings in an application to depression pattern classification and for neuroimaging-based diagnostics problems. Our findings could contribute to the development of new graph features as well as to its application to real-world clinical diagnostic tasks.

## 2.8 Summary

In this chapter, we described two unsupervised algorithms to compute network vector representations using anonymous walks. In the first approach, we use the distribution of anonymous walks as a network embedding. As the exact calculation of network embeddings can be expensive we demonstrate how one can sample walks in a graph to approximate actual distribution with a given confidence. Next, we show how one

can learn distributed graph representations in a data-driven manner, similar to learning paragraph vectors in NLP.

In our experiments, we show that our network embeddings even with a simple SVM classifier achieve an increase in classification accuracy comparing to state-of-the-art supervised neural network methods and graph kernels. This demonstrates that representation of your data can be more promising subject to study than the type and architecture of your predictive model. Separately we verify our models on the medical diagnostics dataset where we want to label the patients' data based on the fMRI-data. Again we see the improvements over the baselines for this setup.

# Chapter 3

# Graph embeddings for combinatorial problems

## 3.1 Motivation

Online networking offers opportunities for new types of marketing. A prime example of such a new marketing technique is viral marketing, whereby organizations run promotion campaigns through word-of-mouth effects within online social networks. The *influence maximization* (IM) problem [157], studied intensively during the last decade, aims to find well-chosen *seed nodes* from which to launch such campaigns to achieve good results.

Recent works [158, 159] have focused on the parameters that define the popularity of a post, campaign, idea, or *meme* within a network. Such works were the first to study the question of how commercial brand posts engage online social network users, drawing from the theory of Uses & Gratifications [160]; they examine post parameters such as *content type* (e.g., entertaining, informational), *media type* (e.g., vivid, interactive), *posting time* (e.g., workday, peak hours) and *valence of comments* (e.g., positive, negative). Interestingly, such studies have reached some ambivalent conclusions; for instance, [158] ascertains that entertaining content decreases the number of "likes", while [159] claims the exact opposite.

Concurrent research has studied the problem of *viral product design* [161, 162], which calls for engineering products by incorporating viral attributes to generate peer-to-peer influence that encourages adoption within a network. Aral and Walker [161] study the question of viral attribute selection under randomized trials only; Barbieri and Bonchi [162] allude to the same problem as a complement to the standard IM problem of selecting a set of seed nodes that maximizes influence, but do not investigate it as a stand-alone

problem in its own right. Conceptually, both these works pertain to attributes attached to products; they do not investigate the more general problem of choosing content, out of a set of eligible options, for any kind of *meme* spreading in a network, to make it viral.

In this section, we introduce and study the problem of selecting content that characterizes any type of *meme*, to maximize its expected spread through a network, starting from a fixed set of initial adopters. For instance, an advertisement post may feature aspects such as *topics*, *people*, *locations* and *abstract themes*. We are particularly interested in those content aspects that are associated with specific *online social network pages*; we denote such aspects as **content attributes**. Fittingly, online social network users themselves are associated with such non-personal network pages: they express their preferences for specific brands, topics of interest, public persons, hobbies, or locations by subscribing to or liking such pages. Thereby, an attribute's popularity can be gauged via its number of subscribers or page likes. For our purposes, we denote the pages that a user subscribes to or likes as **user attributes**. We contend that the more content and user attributes overlap, the more likely that user is to propagate that post. We envisage an organization that aims to achieve a high viral effect of a campaign initiated from its fixed set of subscribers. For example, assume $FlyFast$ airways wants to launch a promotion campaign in social media. $FlyFast$ already has a social network presence, and its page has a subscribers' set $S$ fixed at a given moment, while it faces constraints related to its budget and people's attention span. In their design, $FlyFast$ consultants are interested to identify a set of $k$ content attributes, out of a universe of eligible, mutually compatible options, that will maximize the expected network spread of a post starting from its subscribers' set $S$. Assume that, for $k = 4$, the optimal attribute set is *Best travel Accessories*, *Airline food guide*, *Hipster Europe*, *Backpacker tips*. Guided by this knowledge, $FlyFast$ can infuse its post with complementary content that appeals to users interested in those topics, e.g., promotions to backpackers, references to its hipster audience, and highlights on its food quality. Thereby, it can maximize its promotion's reach.

To our knowledge, we are the first to study the influence maximization problem in which the seed is *given* and post content is *sought*. Our related contributions are as follows:

**Problem Setting** We motivate the influence maximization problem in settings where the set of initial adopters is fixed or even a single point of origin, and the content of a propagated meme can be tuned. We formulate the concept of *digital influence* as a special case of social influence.

**Propagation Model** We devise a *content-aware* propagation model, whereby the probability of influence across edges depends on content. We show that with this model: (i)

the problem of choosing content attributes that maximize influence is NP-hard; (ii) the spread function is not submodular, hence no submodularity-based approximation algorithm applies; and (iii) it is NP-hard to approximate the optimal solution within a factor of $n^{1-\varepsilon}$ for $\varepsilon > 0$.

**Algorithm** We design a fast algorithm, *Explore-Update*, which achieves higher influence spread than baselines; its effectiveness is based on the iterative estimation of the marginal spread achieved by each attribute, while its efficiency is gained by limiting such computations only to nodes within a probability-based distance threshold $\theta$ and attributes potentially affecting such nodes.

**Experiments** We compare Explore-Update to two baselines and show that it always achieves better propagation results, while it is significantly faster than a nave Greedy approach; we calculate the optimal solution on a reduced dataset with a small universe of attributes, showing Explore-Update can achieve optimality; last, we demonstrate the scalability of Explore-Update on seed set size.

## 3.2 Product recommendation in graphs

From the preceding discussion, we conclude that any brand would gain by maximizing the expected effectiveness of its product promotion campaigns within an online social network. We assume that there exists a certain set of *subscribers* to the brand's social network page, and a promotion campaign aims to influence the maximum number of *non-subscribers*; as we discussed, such users, are associated with topics expressing their interests.

### 3.2.1 Problem formulation

We model an online social network as a directed graph $G = (V, E)$, where $V = \{v_1, v_2, ..., v_n\}$ is a set of nodes, each of which corresponds to an individual user, and $E \subset V \times V$ is a set of directed edges representing social relations among users. Each node $v$ has a set of associated attributes $F_v = \{f_v^1, f_v^2, \ldots\}$, from a universe $\Phi$, that define user preferences; we identify these attributes as the non-personal network pages a user expresses interest in. A meme propagated through the network is associated with a set of attributes $F = \{f_{p_1}, f_{p_2}, \ldots, f_{p_K}\} \subseteq \Phi$; these content attributes, along with the user attributes $F_v$ associated with the targeted node $v$, affect the probability of its propagation across a network edge $e_{uv}$.

Accordingly, we define the *Content-Aware Cascade* model (CAC) as a variant of the Independent Cascade model (IC), in which edge propagation probabilities depend on content and user attributes. A CAC diffusion process unfolds in steps, starting from an initial seed set of activated nodes. A node $u$ activated at time step $t$ has a single chance to activate its out-neighbors. The process is incremental, as nodes can alternate only from inactive to active states; the diffusion ends when there is no newly activated node at a given step. At any step, a newly activated node $u$ activates its out-neighbor $v$ with probability $p(u, v)$ equal to:

$$\begin{aligned} p_{uv} &= b_{uv} + q_{uv} \cdot h_{uv}(F_v, F), \quad b_{uv}, q_{uv} \in [0, 1] \\ h_{uv}(F_v, F) &= \min\left\{\tfrac{1-b_{uv}}{q_{uv}}, |F_v \cap F|\right\} \end{aligned} \tag{3.1}$$

where $b_{uv}$ is a *base probability* on an edge and $q_{uv}$ a *marginal probability* that indicates how much the probability of an edge increases for each selected attribute in $F$ matching a preference of node $v$, as indicated by the transition function $h_{uv}(F_v, F)$, with a sanity bound of $\frac{1-b_{uv}}{q_{uv}}$. We emphasize that the marginal probability $q_{uv}$ distinguishes among different user links, albeit not among different attributes for a given link; a more complex model could distinguish among different attributes, or even define a probability distribution function over the set of all attributes [163], to be learned by historical logs. We choose to relegate the problem of defining and learning such probability distribution functions to future work and now study the problem under the modeling assumption that each attribute has the same independent effect on the probability function. Nevertheless, our simplified model forms a special case of any more complex model in which each attribute would have a different effect on the probability function; i.e., in this special case, such effects are rendered equal. Therefore, our subsequent hardness and inapproximability results hold for any such a more complex model as well. Furthermore, parameters $q_{uv}$ and $b_{uv}$ can be obtained from past data, as in [163]; in our setting, we assume that such parameters have been obtained in advance.

Given a seed set $S$ of subscribers, for every set of attributes $F$, we can obtain the total number of activated nodes after running several trials of the diffusion process from $S$ [157]. The *expected* number of activated nodes for a given seed set $S$ and a selected set of attributes $F$ is called *influence spread*, denoted as $\sigma(F|S)$, or, as $S$ is fixed in our problem, just $\sigma(F)$. Thus, $\sigma(F)$ is the *expected* spread of the diffusion, which we can calculate using live-edge instances of the graph (i.e., instances of activated-only edges [157]) as:

$$\sigma(F) = \sum_{X} Prob[X] \cdot \sigma_X(F) \tag{3.2}$$

where $\sigma_X(F)$ is the influence spread in live-edge instance $X$.

We define the Content-Aware Influence Maximization (CAIM) problem as follows:

Given a directed graph $G = (V, E)$, where each node $v$ is associated with user attributes $F_v = \{f_v^1, f_v^2, \ldots\}$ from a universe of eligible attributes $\Phi$, a seed set of adopter nodes $S$, quantities $q_{uv}$, $b_{uv}$ for each edge $e_{uv} \in E$, and a transition function $h_{uv}(F_v, F) = \min\left\{\frac{1-b_{uv}}{q_{uv}}, |F_v \cap F|\right\}$ for edge probabilities, select a set of $k$ attributes $F \subset \Phi$ that maximizes the spread $\sigma(F|S)$ of a diffusion process with content attributes $F$ starting from $S$.

CAIM is a novel problem that aims to find out how one can maximize the benefits of a network promotion campaign with given points of departure. The motivation derives from the fact that, in the real world, brands want to exploit their own social network pages for marketing purposes. Instead of targeting the most influential initiators for a promotion, as in classical IM, one can judiciously invest in the creation of a post with lucrative content, under fixed initiators, guided by the content attributes provided by the CAIM solution. As promotions can be formed with a wide variety of content attributes, each possible attribute set $F$ corresponds to a different probabilistic graph, on which we can compute the influence spread of the seed set $S$; the attribute set $F$ that achieves maximum spread constitutes the CAIM solution. We emphasize that due to the drastic difference between classical IM and CAIM in the way influence spread is achieved, *the solutions to these two problems cannot be qualitatively compared against each other.*

### 3.2.2 Hardness and Inapproximability

We now show the hardness of the CAIM problem and study the properties of influence spread function $\sigma(F)$. To calculate $\sigma(F)$, we first calculate edge probabilities with respect to the selected content attributes $F$ and then estimate the expected spread on the graph starting from the given set of subscribed nodes $S$.

**Theorem 3.1.** *The CAIM problem with the Content-Aware Cascade model is NP-hard.*

*Proof.* Consider an instance of the NP-complete Set Cover problem, defined by a collection of subsets $S_1, S_2, \ldots, S_m$, a universe of elements $U = \{u_1, u_2, \ldots, u_n\}$ and an integer $k$. We are asked whether there are $k$ sets that will cover all elements in $U$. We show that Set Cover can be reduced to a *trivial* instance of CAIM as follows: We

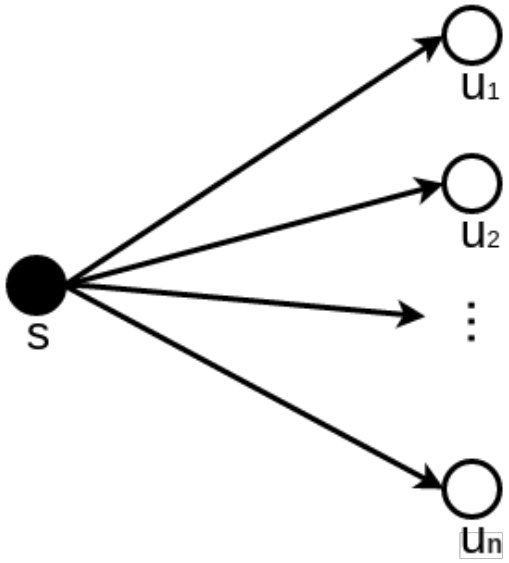



FIGURE 3.1: A graph instance demonstrating that the CAIM problem is NP-hard.

construct a bipartite graph with one activated node on the left side that connects to $n$ nodes on the right side, as shown in Figure 3.1. We map each member $u_i$ of universe $U$ to a node on the right side and add an attribute $f_j$ to set $F_{u_i}$ if $u_i$ belongs to subset $S_j$. We set $b_{uv} = 0$ and $q_{uv} = 1$ for all edges $(u, v) \in E$, i.e., a node $v$ is influenced if at least one of its user attributes is selected. In this trivialized version of CAIM, the spread can be computed deterministically; there is no need for expected spread computations. Then, an algorithm that could optimally solve this trivial instance of CAIM, among others, would decide any instance of SET COVER: if we can target all nodes in the CAIM instance using $k$ attributes, we can in effect cover all elements in $U$ using $k$ subsets in SET COVER. Otherwise, if the optimal spread in CAIM does not reach all nodes, it follows that there is no set of $k$ subsets that covers all elements in SET COVER. Thus, by reduction from SET COVER, CAIM is at least as hard as any problem in NP. □

By Theorem 3.1, there is no polynomial-time algorithm to find an optimal set of attributes $F$, unless P=NP. We now proceed to study the properties of the influence spread function $\sigma(F)$.

A function $\sigma(F)$ is *submodular* if it follows a *diminishing returns* rule: the marginal gain from adding an element to a set $F$ is at most as high as the marginal gain from adding the same element to a subset of $F$. That is, $\sigma(F_1 \cup \{f\}) - \sigma(F_1) \geq \sigma(F_2 \cup \{f\}) - \sigma(F_2)$, where $F_1 \subset F_2 \subset \Phi$, for any $f \in \Phi$.

We call a transition function $h_{uv}(F_v, F)$ *monotonic* on $F$ if, for subsets of attributes $F_1 \subset F_2 \subset \Phi$, it holds that $h_{uv}(F_v, F_1) \leq h_{uv}(F_v, F_2)$, for any node $v$. If the transition function is not monotonic, then the influence spread function is neither monotonic,

nor submodular, because selecting more attributes may reduce probabilities $p(u, v)$ and thereby reduce the total influence spread. We assume that attributes have nonnegative effects on users, rendering the transition function $h_{uv}(\cdot)$ monotonic: edge probabilities can only increase if we add attributes to $F$, i.e. $h_{uv}(F_v, F) \leq h_{uv}(F_v, F + \{f\})$ for any $f \in \Phi$ and $v \in G$; hence $\sigma(F)$ is monotonic. We now examine whether $\sigma(F)$ is also submodular. This turns out to not be the case, even for a monotonic and submodular transition function, as the following counterexample demonstrates.

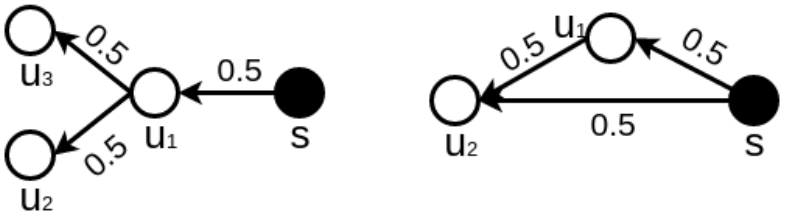



FIGURE 3.2: Increasing & decreasing marginal returns

**Example 3.1.** *Consider the graph on the left-hand side in Figure 3.2, with a universe of attributes* $\Phi = \{A, B, C\}$*, sets of preferred attributes for each node be* $F_{v_1} = \{A\}$ *and* $F_{v_2} = F_{v_3} = \{A, B, C\}$*,* $b_{uv} = 0.5$*,* $q_{uv} = \frac{1}{2|F_v|}$ *on all edges, and one active node* $s$*. Then, consider two subsets of attributes* $F_1 = \emptyset$*,* $F_2 = \{B, C\}$*, where* $F_1 \subset F_2$*, and a attribute* $f = A \in \Phi \setminus F_2$*. The achieved spreads for each attributes subset, and the respective marginal gains obtained after adding attribute* $f$ *to subsets* $F_1$ *and* $F_2$*, are calculated as follows. For subset attribute* $F_1$ *selected, we have:*

$$p_{sv_1} = \tfrac{1}{2}, \quad p_{v_1v_2} = p_{v_1v_3} = \tfrac{1}{2}$$
$$\sigma(F_1) = \tfrac{1}{2} + 2\tfrac{1}{4} = 1$$

*whereas when* $f = A$ *is added to* $F_1$*, we get:*

$$p_{sv_1} = 1, \quad p_{v_1v_2} = p_{v_1v_3} = \tfrac{2}{3}$$
$$\sigma(F_1 + \{A\}) = 1 + 2 \cdot \tfrac{2}{3} = \tfrac{7}{3}$$

*Hence* $\Delta_1 = \sigma(F_1 + \{A\}) - \sigma(F_1) = \frac{7}{3} - 1 = \frac{4}{3}$*. Similarly, for* $F_2$ *selected, we have:*

$$p_{sv_1} = \tfrac{1}{2}, \quad p_{v_1v_2} = p_{v_1v_3} = \tfrac{5}{6}$$
$$\sigma(F_2) = \tfrac{1}{2} + 2 \cdot \tfrac{5}{12} = \tfrac{4}{3}$$

*while when $f = A$ is added to $F_2$, we get:*

$$p_{sv_1} = 1, \quad p_{v_1v_2} = p_{v_1v_3} = 1$$
$$\sigma(F_2 + \{A\}) = 3$$

*Hence* $\Delta_2 = \sigma(F_2 + \{A\}) - \sigma(F_2) = 3 - \tfrac{4}{3} = \tfrac{5}{3}$.

$$\Delta_1 = \sigma(F_1 + \{A\}) - \sigma(F_1) = (\frac{1}{9} + \frac{8}{9} + \frac{12}{9}) - (\frac{1}{8} + \frac{4}{8} + \frac{3}{8}) = \frac{4}{3}$$
$$\Delta_2 = \sigma(F_2 + \{A\}) - \sigma(F_2) = 3 - (\frac{1}{72} + \frac{10}{36} + \frac{75}{72}) = \frac{5}{3}$$

*Since $\Delta_2 > \Delta_1$, the submodularity of $\sigma(F)$ does not hold.*

Given this negative result, the influence function $\sigma(F)$ might have an *increasing returns* property (supermodularity), whereby it would hold that $\sigma(F_1 \cup \{f\}) - \sigma(F_1) \leq \sigma(F_2 \cup \{f\}) - \sigma(F_2)$, for $F_1 \subset F_2 \subset \Phi$ and any attribute $f \in \Phi$. The following counterexample shows that this property does not hold either.

**Example 3.2.** *Consider the graph on the right-hand side in Figure 3.2, with a universe of attributes $\Phi = \{A, B\}$, sets of preferred attributes per node $F_{v_1} = \{A, B\}$ and $F_{v_2} = \{A\}$, $b_{uv} = 0.5$ and $q_{uv} = \frac{1}{2|F_v|}$ on all edges, and one active node $s$. Consider two subsets of attributes $F_1 = \emptyset$ and $F_2 = \{B\}$. Then, for subset attribute $F_1$ selected, we have:*

$$p_{sv_1} = \tfrac{1}{2}, \quad p_{sv_2} = p_{v_1v_2} = \tfrac{1}{2}$$
$$\sigma(F_1) = \tfrac{1}{2} + \left(1 - \left(1 - \tfrac{1}{4}\right)\tfrac{1}{2}\right) = \tfrac{9}{8}$$

*whereas when $f = A$ is added to $F_1$ we get:*

$$p_{sv_1} = \tfrac{3}{4}, \quad p_{sv_2} = p_{v_1v_2} = 1$$
$$\sigma(F_1 + \{A\}) = \tfrac{3}{4} + 1 = \tfrac{7}{4}$$

*Hence* $\Delta_1 = \sigma(F_1 + \{A\}) - \sigma(F_1) = \frac{7}{4} - \frac{9}{8} = \frac{5}{8}$. *Similarly, for* $F_2$ *selected, we have:*

$$p_{sv_1} = \tfrac{3}{4}, \quad p_{sv_2} = p_{v_1v_2} = \tfrac{1}{2}$$
$$\sigma(F_2) = \tfrac{3}{4} + \left(1 - \left(1 - \tfrac{3}{8}\right)\tfrac{1}{2}\right) = \tfrac{23}{16}$$

*while when* $f = A$ *is added to* $F_2$ *we get:*

$$p_{s,v_1} = 1, \quad p_{v_1v_2} = p_{v_1v_2} = 1$$
$$\sigma(F_2 + \{A\}) = 2$$

*Hence* $\Delta_2 = \sigma(F_2 + \{A\}) - \sigma(F_2) = 2 - \frac{23}{16} = \frac{9}{16}$.

$$\Delta_1 = \sigma(F_1 + \{A\}) - \sigma(F_1) =$$
$$(\frac{3}{16} + \frac{6}{16} + \frac{6}{16} + \frac{6}{16} + \frac{1}{16} + \frac{1}{16}) - (\frac{1}{8} + \frac{2}{8} + \frac{2}{8} + \frac{2}{8} + \frac{1}{8} + \frac{1}{8}) = \frac{5}{16}$$
$$\Delta_2 = \sigma(F_2 + \{A\}) - \sigma(F_2) = 2 - (\frac{6}{4} + \frac{1}{4}) = \frac{4}{16}$$

*Since* $\Delta_1 > \Delta_2$, *the influence function* $\sigma(F)$ *is not supermodular either.*

Eventually, we have established the following:

**Theorem 3.2.** *The spread function* $\sigma(F)$ *with a probability transition function* $h_{uv}(F_v, F) = \min\left\{\frac{1-b_{uv}}{q_{uv}}, |F_v \cap F|\right\}$ *is neither submodular nor supermodular.*

By Theorem 3.2, it follows that we cannot use a greedy algorithm with an approximation guarantee based on submodularity, as in [157]. Moreover, in the following, we show that it is NP-hard to approximate the optimal solution to CAIM.

**Theorem 3.3.** *It is NP-hard to approximate the optimal solution to the CAIM problem with the Content-Aware Cascade model within a factor $n^{1-\varepsilon}$ for any $\varepsilon > 0$.*

*Proof.* Consider an instance of the Set Cover problem, in which we need to decide whether we can cover all elements of a universe $U = \{u_1, u_2, \ldots, u_n\}$ by selecting at most $k$ subsets out of a collection of $S_1, S_2, \ldots, S_m \subset U$.

We then construct a graph $G$ for the CAIM problem with a single subscriber node $s$ and nodes $u_1, u_2, \ldots, u_n$ corresponding to elements in $U$, connected so that $u_{i-1}$ points towards $u_i$ for all $i = 2 \ldots n$, and $s$ is connected to $u_1$, and, for every subset $S_j$ an element $u_i$ belongs to, we add a attribute $f_j$ to the preferred attributes of $u_i$. Next, for some integer $c$ we add $\eta = n^c - n - 1$ more nodes $x_1, x_2, \ldots, x_\eta$ such that $u_n$ has outgoing edges to them and each $x_i$ has the same preferred attributes as $u_n$. Graph $G$, shown in Figure 3.3, has $N = n^c$ nodes. We set $b_{uv} = 0$ and $q_{uv} = 1$ for all edges, so that an edge becomes active if at least one of the attributes associated with its target node is selected. Then, if it is possible to select $k$ subsets that cover all elements of universe $U$, we can also have $N = n^c$ activated nodes. Conversely, if there is no selection of $k$ subsets that covers all $U$, then there is at least one node $u_i$ that does not get activated, precluding influence spread to nodes $x_1, x_2, \ldots, x_\eta$. We can then only target at most $n$ out of $n^c$ nodes, a fraction of $n^{1-c} = N^{\frac{1}{c}-1}$. Thus, if we had a polynomial-time algorithm that approximated the optimal solution to CAIM within a factor of $N^{1-\varepsilon}$ for any $\varepsilon > 0$, then it would suffice to set $c = \lceil \frac{1}{\varepsilon} \rceil$ and use that algorithm so as to decisively distinguish between a case that accepts a solution activating all $N$ nodes and one that does not, and thereby also decide Set Cover. Thus, by reduction from Set Cover, we have shown that it is NP-hard to approximate the optimal solution to CAIM within a reasonable factor. □

### 3.2.3 The Explore-Update Algorithm

As it is NP-hard to approximate the CAIM solution within a factor of $n^{1-\varepsilon}$ with the Content-Aware Cascade model, we proceed to design heuristic solutions therefor. We structure our exposition as follows: we first present a simple, yet time-consuming greedy heuristic; then, through a sequence of simplifying assumptions, we will generate a much more efficient algorithm called Explore-Update.

Our first proposal is a baseline greedy algorithm that selects the attribute of the highest marginal gain to add at each iteration, shown in Algorithm 5. This is an adaptation of the *Local Update* algorithm in [162] to our problem. Intuitively, it is reasonable to greedily select the locally best attribute in each iteration, especially for small values of $k$.

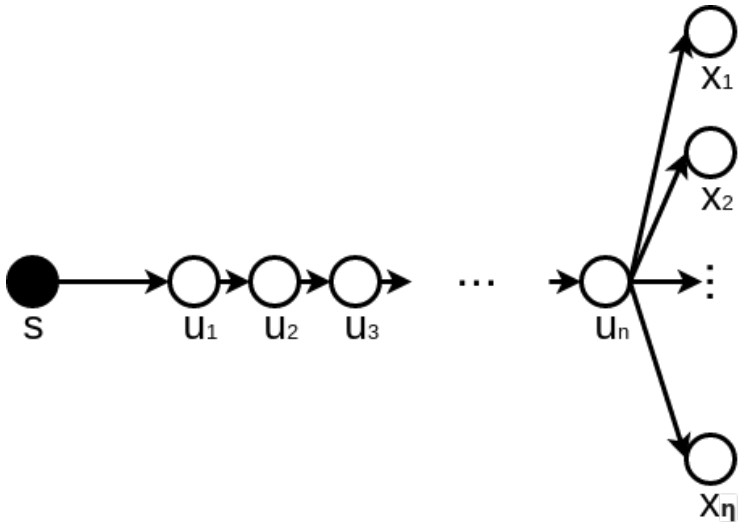



FIGURE 3.3: A graph instance demonstrating that it is NP-hard to approximate the optimal solution to the CAIM problem.

**Algorithm 5** Greedy($G$, $S$, $k$)

$F = \emptyset$
**while** $|F| < k$ **do**
  **for** every $f \in \Phi \setminus F$ **do**
    calculate $\sigma(F + \{f\})$ using Monte Carlo simulations
  **end for**
  $F = F \cup_f \{\sigma(F + \{f\})\}$
**end while**
**return** $F$

This kind of algorithm has been shown to achieve better quality than others in classical Influence Maximization [164–166].

Though simple and effective, Algorithm 5 is inefficient due to its calculation of influence spread by MC simulations. In a manner reminiscent of [164], we can improve efficiency by considering *maximum influence paths* between nodes and the seed set. We call a path $P_{max} = \langle u = u_0, u_1, u_2, \ldots, v = u_m \rangle$ between vertices $u \in S$ and $v \in G$ *maximum influence path* (MIP) if this path is the most probable among all paths between $u$ and $v$: $P_{max} =_P \prod_{i=0}^{m-1} prob(u_i, u_{i+1})$. Under the *simplifying assumption* that influence is propagated only through MIPs, we can estimate influence spread in polynomial time as follows: For a threshold $\theta$ and a node $v$, we build a tree structure called *in-arborescence* $A_{in}(v)$, which includes all MIPs of probability higher than $\theta$ from any node to $v$: $A_{in}(v) = \{\mathsf{MIP}(u, v) \mid prob(\mathsf{MIP}(u, v)) > \theta, u \in G\}$. Then, given a node $u$, the seed set $S$, and an arborescence $A_{in}(v)$, Algorithm 6 recursively estimates the probability that $u$ is activated in $A_{in}(v)$, i.e., its *activation probability* $ap(u, A_{in}(v))$.

**Algorithm 6** calculateAP($u$, $A_{in}(v)$, $S$)

**if** $u \in S$ **then**
  $ap(u, A_{in}(v)) = 1$
**else if** $u$ has no in-neighbors in $A_{in}(v)$ **then**
  $ap(u, A_{in}(v)) = 0$
**else**
  $ap(u, A_{in}(v)) = 1 - \prod_{\omega \in N_{in}(u)} (1 - ap(\omega, A_{in}(v)) prob(\omega, u))$
**end if**
**return** $ap(u, A_{in}(v))$

Based on these calculations, for all nodes $u \in G$, we can calculate the influence spread $\sigma(F)$ as follows:

$$\sigma(F) = \sum_{u \in G} ap(u, A_{in}(u)) \tag{3.3}$$

We can then employ Equation (3.3) so as to estimate influence spread in Algorithm 5, in lieu of MC simulations, deriving Algorithm 7; at each iteration, we compute the in-arborescence of node $u$ for a given threshold $\theta$ by converting each probability $p_e$ on an edge $e$ to $-\log p_e$ and employing an efficient implementation of Dijkstra's algorithm. If computing an arborescence takes time $t$, then Lines 4-6 take $nt$ and the total time is $O(k|\Phi|nt)$.

**Algorithm 7** Arb($G$, $S$, $\theta$, $k$)

$F = \emptyset$
**while** $|F| < k$ **do**
  **for** every $f \in \Phi \setminus F$ **do**
    **for** every $u \in G$ **do**
      compute $A_{in}(u)$ with threshold $\theta$
      $ap(u, A_{in}(u)) =$ calculateAP$(u, A_{in}(u), S)$
    **end for**
    calculate $\sigma(F + f) = \sum_{u \in G} ap(u, A_{in}(u))$
  **end for**
  $F = F \cup_f \{\sigma(F + \{f\})\}$
**end while**
**return** $F$

We further reduce the runtime of Algorithm 7 by eschewing redundant iterations of the loops over nodes $u$ and attributes $f$. First, we limit the calculation of in-arborescences and activation probabilities only to nodes whose in-arborescence under threshold $\theta$ reaches at least one node in $S$; only such nodes can yield non-zero estimated activation probability. To find out these nodes, we compute the out-arborescence of all nodes in $S$, $A_{out}(S)$, consisting of all MIPs of probability higher than $\theta$ from a node $v \in S$

to other nodes in $G$. Nodes in $A_{out}(S)$ yield non-zero activation probability estimates. Yet the set of paths in $A_{out}(S)$ may contain directed loops, hence we cannot apply a recursive algorithm like Algorithm 6 directly on $A_{out}(S)$; we still need to obtain the in-arborescence $A_{in}(u)$ of each $u \in A_{out}(S)$; we do so while building $A_{out}(S)$, by adding $\mathsf{MIP}(v, u)$ to $A_{in}(u)$ for each $u \in A_{out}(v)$. Algorithm 8 illustrates this Explore process.

**Algorithm 8** Explore($G$, $F$, $S$, $\theta$)

$A_{out}(S) = \emptyset$
$A_{in}(u) = \emptyset$ for every $u$ in $G$
**for** every $v \in S$ **do**
  compute $A_{out}(v)$ for given $\theta$ and $F$
  update $A_{in}(u)$ for each $u \in A_{out}(v)$
**end for**
**return** $\{A_{in}(u) \neq \emptyset |$ for $u \in G\}$

Then we can calculate influence spread $\sigma(F)$ using the union of such in-arborescences, $A_{in}$, by Algorithm 9.

**Algorithm 9** Update($A_{in}$, $S$)

1: **for** every $u \in A_{in}$ **do**
2: $ap(u) =$ CalculateAP($u$, $A_{in}(u), S$)
3: **end for**
4: $\sigma(F) = \sum_{u \in A_{in}} ap(u)$
5: **return** $\sigma(F)$

Second, we limit the calculation of marginal gain in Algorithm 7 only to those attributes that can affect the influence spread. We call an edge in $G$ *participating*, if at least one of its endpoints are in $A_{out}(S)$. Figure 3.4 presents a graph for a seed set $S$ (and selected attributes set $F$) in the green area; the yellow area includes nodes in $A_{out}(S)$; the set of participating edges $\Pi$ is shown in solid and dotted lines; dotted edges have only one endpoint in $A_{out}(S)$; non-participating edges are shown in dashed lines, in the gray area.

Non-participating edges cannot increase influence spread, regardless of whether their probability is increased; only participating edges have such potential. We limit the attributes Algorithm 7 considers based on this observation. Let $E(f)$ be the set of edges that include attribute $f$ among their preferred attributes, hence their probability is affected when adding $f$ to $F$. Then, at any iteration, if *none* of the edges in $E(f)$ is a participating edge, i.e., $E(f) \cap \Pi = \emptyset$, then attribute $f$ need not be examined as a candidate to be added to $F$; it bears no effect to influence function $\sigma(F + \{f\})$.

Putting together our enhancements to Algorithm 7, we design the polynomial-time Explore-Update algorithm (Algorithm 10). In a nutshell, at each iteration, this Explore-Update algorithm selects the hitherto unselected attribute $f$ affecting participating edges

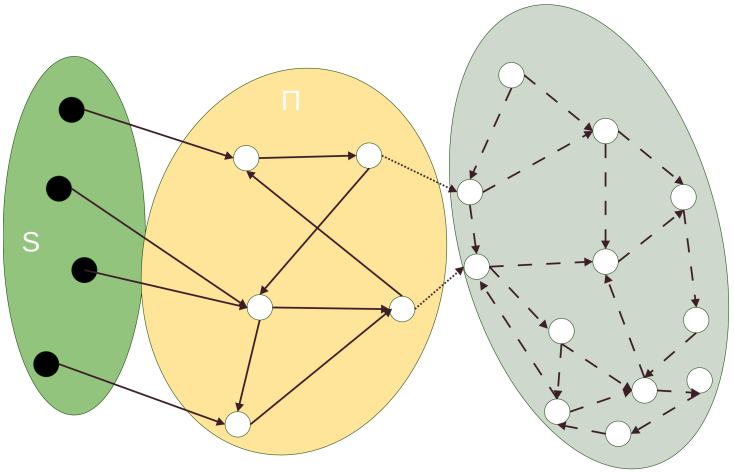


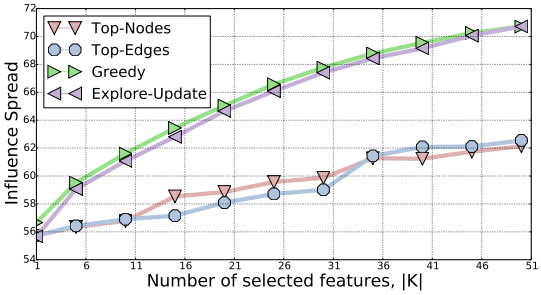

FIGURE 3.4: Participating and non-participating edges

**Algorithm 10** Explore-Update($G$, $S$, $k$, $\theta$)

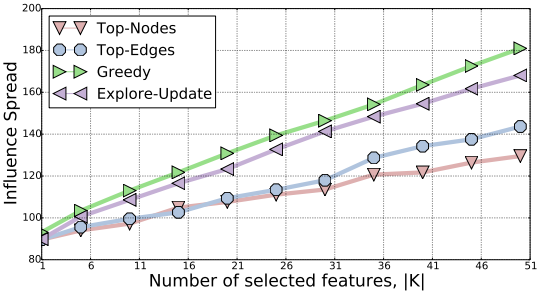
$F = \emptyset$
$A_{in}$ = Explore($G$, $F$, $S$, $\theta$)
$\Pi = \{(u, v) \in G | u \in A_{in}$ or $v \in A_{in}\}$
**while** $|F| < k$ **do**
  **for** $f \in \Phi \setminus F$ **do**
    **if** $E(f) \cap \Pi \neq \emptyset$ **then**
      $A_{in}$ = Explore($G$, $F + \{f\}$, $S$, $\theta$)
      $\Pi_f = \{(u, v) \in G | u \in A_{in}$ or $v \in A_{in}\}$
      $\sigma(F + \{f\})$ = Update($A_{in}$, $S$)
    **end if**
  **end for**
  $f_{max} =_f \{\sigma(F + \{f\})\}$
  $F = F \cup f_{max}$
  $\Pi = \Pi_{f_{max}}$
**end while**
**return** $F$

that bring about the largest increase of influence spread, using the Explore procedure for calculating in-arborescences and the Update procedure for calculating influence spread, while updating the set of participating edges $\Pi$ at each iteration and using it to determine which attributes need to be examined at the next iteration.

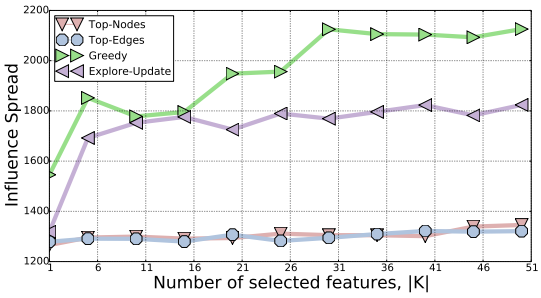
Let the time complexity to calculate an out-arborescence for node in $S$ be $t_{out\theta}$, then the Explore procedure takes $|S|t_{out\theta}$ and the Update procedure takes $O(n_{in\theta}n_{out\theta})$ time, where $n_{in\theta}$ is the number of nodes in in-arborescences, and $n_{out\theta}$ is the number of nodes in out-arborescence of $S$. Therefore, if we perform $\kappa$ calculations of $A_{in}$ per iteration, the total runtime is $O(k\kappa(|S|t_{out\theta} + n_{in\theta}n_{out\theta}))$. In effect, the Explore-Update algorithm is expected to perform well when the size of arborescences is small, and the number of



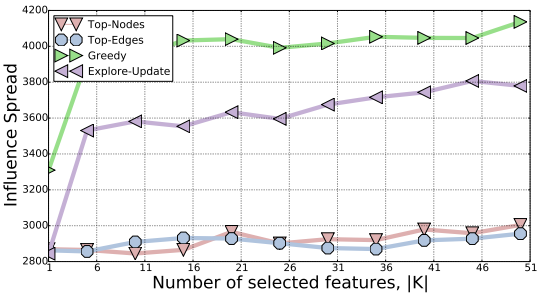

(a) Multivalency on Gnutella



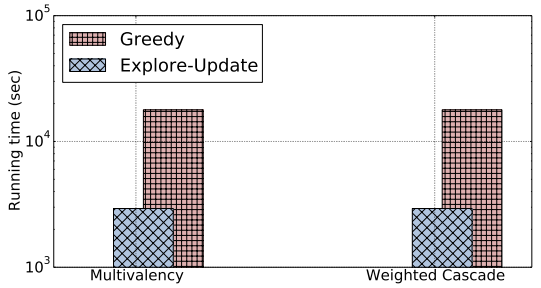

(b) Weighted Cascade on Gnutella



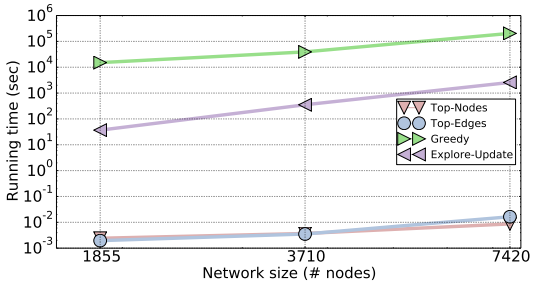

(c) Multivalency on VK




(d) Weighted Cascade on VK




Runtime on Gnutella, $k = 50$




Runtime on VK with Multivalency, $k = 20$.

FIGURE 3.5: Influence spread and runtime results.

updates $\kappa$ per iteration is smaller than $|\Phi|$. As propagation probabilities on edges are usually small in real networks, the size of arborescences is indeed expected to be small. The number of updates depends on the structure of the network. In a large-diameter network where multiple hops are required to reach most nodes from $S$ via a MIP, there is a good chance to reduce the number of computations significantly. We investigate this matter experimentally in the following.

### 3.2.4 Experimental evaluation

In this section, we present a comprehensive experimental study on the Greedy and Explore-Update algorithms we have introduced. All experiments were run on a 32GB

Intel Core i5-2450M CPU machine @ 2.50GHz, while algorithms were implemented[1] in C++.

As there is no previous work on the CAIM problem, we compare to basic baselines. Still, as we discussed, the previous work that comes closest to our problem is that by Barbieri and Bonchi [162]; yet that work solves primarily the problem of selecting a set of seed nodes, and secondarily a set of product attributes, to maximize product influence in a network. The best-performing algorithm for updating an attribute set in [162], *Local Update*, performs one addition or removal of an attribute to/from the current attribute set at each iteration; in effect, our Greedy algorithm can be considered as an adaptation of *Local Update* to our problem, where only additions of attributes are needed. Therefore, to the extent that comparison to [162] is possible, *we conduct it via the comparison to the Greedy algorithm itself.* Another method for updating an attribute set proposed in [162], *Generic Update*, is a hard-to-tune genetic algorithm, which may lead to an unpredictable number of output attributes. Besides, as the experimental study in [162] shows, *Genetic Update* offers no qualitative advantage while it is much slower than *Local Update*, which is already by far the most time-consuming algorithm in our study. Therefore, we do not consider a genetic algorithm in our experimental study.

**Diffusion models.** In the Content-Aware Cascade model, the probability on edge $(u, v)$ is a linear function of product and base probabilities $q_{uv}$ and $b_{uv}$. To assign these probabilities we use two techniques prevalent in previous work [164].

- Weighted Cascade model: probability $\frac{1}{d_v}$ is assigned to edge $(u, v)$, where $d_v$ is the in-degree of node $v$. We use this model for the sake of compatibility with previous works, even while it may fit less to our problem setting.
- Multivalency model: the probability for edge $(u, v)$ is drawn uniformly at random from a set of probabilities. We choose that set to be $[0.02, 0.04, 0.08]$.

We calculate $b_{uv}$ for every edge $(u, v)$, and set $q_{uv} = \frac{b_{uv}}{|F_v|}$.

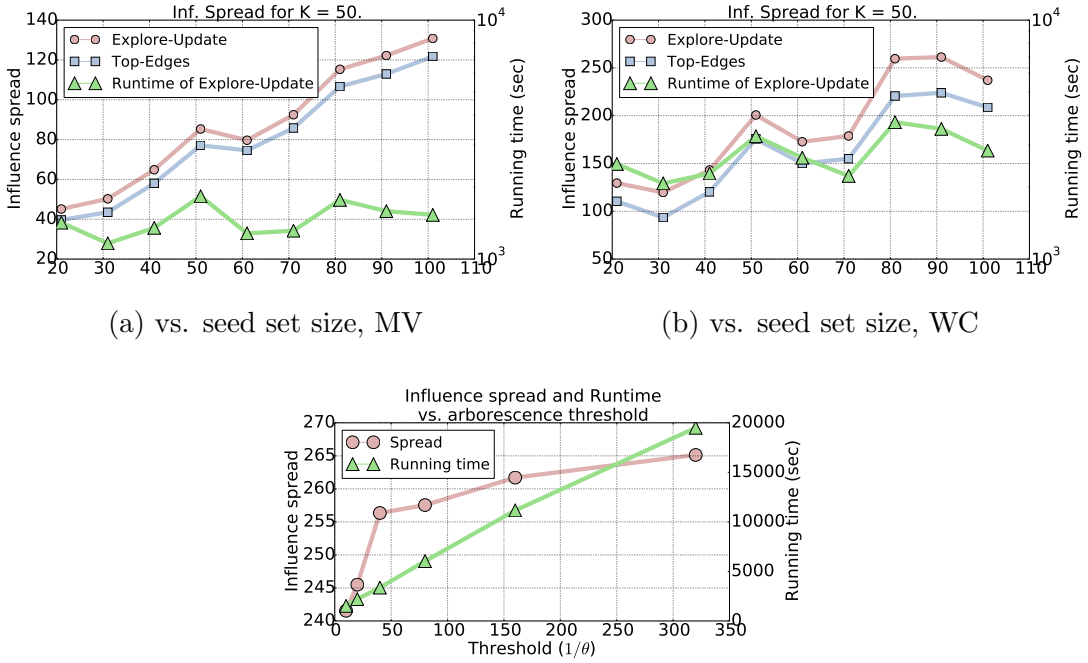
**Algorithms.** We compare the Explore-Update algorithm under the different threshold of $\theta$ values to three other algorithms:

- **Greedy** This is Algorithm 5 in this section, which is effectively an adaptation of *Local Update*, the best algorithm in [162]. A similar algorithm has been used extensively in the context of the Influence Maximization problem, and always demonstrated top performance in terms of spread while being slower than other

[1] Documentation and code is available at https://github.com/nd7141/Explore-Update

| Dataset | Gnutella | VK |
|---|---|---|
| Nodes | 10,876 | 7,420 |
| Edges | 39,994 | 57,638 |
| Average Clustering Coefficient | 0.0062 | 0.28 |
| Number of Triangles | 934 | 168,284 |
| Diameter | 9 | 16 |
| Attributes/Seed sets | 151 | 3,882 |
| Default Seed Size | 34 | 15 |

TABLE 3.1: Data characteristics




(a) vs. seed set size, MV

(b) vs. seed set size, WC

(c) vs. $\theta$

FIGURE 3.6: Influence and runtime vs. seed set size and $\theta$ on Gnutella.

heuristics [165]; it requires specifying the number of Monte-Carlo simulations to calculate influence spread, as we do in the following.

- **Top-Nodes** This algorithm measures each attribute's frequency among node preferences and selects the $k$ most frequent ones.
- **Top-Edges** This algorithm assigns to each edge $e = (u, v)$ the attribute preferences of node $v$, $F_v$, and select the $k$ most frequent attributes across all edges.
- **Brute-Force** This algorithm finds all possible sets of attributes of size $k$, computes each one's influence spread using Monte-Carlo simulations, and opts for the best. Because the solution space is exponential, we use this method on reduced datasets.

**Datasets.** We run experiments in two real-world networks. The first network is a peer-to-peer file sharing directed network Gnutella[2], where nodes represent hosts and edges represent connections between the Gnutella hosts. Our second network is extracted by crawling the social network VK[3]; nodes are users and edges are friendships among them. Statistics are presented in Table 3.1.

**Attribute assignment and seed selection.** We utilize one general and one ad-hoc method for attribute preference assignment. In Gnutella, to assign attribute preferences set $F_v$ to node $v$, we find the block partitioning that minimizes the description length of the network by stochastic block model ensemble; this technique is used to discover the block structure of empirical networks and results to block memberships for each node [167, 168]. We allow nodes to have overlapping memberships to different blocks. Each block $\beta_i$ is associated with a distinct attribute $f_i$. The attribute preference set of a node $v_j$, $F_{v_j}$ is the set of attributes of the blocks $v_j$ belongs to. The returned partitioning consists of 151 blocks; the default seed set $S$ is one of the blocks, of size 34. For VK, the data comes along with annotations of *groups* and *pages*, which allow us to derive both node attributes and seed sets. A *group* or *page* is a community of users that share content with each other and communicate about a topic of interest (e.g., football clubs or TV series). We use these group memberships to derive both node attributes and seed sets, consistently to our motivation. There are 3882 such groups; the default seed size is 15. Unless otherwise indicated, in our experiments we use the default seeds.

**Influence spread.**

Figures 3.5(a-b) present our results on competing algorithms' influence spread[4] on the Gnutella network, varying number of selected attributes $k$ from 1 to 51. We used 10000 MC simulations for Greedy, and $\theta = 1/320$ for Explore-Update. We observe that Explore-Update arrives just 1% and 5% below the performance of Greedy with the Multivalency and Weighted Cascade model, respectively. On the other hand, the Top-Edges and Top-Nodes algorithms reach only 88% and 85% of the spread of Explore-Update. Figures 3.5(d-e) present influence spread in VK network. Now Greedy used with just 500 Monte-Carlo simulations comfortably achieves 15%, 37%, and 48% higher spread than Explore-Update with $\theta = 1/40$, Top-Nodes, and Top-Edges, respectively, in MV model. The picture is similar to the WC model, where Greedy achieves spread 9%, 38%, 40% higher than Explore-Update, Top-Nodes, and Top-Edges. Overall, our results confirm that Explore-Update achieves high influence spread for networks where the local neighborhood of the seed set has structure amenable to long-distance arborescences.

[2]https://snap.stanford.edu/data/p2p-Gnutella04.html
[3]https://vk.com/
[4]We use 10000 Monte-Carlo simulations to compute the final spread of all solutions.

**Runtime.**

We now compare algorithms in terms of runtime. Figure 3.5(c) presents the results with Gnutella for $k = 50$; Explore-Update ($\theta = 1/320$) runs an order of magnitude faster than Greedy (10000 simulations); Top-Edges and Top-Nodes output a selected set in less than a second, hence we do not include them. Next, we investigate how the algorithms scale with increasing network size. We extract subnetworks of VK consisting of 1855, 3710, and 7420 nodes of the original network (i.e., 1/4, 1/2, and full network) and proportional edge density to the full network. In all cases, we compute the runtime on seed set $S$ of size 15, with the Multivalency model for $k = 20$, for Greedy (10000 simulations), Explore-Update ($\theta = 1/40$), and the Top-Edges and Top-Nodes heuristics. Figure 3.5(f) shows that runtime scales linearly in network size in all cases. Moreover, we ascertain that while Explore-Update fares no better than Greedy in terms of influence spread, it is much faster.

**Effect of Seed Size.**

We now test the performance of Explore-Update for different sizes of the seed set $S$. We select different seed sets from size 21 (minimal size for the current block partition) to 101 with step 10 on Gnutella. Figure 3.6(a-b) presents the influence spread for Explore-Update and Top-Edges for $k = 50$, as well as the runtime of Explore-Update, whereas Greedy is orders of magnitude slower for this setup, and Top-Nodes performs worse than Top-Edges. We note that Explore-Update always achieves better influence spread than Top-Edges. Interestingly, influence spread and runtime do not always grow with $|S|$. This is explicable by the fact that different seed sets induce different local structures.

**Effect of $\theta$.**

Next, we study the effect of the $\theta$ threshold, which controls the size of arborescences and thereby the influence spread achievable from a seed set $S$. Figure 3.6(c) presents the influence spread and runtime with the Gnutella network for $\theta$ in $\{\frac{1}{10}, \frac{1}{20}, \ldots, \frac{1}{320}\}$, with the WC model for $k = 50$. The runtime of Explore-Update grows linearly in the inverse threshold $\theta$, while influence spread grows logarithmically in it. A good tradeoff between quality and runtime is found at the knee point in the influence spread curve for $\theta = \frac{1}{40}$.

**Comparison to the Optimal Solution.**

By Theorem 3.3, we proved it is NP-hard to approximate the optimal solution to CAIM. Now, we compare the results of heuristics to the optimal solution obtained by brute force; we reduce the total number of attributes to 16 and use a reduced Gnutella network by selecting 2K nodes, yielding similar degree distribution properties to the original. Figure 3.7 shows the influence spread results, with the Multivalency model, for a random seed

set of size 10. Remarkably, Explore-Update finds the optimal set of attributes with varying $k$.

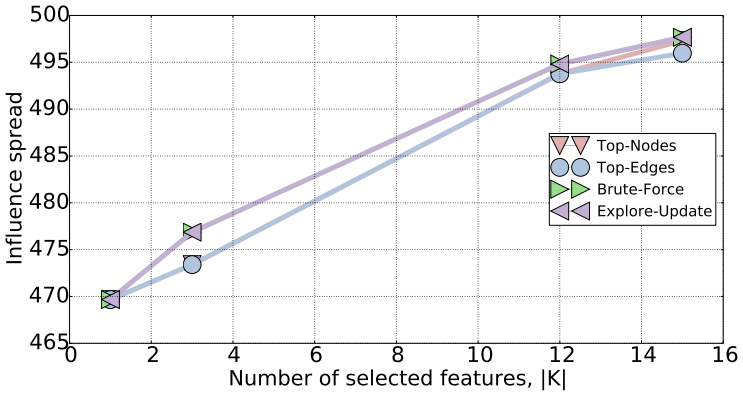



FIGURE 3.7: Influence spread on reduced network.

Next, we select $k = 10$, yielding $\binom{16}{10} = 8008$ possible attribute sets, and calculate, with a new random seed set of size 428, the *rank* of each algorithm's solution among all possible attribute sets: for each attribute set, we compute its influence spread using 10000 MC simulations; we sort sets by their spread values and identify the rank of the solution returned by each heuristic. Table 3.2 presents those ranks. Explore-Update selects the optimal solution, while Greedy with 500 parsimonious MC simulations yields the fifth-best attribute set. The selected attribute sets differ from each other in 2 out of 10 attributes. We obtained similar results for other values of $k$, with Explore-Update always returning the optimal attribute set.

| Algorithm | Rank | Spread |
|---|---|---|
| Explore-Update | 1 | 34.592 |
| Greedy | 5 | 34.114 |
| Top-Edges | 113 | 33.592 |
| Top-Nodes | 113 | 33.592 |
| ... | | |
| — | 8008 | 27.82 |

TABLE 3.2: Algorithm ranking w.r.t. optimal solution.

**Real-World examples.**

Last, we looked into the actual results - seed sets and selected attribute sets of our experiments, with special attention to the VK data set with the multivalency model, and inspected our results. One interesting observation was that those attributes that are liked by seed set users were rarely among the ones selected in the final solution; this fact indicates that our problem makes good practical sense, while a straightforward naive solution of sticking to what is liked by seed nodes does not yield good results.

Nevertheless, selected attributes exhibited a remote, yet unpredictable, resemblance to the attributes liked by seed set nodes. For example, with a group titled "La vie et l'amour" as seed, the selected attributes in our VK network sample included "Home Comfort — Design — Interior Design — Style". With "Psychology of Relations" as seed, the selected attribute set included "Philosophy of Life". Such analogies between seed set and selected attributes, while retrospectively intuitive, would not be derived otherwise; they depend on the way nodes of diverse interests interact within the overall network structure. Such results vindicate our problem motivation.

We also checked how result sets change when we vary $k$. For example, we select 100 out of 431,374 subscribers of "Esoterica YOGA MEDITATION" as a seed set. With $k = 3$, the selected attributes are {"MODA", "La vie et l'amour", "Blog for Men"}. As "Esoterica YOGA MEDITATION" targets primarily women, results such as "MODA" and "La vie et l'amour" are unsurprising. Nevertheless, interestingly, both E-U and Greedy also return "Blog for Men" as a selected attribute, whereas the simple Top-Nodes and Top-Edges heuristics do not. This result shows that our algorithm can select nontrivial attributes.

### 3.2.5 Summary

This section proposed the problem of content-aware influence maximization (CAIM). The goal is to select $k$ attributes that characterize a propagated meme's content, such that its spread across a network from fixed points of departure is maximized, whereby different attribute sets yield different propagation probabilities across network edges. To our knowledge, there is no previous work on this problem. We formulated a content-aware cascade model and showed that the problem is NP-hard and inapproximable, while the influence function is neither submodular nor supermodular. We developed an efficient algorithm for CAIM using bounded local arborescences to calculate influence spread. Our experimental study demonstrates that this Explore-Update algorithm selects topics sets that achieve high spread and is orders of magnitude faster than a conventional Greedy solution resembling algorithms developed for related problems. We also provide evidence that our E-U algorithm can achieve the optimal solution when the number of selected topics is small. In the future, we plan to study other propagation models and investigate the parallelization of Explore-Update.

## 3.3 Influencer recommendation in social networks

### 3.3.1 Introduction

Influence Maximization is the problem of finding influential nodes in the network according to the influence propagation model. It has found use cases in numerous domains such as control of contamination in water networks [169], viral marketing in social networks [170], and content recommendation for users [171]. For example, a marketing company that wants to acquire a small initial set of initial adopters to promote a product to its followers is a typical application of influence maximization. From the outset, this problem has gained a lot of attention and many challenges associated with the selection of the initial set, with the design of realistic propagation model, and with the computation of influence function have been addressed. These works have made a major leap towards understanding the influence in the networks from a purely topological perspective of the network and the influence model therein.

Somewhat parallel track of research has been concerned with vector representations of networks, also known as *embeddings*. There has been a substantial effort to design the node, edge, or (sub)graph vector representations as a native data format for classical machine learning algorithms such as SVM and neural networks. Link prediction [62], network visualization [172], taxonomy recovery [173], and protein classification [174] problems are some examples of applications, where graph embeddings have been used successfully. In these problems, the nodes, or other graph substructures, are embedded in a latent vector space so that machine learning algorithms could be applied to the vectors directly. The net effect of this approach is that after the representation embeddings have been obtained one can focus on the appropriate selection and design of machine learning algorithms that have been studied over the last decades.

In this work, we attempt to apply representation learning algorithms to facilitate the seed set completion of influence maximization. In particular, we assume that we identified a small set of nodes that we can consider as influential. We then seek to extend this seed set by using pairs (embedding, label) from the seed set to train a binary classification algorithm. This problem, which we frame as influence completion, is motivated by the high cost of finding a large set of influential users due to substantial running time or significant use of memory resources of the traditional algorithms [175]. Instead, we use node embeddings to find the extension of the seed set by using only a fraction of all the nodes in the graph.

In Influence Maximization problem one seeks a small set of nodes that would maximize the influence function $\sigma_\mu(S)$ for a given probabilistic graph $G = (V, E, P)$, where $P$

defines the probabilities for every edge, given a set of nodes $S \subset V$ and a diffusion model $\mu$. Diffusion models define the way the information propagates from the initial set $S$ to other nodes in the graphs. An example of a diffusion model is Independent Cascade model, where each node in $S$ has a single and independent attempt to append its neighbor to a set of activated nodes. Due to the space limit, we refer an interested reader to exact definition of Independent Cascade model and influence function in [170].

The classic Influence Maximization (IM) problem, formulated by Kempe et al. [157], has been intensively studied over the last decade. Recently, the focus has shifted to providing realistic definitions of the concept of *influence spread*. Barbieri et al. [163] proposed the *Topic-Aware Influence Cascade* (TIC) and *Topic-Aware Linear Threshold* (TLT) models, which are extensions of the IC and LT models [157]. We briefly discuss the related work about this topic below.

**Classic Influence Maximization.** The first solutions to the IM problem were proposed by Domingos and Richardson [176, 177], yet had no guarantees on influence spread. Then, Kempe et al. [157] formulated the problem based on the Independent Cascade and Linear Threshold propagation models, proved its NP-hardness, and proposed a greedy algorithm with a $(1 - 1/e - \epsilon)$ approximation guarantee. Subsequent works investigated the efficiency and scalability questions, either with heuristics [164, 178, 179] or preserving an approximation guarantee [166, 180–183].

**Topic-Aware Influence Maximization.** Barbieri et al. [163] were the first to look at social influence taking content characteristics into consideration. They proposed methods that learn propagation model parameters such as topic-aware influence strength from a query log of past propagation traces and verified experimentally that a larger influence spread can be engendered when taking item characteristics into consideration via their Topic-Aware Influence Maximization (TIM) models.

Aslay et al. [184] studied online TIM queries; the incentive for this online scenario is that many independent advertisers wish to instantly detect the $k$ most influential users for advertising purposes; each advertisement contains a different set of keywords and hence induces a new probabilistic graph creating a separate TIM instance; the authors proposed an offline-online solution, INFLEX, based on an index used to identify similarities among a new and log TIM queries; pre-computed solutions for log queries are aggregated online so as to provide an approximate solution for a TIM query.

The online TIM problem is also studied in [185, 186]. Chen et al. [185] studied topic-aware influence results on two real networks and utilized the derived properties to form three preprocessingbased algorithms, of which MIS is the best; its main difference from INFLEX is that, in MIS, pre-computed seed sets are based on each separate topic rather

than on a mixture of topics from different log queries. Chen et al. [186] utilized the maximum influence arborescence (MIA) model [164] to achieve high influence spread with a theoretical guarantee. The core idea is to utilize upper- and lower-bounding techniques so that an exact marginal influence is computed only for the most promising nodes. This work provides a state-of-the-art solution for the online TIM problem [184].

Recently, Li et al. [187] proposed a variation on the online TIM problem, namely the alternative problem of Keyword-Based Targeted Influence Maximization (KB-TIM). By KB-TIM, each user is associated with a weighted vector of preferences for distinct keywords, which stand for topics. This vector can be generated by applying topic modeling techniques [188] on aggregated user social activities, such as posts, likes, etc. An advertisement then achieves an impact determined by its own topic-oriented keywords. The KB-TIM problem aims to maximize an advertisement's impact, expressed in terms of its spread to target users relevant to its keywords. The solution in [187] draws from previous work in [183], with the main difference being that, while in [183] $\theta$ users in a sampled Reverse Reachable (RR) set [182] are counted without prejudice, in [187] these sampled users are accounted in terms of exerted advertisement impact; [187] also employs two indexing methods to precompute RR sets for different keywords, so as to obtain RR sets associated with the query keywords on the fly. Nevertheless, results in [187] are not compared to those in [186].

**Influence Maximization with VPD.** Aral and Walker [161] investigated the problem of viral product design under randomized trials focusing on product features like personalized referrals and broadcast notifications. Thereafter, Barbieri and Bonchi [162] studied the problem of influence maximization *in conjunction with* that of viral product design, aiming to detect a combination of seed nodes and product attributes that maximize influence in a network. The proposed solutions are generic methods named *Local Update* and *Genetic Update*; the former is a greedy algorithm allowing for both addition and removal of attributes at each greedy iteration; the latter is a brute-force method that randomly selects a subset of all attributes. By contrast, we investigate the problem of content selection for a post (not a product) as a stand-alone problem in its own right and study its distinctive characteristics.

### 3.3.2 Finding initial influential set

In the Independent Cascade (IC) model [10], each node gets only a single chance to activate its neighbors after it is itself activated. However, it has been shown that the final set of active nodes can be equivalently found by means of a *live-edge graph* [10].

Consider a time step $t$ of the IC diffusion process. A node $u$ that has just become active, is then given a chance to activate a neighbor $v$ along edge $e_{uv}$ with probability $p_{e_{uv}}$. Such an activation of $v$ by $u$ is independent of other nodes in the network. Thus, edge $e_{uv}$ is present in the network, or *live*, with probability $p_{e_{uv}}$, or, otherwise, it is *blocked*. Then the following proposition holds [10]:

**Proposition 3.4.** *A node $v$ is active iff there is a path from the set of initially activated nodes $S_0$ to $v$ made entirely of live edges.*

By Proposition 3.4, we can view the IC diffusion process as follows: we first decide whether an edge is live or blocked, and then, starting from seed set $S_0$, we activate all nodes reachable by other active nodes via live-edge paths. Let $R_G(S)$ be the set of nodes that is reachable from $S$ in graph $G$ and let $G' = (V', E')$, where $V' \equiv V$ and $E'$ is the set of live edges in $E$, i.e. $G'$ is the graph that results by keeping only live edges in $G$. Then, the final active set is the set of nodes reachable from $S_0$ in $G'$, $R_{G'}(S_0)$. We suggest that this alternative view of the diffusion process can be leveraged to measure the potential each node has to activate other nodes, thereby suggesting good choices for seed set $S_0$.

Our algorithm design starts out from the following observation: when we remove blocked edges from a connected undirected graph $G$, we end up with a live-edge graph instance $G'$ having, in general, many disjoint connected components (CCs). Then, with a seed budget of $k$ nodes, we can straightforwardly maximize the influence spread on instance $G'$ itself by selecting one node from each of the top-$k$ CCs by size into the seed set $S$ (breaking ties arbitrarily). That is so because, if we arbitrarily pick up a node $v$ from a connected component $CC$ and include it into the seed set $S$, then all other nodes in $CC$ will be activated by the diffusion process, according to Proposition 3.4. Thus, by choosing any node from each of the $k$ largest CCs, we ensure activating all nodes in those CCs, and hence maximizing influence spread in graph instance $G'$.

Nevertheless, our above observation is valid only for a particular graph instance $G'$ at hand. The solution maximizing influence spread on a particular $G'$ does not necessarily maximize influence spread *in expectation*, for any randomly generated graph $G'$. Yet, it provides a sample of how a diffusion process may look. We propose that, by generating many such graph instances $G'$ and aggregating a score per node from all of them, we can end up with a good approximation of each node's importance in the overall diffusion process. Then, our solution will consist of the $k$ nodes of highest score. This process of assigning scores to nodes is called *score accumulation phase*.

The question that arises from our approach is how exactly we should collect nodes' scores. A score should reasonably depend on the number of appearances of a node

$v$ in the top-$k$ CCs of a graph instance $G'$. At the same time, we should take into consideration the fact that, in each graph instance $G'$, one and only node $v$ per top-$k$ connected component $CC$ is sufficient to lead to a maximum-influence solution on $G'$. Thus, the scores we assign should be *shared* among nodes within the same CC. Putting these two considerations together, we conclude that a reasonable *score function* for node $v$ is $\frac{1}{|CC_v|}$, where $|CC_v|$ is the size of the CC to which $v$ belongs. We claim that, eventually, after $R$ iterations, those nodes that have accumulated the highest scores will be good candidates for inclusion into the seed set $S$ of size $k$.

While the accumulation phase collects scores that indicate good candidates for inclusion, it suffers from a drawback: neighboring nodes may find themselves in the same CC too often, and collect similar scores as each other, even though only one of them would be in most cases sufficient to bring about the same influence effect that both of them exert. Therefore, we reason that, once a node $u$ is selected into the seed set $S$, then, for each edge $e_{uv}$ incident on $u$, the score of node $v$, adjacent node to $u$, who is likely to be in the same CC as $u$, should be penalized in a manner proportional to the score $v$ has accumulated and the probability that $e_{uv}$ is active, $p_{e_{uv}}$. This process of penalizing the scores of selected nodes' neighbors is called *penalization phase*. In detail, in the penalization phase we include nodes to the seed set $S$ by descending score, while, at the same time, for each edge $e_{uv}$ incident on a selected node $u$, we update the score of node $v$, adjacent to $u$, by the formula $s_v = (1 - p_{e_{uv}}) \cdot s_v$.

Algorithm 11 presents our Harvester heuristic. Setting $m = |E|$ and $n = |V|$, we compute connected components in an $O(m)$ BFS, and maintain a Fibonacci heap of the top-$k$ components by size. As there are $O(n)$ CCs in any graph instance, each iteration of the accumulation phase needs $O(m + n + k \log(n))$ time, hence $O(R(m + n + k \log(n))$ for $R$ iterations. In the penalization phase, we store the score values in a Fibonacci heap, hence need $O(k \log(n) + m)$, where $O(m)$ stands for penalization operations across edges. Thus, the time complexity of Algorithm 1 is $O(R(m + n + k \log(n))$, dominated by the accumulation phase.

**Algorithm 11** Harvester($G$, $k$)

initialize $S = \emptyset$
/* accumulation phase */
$s_v = 0$ for all $v \in V$
**for** $i = 1$ to $R$ **do**
  generate $G'$ keeping each edge $e \in G$ with prob. $p_e$
  find top-$k$ connected components $CC_{G'}$ in $G'$
  $W = \emptyset$
  **for** $j = 1$ to $k$ **do**
    $CC_{G'}(j) = argmax\{|CC_{G'}(i)|, CC_{G'}(i) \in CC_{G'} \setminus W\}$
    $W = W \cup CC_{G'}(j)$
    **for** node $v \in CC_{G'}(j)$ **do**
      $s_v += 1/|CC_{G'}(j)|$
    **end for**
  **end for**
**end for**
/* penalization phase */
**for** $i = 1$ to $k$ **do**
  select $u = argmax(s_v | v \in V \setminus S)$
  $S = S \cup u$
  **for** each edge $e_{uv}$, $v \in V \setminus S$ **do**
    $s_v = (1 - p_{e_{uv}}) \cdot s_v$
  **end for**
**end for**
**return** $S$

### 3.3.3 Influence spread and Running time

**Data sets.** For evaluation purposes we run our algorithms against two real-world networks. The first network, NetHEPT, is an academic collaboration network in "High Energy Physics - Theory" section of the e-print arXiv[5]. The same data set is used in [157], where nodes represent authors and edges are papers' co-authorship relationships. Our second data set, taken from the archive of Jure Leskovec, is the Gnutella peer-to-peer file sharing network from August 2002[6]. In this network nodes are hosts in the Gnutella network and edges are connections between the hosts. We refer to these networks as NetHEPT and Gnutella networks. Statistics on them are provided in Table 3.3.

[5] http://www.arXiv.org
[6] http://snap.stanford.edu/data/p2p-Gnutella09.html

TABLE 3.3: Statistics for two real data sets.

| Data set | HepNEPT | Gnutella |
|---|---|---|
| Number of nodes | 15K | 8K |
| Number of edges | 31K | 26K |
| Average degree | 4.12 | 6.41 |
| Maximum degree | 64 | 102 |
| Number of CC | 1782 | 6 |
| Maximum CC size | 6794 | 8104 |
| Median CC size | 2 | 2 |
| Clustering coefficient | 0.49 | 0.009 |
| Diameter (longest shortest path) | 22 | 10 |

**Cascade models.** We compare all algorithms under the general IC model with non-uniform propagation probabilities, using the following to assign propagation probabilities to edges of a graph:

- *Categories model*: Based on the assumption that nodes with high degrees are likely to have larger influence on their neighbors, the Categories model is built in the following manner: let $\{0.01, 0.02, 0.04, 0.08\}$ be a set of possible probability values. We sort nodes by their degrees in ascending order and divide them into four equal-sized chunks (except, maybe, the last chunk). We map the set of values to the chunks so that nodes with low degrees have value 0.01, while nodes with high degree have highest possible value 0.08. For an edge $(u, v)$ we pick a propagation probability at random between value of $u$ and value of $v$.

**Algorithms.** We compare our Harvester heuristic against state-of-the-art scalable heuristics for the influence maximization, as well as some baseline heuristics.

- *Harvester*: Our Algorithm 11 based on the aggregation of live-edge graphs. We use $R = 500$ iteration; we found out that our results showed little difference in influence spread for values beyond that.
- *GDD*: This is a generalized version of DegreeDiscountIC heuristic of [189], fine-tuned for non-uniform propagation probabilities as follows. Let $d_v$ be the degree of node $v$ and $t_v$ is the number of edges incident to active neighbors of $v$. We denote $p_1, p_2, \ldots, p_{t_v}$ as the set of propagation probabilities on edges incident

to active neighbors of $v$ and $p_{t_v+1}, \ldots, p_{d_v}$ as the set of propagation probabilities on edges incident to inactive neighbors. Then a score for node $v$ is calculated as $(1-p_1)(1-p_2)\ldots(1-p_{t_v})(1+p_{t_v+1}+\ldots+p_{d_v})$, expressing[7] the expected marginal gain of activated nodes we can get by activating $v$. Then, at each iteration, we select the hitherto inactive node of highest score and update the scores of its neighbors according to the above formula. This heuristic generalizes the case of uniform probabilities presented in [189], where the score of a node $v$ is $(1-p)^{t_u}(1+(d_u-t_u)p)$. We confirmed that our accurate generalization outperforms the original DegreeDiscountIC for the whole range of $k$.

- *PMIA(θ)*: The algorithm defined for the prefix-excluding maximum influence arborescence model in Chen et al. [190] with parameter $\theta = 1/20$.
- *Degree*: A heuristic that selects the $k$ nodes of largest degree.
- *Random*: A heuristic that selects $k$ random nodes in the graph.

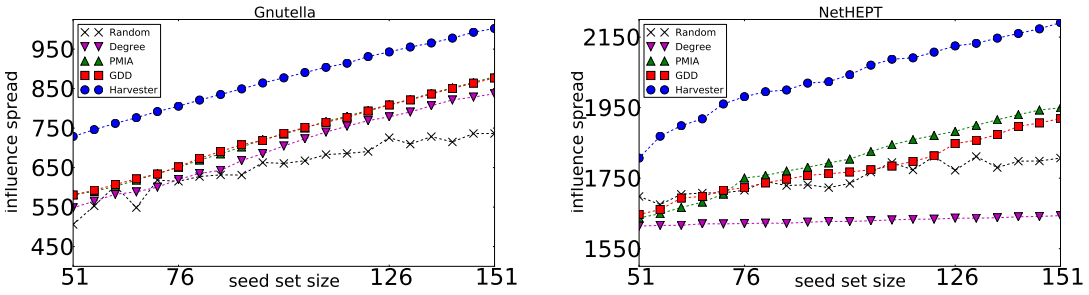



FIGURE 3.8: Influence spread on Gnutella and NetHEPT networks for Categories model.

Figure 3.8 shows our influence spread results for the Gnutella and NetHEPT networks.

For the Gnutella network, the Random heuristic performs poorly as a baseline compared to Harvester (it is 33.% lower for Categories model). Results for Degree heuristic are worse by 25.4% than those obtained by Harvester. PMIA and GDD achieve similar results (the discrepancy of percentage differences among all three models is no more than 1%), which are, however, lower than those of Harvester: for the Categories model, the Harvester has 20% and 19.8% higher results than PMIA and GDD, respectively.

For the NetHEPT network, Random algorithm shows poor results again (16.3% lower than Harvester), while the average difference of Degree heuristic compared to Harvester is 25.1%. Interestingly, Random performs even better than Degree for Random and

[7] This formula is valid under the assumption of non-parallel edges among nodes; in the case of parallel edges among a pair of nodes $(u, v)$, the probability $p_i$ in the second factor has to be substituted by $1-\prod_{j=1}^{w}(1-p_j)$, where $w$ the number of parallel edges between a pair of nodes $(u, v)$.

Categories models; this means that selecting nodes uniformly at random can result in larger influence coverage than selecting high-degree nodes for those models, as selecting the nodes of top degrees may be an overkill solution. PMIA and GDD achieve, again, similar spreads for the three models. However, Harvester obtains 12.9% and 14.5% in the Categories model.

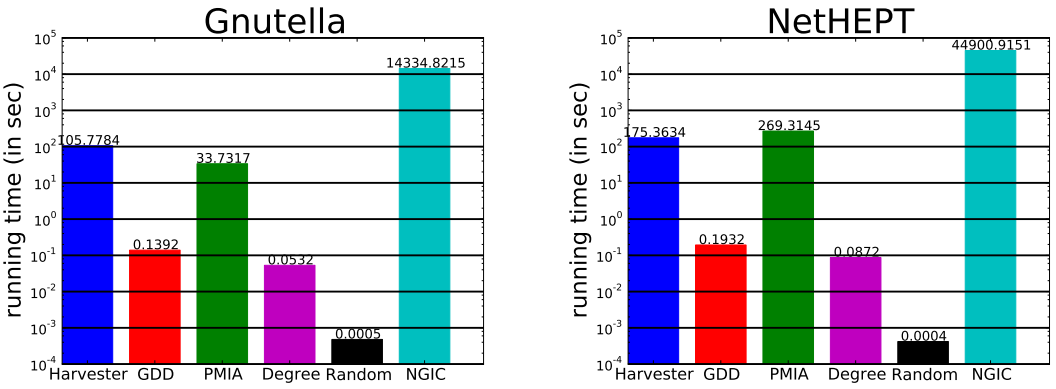



FIGURE 3.9: Running time on Gnutella and NetHEPT networks for Categories model.

Figure 3.9 reports the runtime of different algorithms when finding seed set $S$ for $k = 51$. Note that the $y$-axis is in log scale. The experiments are run on an Intel Core i5-2450M CPU @ 2.50GHz with 6G memory. All results are measured on reasonably efficient implementations of the different algorithms. For the sake of comparison, we also add a representative improvement of the original greedy approach of Kempe et al. [157], namely the *NewGreedyIC* algorithm (NGIC) by Chen et al. [189] with $R = 1000$.

In the Gnutella network, Degree, Random, and GDD algorithms present the best results, finding a seed set in less than a second. PMIA completes in 34 seconds for Categories models, respectively, while Harvester takes slightly more time to finish, 106 seconds, yet more robust to variations in the underlying probability model. Notably, NewGreedyIC takes about 4 hours to find a seed set $S$ on a comparatively small network.

The picture is similar for the NetHEPT network, where the GDD, Degree, and Random heuristics run faster. PMIA finishes is seed node selection in 269 seconds, while Harvester finishes in 175 seconds, again more robustly on the underlying model; notably, on this *larger* network Harvester is faster than the fastest state-of-the-art PMIA heuristic on the more demanding underlying Categories model, corroborating the robustness of Harvester. As previously, NewGreedyIC takes several hours to finish, terminating in more than 12 hours for MultiValency, Random, and Categories models, respectively.

In a nutshell, GDD, Degree, and Random have runtimes within a second, because these are very simple models that do not require much computation. Harvester and PMIA require 3 orders of magnitude more but they achieve better quality in influence spread.

In what follows we explain how we can decrease the running time if the initial seed set is already found.

### 3.3.4 Influence Completion

We are interested in extending a small seed set of influential nodes by an additional set of marginally influential nodes. We define the problem as follows:

**Definition 3.5** (Influence Completion problem)**.** Let $G$ be a probabilistic graph and $S$ is the set of the most influential nodes of size $k$. We are asked to find a set $T$ of size $k+l$ and $S \subset T$ so that the influence function $\sigma_\mu(T)$ is maximized.

The problem is motivated when we are already given a seed set of initial adopters and we seek to find an additional extension of this set to maximize the value. At the same time, graph embeddings are useful data representation, which can be used for multiple applications such as graph classification [174] and clustering [191]. Therefore we aim to use node embeddings for the learning influence of each node by observing the relationship between the node representations and the set of first influential users. The intuition behind it is that if node embeddings define local and global properties of a graph then given a small set of influential nodes we can capture the "influential" topology of the network by learning a model on influential node embeddings.

Influence Completion problem is NP-hard problem as we can remove the set $S$ and the corresponding edges from $G$, in which case the problem is reduced to Influence Maximization, which is known to be NP-hard. However, one can first obtain some ground-truth seed set, either as an output of Influence Maximization algorithm or by domain-specific knowledge, and then use the embeddings of the nodes to acquire new "similarly influential" nodes, which we show next.

Our approach relies on the already identified initial set of nodes, which we use to build the dataset for our supervised classification model. The algorithm **InfEmb** takes as input a graph $G$, a seed set $S$, and an integer $l$ and outputs an extended set $T \supset S$ that attempts to maximize the influence function $\sigma(T)$.

TABLE 3.4: Networks used in experiments. The columns are: name of the network, number of nodes and edges, average clustering coefficient, diameter.

| Dataset | Directed | Nodes | Edges | ClusteringCoefficient | Diameter |
|---|---|---|---|---|---|
| GRQC | No | 5242 | 14496 | 0.52 | 17 |
| Wiki | Yes | 7115 | 103689 | 0.14 | 7 |
| FB | No | 4039 | 88234 | 0.60 | 8 |

**Algorithm 12 InfEmb** algorithm

**Input:** graph $G$, seed set $S$, and integer $l$.

**Output:** set $T$, s.t. $|T| = |S| + l$

1: Compute a feature map $\phi : v \mapsto R^d$ for all nodes in $G$.

2: Train a classification model $f : \mathbb{R}^d \mapsto [0, 1]$ of influence score for each node.

3: $T = S$

4: Append top-$l$ nodes to $T$ based on influence score that are not yet in $S$.

5: **return** $T$

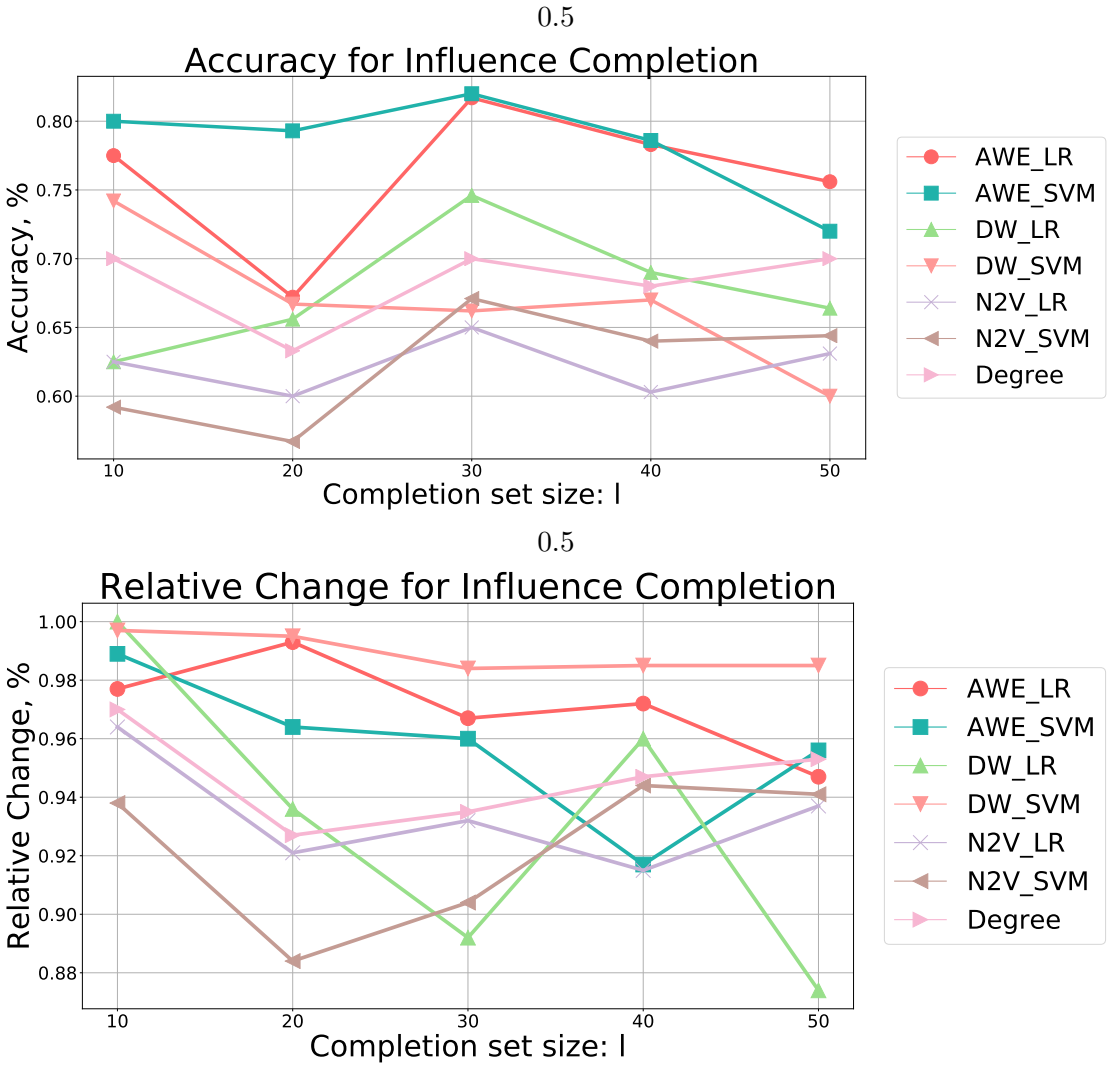
The algorithm **InfEmb** has essentially three steps. In the first step (Line 1), the algorithm computes embeddings for each node in the graph, including those in $S$. In the experimental section, we deal with several representation algorithms that encode graph structure, as well as, direct computation of the statistics related to the node neighborhoods and seed set. In the second step (Line 2), the algorithm trains a classification algorithm $f$ that outputs influence score for each node in the graph. For this, we use the nodes in $S$ as the positive labels. We sample the negative labels from the remaining nodes reversely proportional to their degrees. That is the nodes that have a small degree will most likely appear as negatives, which is motivated by the fact that the degree of a node serves as a good proxy for the influence score [170]. The influence score is then the likelihood that the algorithm $f$ assigns to each node to be a positive label. In experiments, we compare different options for a feature map $\phi$ and a classification model $f$. In the third step (Line 4), the algorithm includes top-$l$ nodes that are not yet in the given set $S$. We evaluate the quality of the node embeddings in the following section.

### 3.3.5 Experimental evaluation

For evaluation of our approach, we obtain solutions to Influence Completion problem on several datasets and compare it against several baselines using several evaluation metrics.

**Datasets.** We use three publicly available datasets, GRQC[8], Wiki[9], and Facebook[10].

FIGURE 3.10: Accuracy and Relative Change in influence spread for **GRQC** dataset.

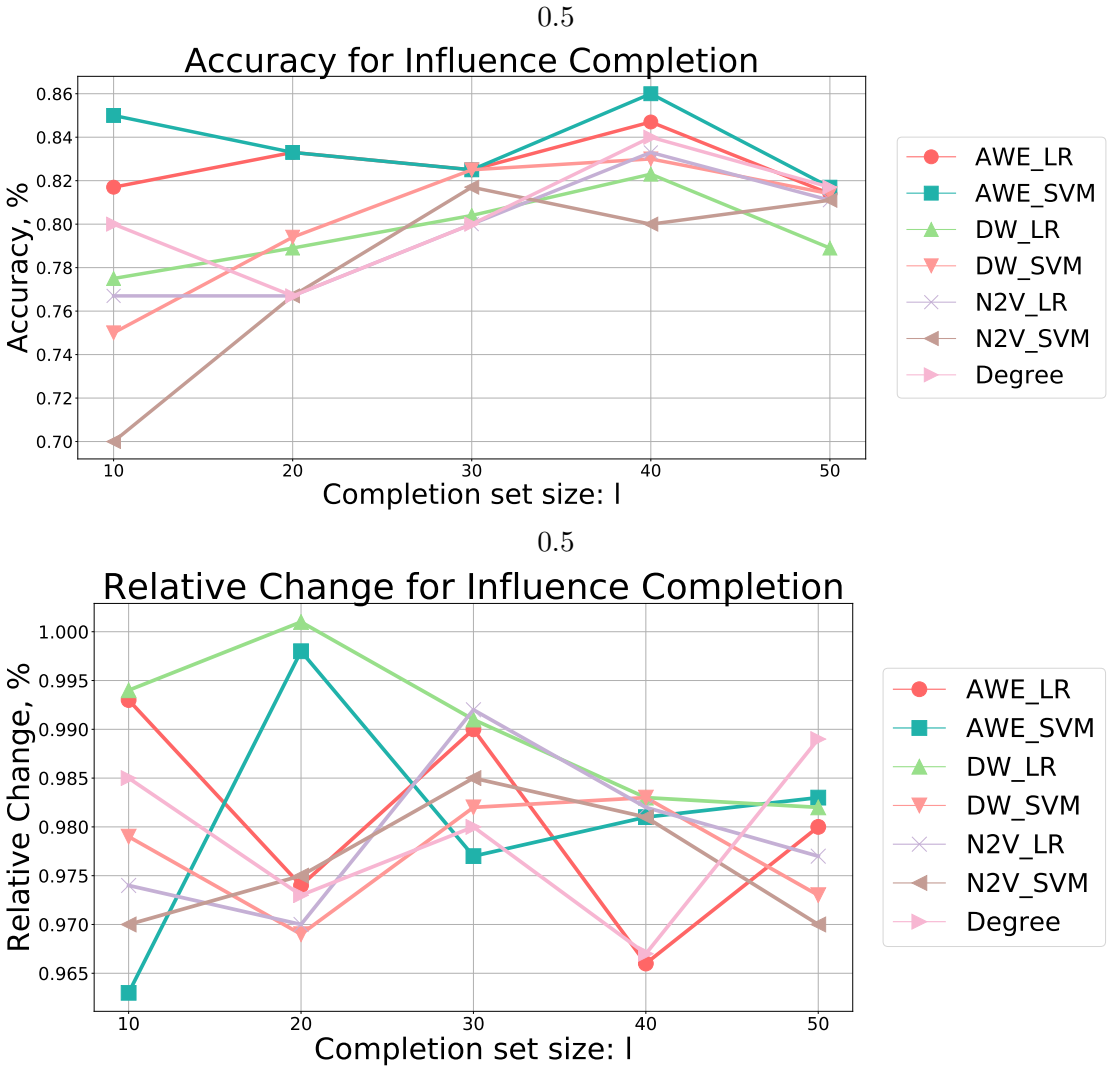



GRQC dataset is a collaboration network in General Relativity and Quantum Cosmology area authors are nodes and edges correspond to their co-authorship of the paper. Wiki network is a graph where nodes are the users in Wikipedia and edges are among the users who give the votes to become administrators. Facebook is a social network with users as nodes and edges denote their friendships in the network. Each edge in undirected network is replaced by two directed edges. The datasets vary in their structural properties as shown in the Table 3.4.

**Propagation models.** Similar to previous research we use Independent Cascade model to determine influence propagation in the network. In this model, the influence propagates from one node to another node with a probability of the edge. We assigned

[8]http://snap.stanford.edu/data/ca-GrQc.html
[9]http://snap.stanford.edu/data/wiki-Vote.html
[10]https://snap.stanford.edu/data/ego-Facebook.html

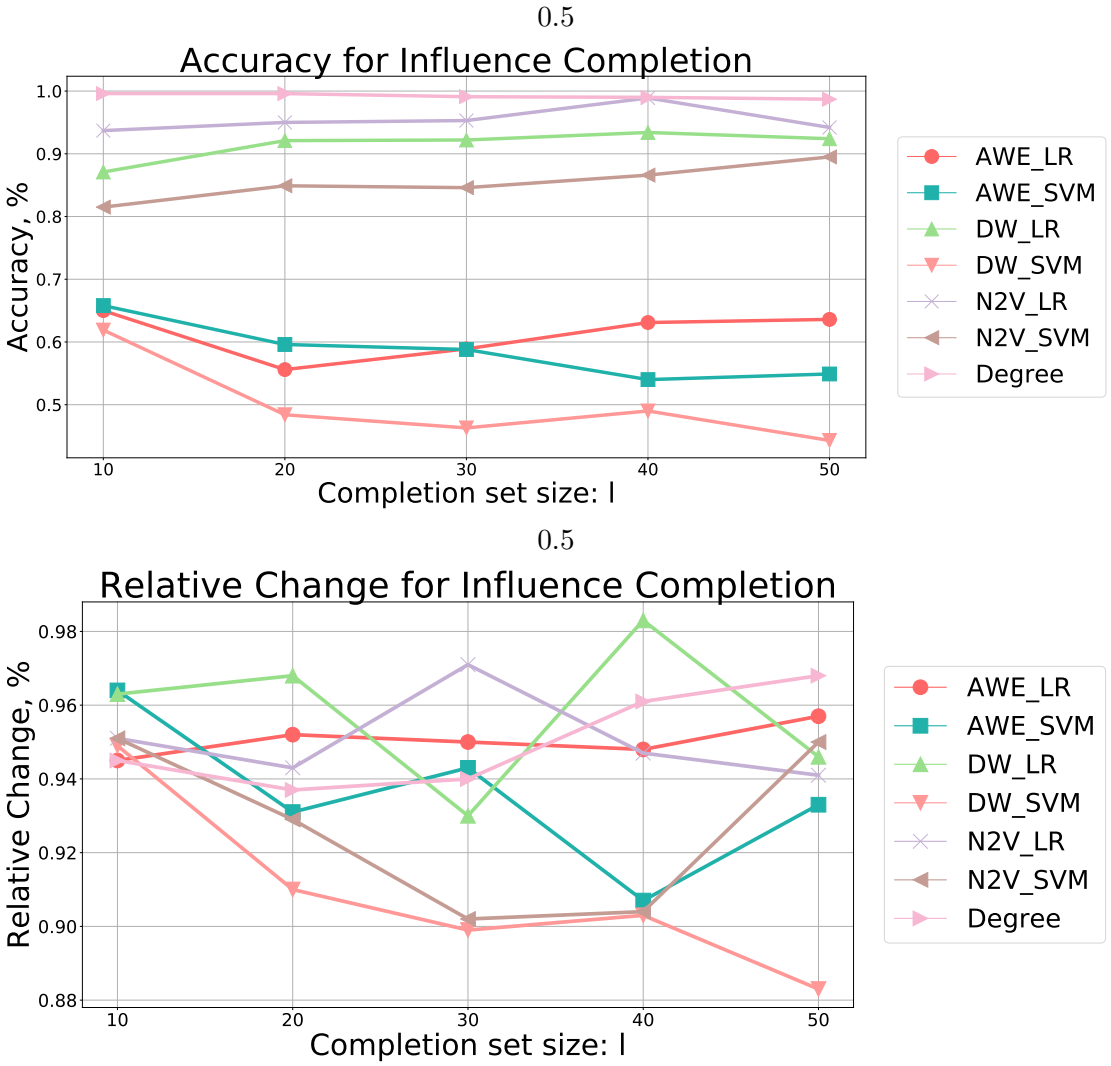
FIGURE 3.11: Accuracy and Relative Change in influence spread for **Wiki** dataset.




probabilities according to a weighted cascade model [170], i.e. an edge $(u, v)$ has a probability $1/d_v$, where $d_v$ is degree of a node $v$.

**Algorithms.** We test our algorithm **InfEmb** with different choices of functions $\phi$ and $f$. We select the feature map $\phi$ from the embedding methods node2vec [62], DeepWalk [192], and AWE [174]. We denote the embeddings as N2V, DW, and AWE respectively. The first two are popular algorithms for node embeddings that create vector representations from the random walks in the graph. AWE is a version of anonymous walk embeddings [174] adapted for node embeddings. It has been shown [119] that the distribution of anonymous walks for a single node is sufficient to reconstruct a neighborhood around this node, i.e. AWE node embeddings encode the local graph structure around the node. Hence, if the propagation of influence happens within a neighborhood of influential nodes, this method could distinguish the influence of nodes by looking at the

FIGURE 3.12: Accuracy and Relative Change in influence spread for **FB** dataset.




vector representations of the neighborhoods. For the AWE we compute a distribution of anonymous walks for each node starting at that node with a fixed length $L = 4$. In addition to the vectors obtained by the feature maps $\phi$, we append information about the number of influential nodes in the neighborhood and its degree to the node embeddings, which should eliminate the addition of two influential nodes to the seed set that are located in one-hop distance.

We also experiment with the classification function $f$ and select it as SVM or Logistic Regression (LR). We use the prediction values from classification model $f$ to rank all the nodes in the graph and we select top-$l$ nodes from this list that are not yet in the set $S$. We compare our model with a baseline algorithm, Degree, where the top-nodes are selected from the ranked list according to their degrees.

**Evaluation metrics.** We measure the performance of the proposed algorithms by two metrics related to influence maximization. To measure *accuracy* of the approach, we train the model on $k$ first positive and $k$ negative nodes and compute the overlap between the top $[k+1,\ k+2,\ \ldots,\ k+l]$ nodes returned by the model and the one that we consider as ground-truth. We set $k = 10$ and vary the completion set size $l$ in the range $[10,\ \ldots,\ 50]$. To get ground-truth nodes one can either use a standard traditional influence maximization algorithm or rely upon domain knowledge. We use D-SSA algorithm [193] that proved to be among state-of-the-art approaches for influence maximization. Intuitively, the accuracy of the model shows the discrepancy between the predictions of the classifier and the ground-truth values that are hard to obtain. Additionally, in some scenarios we want to measure the *relative change* (RC) of influence spread by the model's seed set compared to the given oracle seed set. While the model may miss some of the top-influential nodes, the returned by model nodes can still be highly influential and therefore RC measures the discrepancy of influence spread. More formally, we use first $k$ ground-truth nodes and top-$l$ nodes returned by the model to compute influence spread $s_{cl}$. We then compute influence spread $s_{gr}$ of ground-truth $k+l$ nodes and define relative change of the model as $\frac{s_{gr}}{s_{cl}}$.

**Results.** Results for GRQC dataset are presented in Figure 3.10. On the left, the accuracy, i.e. the overlap between the ground-truth seed set and the set returned by the classifier, is presented. AWE is consistently on the top with 10% and 2% uplift in accuracy compared to degree approach for $l$=10 and $l$=50 respectively. The value of accuracy decreases from 80% to 0.68% for AWE-SVM model. DW and N2V embeddings, in general, perform comparably with the degree approach. We also see that the classification models perform similarly for all algorithms in terms of accuracy, while SVM is generally more robust across the whole range of values $l$.

The Relative Change (RC), i.e. the influence spread of the ground-truth set divided by the influence spread of the influence set of the algorithm, is presented on the right. RC for AWE and DW is at the top, with 0.97 and 0.99 change in influence spread respectively for $l$=10. RC for DW and N2V models is above 80% and in general comparable with the baseline degree algorithm.

Results for Wiki and FB datasets are presented in Figures 3.11 and 3.12 and share the main insights as for the GRQC dataset. In particular, for Wiki dataset the accuracy and relative decrease for AWE is consistently at the top, being always higher 80% and 0.9 for two metrics respectively. N2V and D2W performs similarly to the degree baseline with 78%-80% accuracy and $0.95 \pm 0.16$ relative change. For FB dataset, the accuracy for degree baseline is quite high, which implies there is a strong correlation between the degree and the influence of the node. Yet, in terms of relative change, the list of nodes

of InfEmb algorithm is still highly influential, especially when it comes to predicting the first nodes with $l = 10$. We also obtained the results for the random baseline, when the nodes for completion set are taken uniformly at random across all nodes in the graph. We didn't include it in the figures for visibility reasons, however, the accuracy and relative decrease are significantly smaller compared to other algorithms, being lower than 1% and 50% for two metrics respectively.

### 3.3.6 Summary

This section reexamined the classical influence maximization problem in a network, as well as its dual variant, the seed minimization problem. We proposed Harvester, an efficient heuristic, based on score aggregation by multiple live-edge graphs, that can be gracefully customized for both problem variants. Our algorithm achieves good solutions for both problems, while its strength shines in the case of the seed minimization problem. On that problem, a seed set size is not given in advance as a constraint but has to be discovered as a minimization objective. The minimal size a given algorithm can achieve has to be conventionally discovered by an iterative, trial-and-error search process, each iteration of which runs a costly cascade process to estimate the expected influence spread. Harvester tackles this problem in a very elegant manner; while aggregating scores by live-edge instances, it also inherently provides a good prediction of the minimal seed size in advance. Our experimental study on real-world data demonstrates the effectiveness and robustness of this technique in reducing the required number of search iterations by a factor of two, while it also shows that both Harvester algorithms provide efficient, scalable, and competitive solutions in comparison to the state-of-the-art scalable approaches, especially on large values of seed set size and target influence spread with complex graph models. In the future, we plan to investigate how our general approach can be adapted for other influence diffusion models.

# Chapter 4

# Conclusion

## 4.1 Synopsis

In this dissertation we have made contributions to the computational graph theory in several ways: proposing new graph and node embeddings (Chapter 2); formulating and proposing an algorithm to a hard optimization problem on graphs and proposing an algorithm based on embeddings for combinatorial problems (Chapter 3. Besides its scientific novelty, in the last section, the work provides an example of merging two principles of solving problems on graphs, traditional greedy approach, and modern machine learning algorithms.

In Chapter 2 we introduce a concept of anonymous walks in undirected graphs. The distribution of anonymous walk becomes a complete graph invariant allowing us to have an invariant topological measure to compare different graphs. We study some of the properties of anonymous walks and show an algorithm to obtain canonical labeling that can be used for graph isomorphism problem. Our contributions include the analysis of the complexity of computing such distribution and exact upper bounds on the number of samples to approximate exact distribution. We also propose a neural network algorithm to compute embeddings based on the sampled positive and negative pairs of anonymous walks to compute which anonymous walks occur together and therefore compute a graph embedding. Our methods are unsupervised, do not require any additional data, and show competitive performance in the graph classification task, compared to state-of-the-art supervised methods and graph kernels. We conclude the section with an application to medical diagnostics of predicting depression and epilepsy based on the fMRI images of a brain, where we input adopted anonymous walk embeddings for nodes to the classifiers.

In the next Chapter 3 we formulate another combinatorial problem on graphs of product recommendation. We show that this highly relevant problem is NP-hard and moreover

finding any meaningful approximation to this problem is also NP-hard. Having such strong inapproximability, we show that the greedy approach for this problem achieves better performance than several baselines. We propose two modifications to the greedy approach that do not sacrifice the quality of the solutions but reduces the running time by an order of magnitude. We continue by considering a well-known problem of influence maximization, which has a direct relation to the aforementioned problem of product recommendation. We propose a solution that is efficient for undirected graphs and which naturally adapts to the problem of seed minimization. Next, we show that we can effectively reuse the node embeddings we developed in the previous sections to facilitate the hard problem of finding an influential set. For this, we search for similarities of the already discovered influential set of nodes and the remaining not yet activated nodes using vector representations of the nodes and learning supervised classifier to predict a probability of a node being influential.

## 4.2 Future directions

In this section, we discuss several of the topics that we think have a major impact on coupling traditional combinatorial methods with the efficiency of machine learning tools.

**Pretrained graph models** Similar to the trends of machine vision and natural language processing there is room for research on pretraining embeddings for graphs on large corpus of data and then using these models as a part of the learning pipeline for the task. Embeddings based on shallow models have been a key to understanding that vector representations of graphs are useful for many graph applications and have brought many novel ideas for graph research scientists. Pretrained graph models can be the next step of improving quality of solutions for many tasks. An additional benefit of having pre-trained models is a standardized benchmark of datasets and models for different problems on graphs.

**Solving NP-hard problems** This area presents a particularly interesting topic as it combines years of computer science research on resolving hard problems with machine learning approaches. One of our results related to solving graph isomorphism problems falls into this domain; however, there are many open questions and unsolved tasks. Broadly speaking the solutions can be divided into two classes, based on the expert demonstration and the agents experience. The first class of solutions depends on the solved instances of the problems (e.g. SAT) provided by the solver. As the problems are NP-hard its very prohibitive to obtain the solutions for many instances, so either the algorithms utilize already collected instances of the problems or the solvers are used for some small enough sizes to obtain results fast enough. The benefit of such methods that

they can learn to imitate the expert decisions for unseen examples and even generalize to bigger sizes of the problems for which solvers do not cope. Another class is based on reinforcement learning ideas that guided by the reward the agent will be able to update its policy correctly, essentially improving over random search strategies. A large drawback for such problems is that it requires a lot of experience rollout to happen before the agent can effectively select the decisions; however such approaches are at the moment the only way to solve problems where the optimal sets are hard to obtain. For the latter class, its even harder to solve decision problems, where a binary answer is expected instead of the continuous value that allows us to compare it to the optimal solution. For example, discussed in this thesis a problem of finding maximal cover in a graph requires us to find one cover among a huge number of alternatives and getting an approximation to the longest path problem does not approach us to the right decision. This extreme difficult problems where the search space is big and the required solution occupies an infinitesimal fraction of it urges new tools and ideas to appear for the generation to come.

# Appendix A

# History of Weisfeiler-Lehman Algorithm

Weisfeiler-Leman algorithm was developed by two Soviet mathematicians, Boris Weisfeiler and Andrew Leman, who were working in the same scientific group supervised by A.S. Kronrod. The algorithm has high expressive power and rarely fails in practice, which led to believe that it works in general for the short period of time. Later it was realized that there are special types of graphs, where the algorithm may miss non-isomorphic graphs, but its existence still has a large impact on the field of graph isomorphism and representation. The biography of Boris Weisfeiler can be found online[1]; however, for Andrew Leman the information is very scarce, so we decided to include it in this thesis, collecting it from the friends and colleagues of Andrew.

Andrew Leman (3 September 1940 - 2012) obtained specialist degree (equivalent to M.Sc.) in mathematics from Moscow State University in 1962, where he studied in the same group with Boris Weisfeiler. In the USSR he worked in the Institute of Theoretical and Experimental Physics (ITEF) during 1962-1968, then in Institute of Control Sciences (IPU) from 1968 to 1976, and after in the Soviet Institute of System Analysis (ISA) from 1976 to 1990. His first Ph.D. thesis was about enumeration of all cellular algebras and associative schemes of small order. It's during this time that he was working on graph isomorphism problem and co-authored WL algorithm. Unfortunately due to the conflict between the head of the Ph.D. defense committee and Andrew's supervisor, his thesis was never officially defended. He wrote a second thesis on the interface adaptation, which led to the development of heterogeneous programs into a big complex software. After the collapse of the iron curtain, he immigrated to the USA, living there for the rest of his life and working in several Silicon Valley startups. Together with his friends he

[1] For example here or here

founded a company Cognitive Technology Inc. that was developing optical recognition systems. His last company was Invitae, a bioinformatics startup that makes analysis genes.

Andrew Leman was a very versatile programmer, helping on the range of problems from simulating quantum effects to developing the first world champion in chess games. He is one of the co-authors of the first Soviet database system. During his studies he also regularly participated in the mathematical Olympiad and he wrote a handbook of the challenging mathematical problems for students that is widely used in preparation. His friends remember him as a reliable and loyal person with a strong sense of humor and a big talent in math.

# Bibliography


[1] Leonhard Euler. Solutio problematis ad geometriam situs pertinentis. *Commentarii academiae scientiarum Petropolitanae*, pages 128–140, 1741.

[2] John Michael Harris, Jeffry L Hirst, and Michael J Mossinghoff. *Combinatorics and graph theory*, volume 2. Springer, 2008.

[3] Donald L. Kreher and Douglas R. Stinson. Combinatorial algorithms: Generation, enumeration, and search. *SIGACT News*, 30(1):33–35, March 1999. ISSN 0163-5700. doi: 10.1145/309739.309744. URL http://doi.acm.org/10.1145/309739.309744.

[4] Matthew Richardson and Pedro Domingos. Markov logic networks. *Machine learning*, 62(1-2):107–136, 2006.

[5] Muhan Zhang and Yixin Chen. Link prediction based on graph neural networks. In *Advances in Neural Information Processing Systems*, pages 5165–5175, 2018.

[6] Nino Shervashidze, S. V. N. Vishwanathan, Tobias Petri, Kurt Mehlhorn, and Karsten M. Borgwardt. Efficient graphlet kernels for large graph comparison. In *Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics, AISTATS 2009, Clearwater Beach, Florida, USA, April 16-18, 2009*, pages 488–495, 2009.

[7] Paul J Kelly et al. A congruence theorem for trees. *Pacific Journal of Mathematics*, 7(1):961–968, 1957.

[8] Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M. Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12:2539–2561, 2011.

[9] J. A. Bondy. *A Graph Reconstructor's Manual*, page 221252. London Mathematical Society Lecture Note Series. Cambridge University Press, 1991. doi: 10.1017/CBO9780511666216.009.

[10] David Kempe, Jon Kleinberg, and Éva Tardos. Maximizing the spread of influence through a social network. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '03, pages 137–146. ACM, 2003. ISBN 1-58113-737-0.

[11] Karsten M. Borgwardt and Hans-Peter Kriegel. Shortest-path kernels on graphs. In *Proceedings of the 5th IEEE International Conference on Data Mining (ICDM 2005), 27-30 November 2005, Houston, Texas, USA*, pages 74–81, 2005.

[12] Harold N. Gabow and Shuxin Nie. Finding long paths, cycles and circuits. In Seok-Hee Hong, Hiroshi Nagamochi, and Takuro Fukunaga, editors, *Algorithms and Computation*, pages 752–763. Springer Berlin Heidelberg, 2008.

[13] N. Biggs. *Algebraic Graph Theory*. Cambridge University Press, 2nd edition, 1993.

[14] C. Godsil and G. Royle. *Algebraic Graph Theory*, volume 207 of *Graduate Texts in Mathematics.* volume 207 of Graduate Texts in Mathematics. Springer, 2001.

[15] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition*. 2009.

[16] J. E. Hopcroft and J. K. Wong. Linear time algorithm for isomorphism of planar graphs (preliminary report). In *Proceedings of the Sixth Annual ACM Symposium on Theory of Computing*, STOC '74, 1974.

[17] Eugene M. Luks. Isomorphism of graphs of bounded valence can be tested in polynomial time. In *Proceedings of the 21st Annual Symposium on Foundations of Computer Science*, SFCS '80, 1980.

[18] László Babai. Graph isomorphism in quasipolynomial time. *CoRR*, abs/1512.03547, 2015. URL http://arxiv.org/abs/1512.03547.

[19] Brendan D. Mckay and Adolfo Piperno. Practical graph isomorphism, ii. *J. Symb. Comput.*, 2014.

[20] Jin-yi Cai, Martin Fürer, and Neil Immerman. An optimal lower bound on the number of variables for graph identifications. *Combinatorica*, 1992.

[21] Pascal Schweitzer. Problems of unknown complexity.

[22] Mark K Goldberg. A nonfactorial algorithm for testing isomorphism of two graphs. *Discrete Applied Mathematics*, 6(3):229–236, 1983.

[23] László Babai, William M Kantor, and Eugene M Luks. Computational complexity and the classification of finite simple groups. In *24th Annual Symposium on Foundations of Computer Science (sfcs 1983)*, pages 162–171. IEEE, 1983.

[24] Merrick Furst, John Hopcroft, and Eugene Luks. Polynomial-time algorithms for permutation groups. In *Proceedings of the 21st Annual Symposium on Foundations of Computer Science*, SFCS '80, 1980.

[25] I. S. Filotti and Jack N. Mayer. A polynomial-time algorithm for determining the isomorphism of graphs of fixed genus. In *Proceedings of the Twelfth Annual ACM Symposium on Theory of Computing*, STOC '80, 1980.

[26] László Babai, Paul Erdős, and Stanley M. Selkow. Random graph isomorphism. In *SIAM Journal on Computing*, 1980.

[27] Yaohui Lei. A survey of graph and subgraph isomorphism problems. 2004.

[28] Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(Sep):2539–2561, 2011.

[29] Christopher Morris, Martin Ritzert, Matthias Fey, William L Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. *arXiv preprint arXiv:1810.02244*, 2018.

[30] Neil Immerman and Eric Lander. Describing graphs: A first-order approach to graph canonization. In *Complexity theory retrospective*, pages 59–81. Springer, 1990.

[31] Vikraman Arvind, Johannes Köbler, Gaurav Rattan, and Oleg Verbitsky. On the power of color refinement. In *International Symposium on Fundamentals of Computation Theory*, pages 339–350. Springer, 2015.

[32] Christoph Berkholz, Paul Bonsma, and Martin Grohe. Tight lower and upper bounds for the complexity of canonical colour refinement. *Theory of Computing Systems*, 60(4):581–614, 2017.

[33] László Babai and Ludik Kucera. Canonical labelling of graphs in linear average time. In *20th Annual Symposium on Foundations of Computer Science (sfcs 1979)*, pages 39–46. IEEE, 1979.

[34] Martin Fürer. On the combinatorial power of the weisfeiler-lehman algorithm. In *International Conference on Algorithms and Complexity*, pages 260–271. Springer, 2017.

[35] Brendan D McKay and Adolfo Piperno. Practical graph isomorphism, ii. *Journal of Symbolic Computation*, 60:94–112, 2014.

[36] Tommi Junttila and Petteri Kaski. Engineering an efficient canonical labeling tool for large and sparse graphs. In *2007 Proceedings of the Ninth Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 135–149. SIAM, 2007.

[37] José Luis López-Presa, Antonio Fernández Anta, and Luis Núñez Chiroque. Conauto-2.0: Fast isomorphism testing and automorphism group computation. *arXiv preprint arXiv:1108.1060*, 2011.

[38] Paolo Codenotti, Hadi Katebi, Karem A Sakallah, and Igor L Markov. Conflict analysis and branching heuristics in the search for graph automorphisms. In *2013 IEEE 25th International Conference on Tools with Artificial Intelligence*, pages 907–914. IEEE, 2013.

[39] Brendan D McKay et al. *Practical graph isomorphism.* Department of Computer Science, Vanderbilt University Tennessee, USA, 1981.

[40] Stephen G Hartke and AJ Radcliffe. Mckays canonical graph labeling algorithm. *Communicating mathematics*, 479:99–111, 2009.

[41] Takunari Miyazaki. The complexity of mckay's canonical labeling algorithm. In *Groups and Computation, Proceedings of a DIMACS Workshop, New Brunswick, New Jersey, USA, June 7-10, 1995*, 1995.

[42] Anuj Dawar and Kashif Khan. Constructing hard examples for graph isomorphism. *arXiv preprint arXiv:1809.08154*, 2018.

[43] Daniel Neuen and Pascal Schweitzer. Benchmark graphs for practical graph isomorphism. *arXiv preprint arXiv:1705.03686*, 2017.

[44] Daniel Neuen and Pascal Schweitzer. An exponential lower bound for individualization-refinement algorithms for graph isomorphism. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pages 138–150. ACM, 2018.

[45] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning.* MIT press, 2016.

[46] Alexander Schrijver. *Theory of linear and integer programming.* John Wiley & Sons, 1998.

[47] Richard S Sutton and Andrew G Barto. Reinforcement learning: An introduction. 2011.

[48] HongYun Cai, Vincent W. Zheng, and Kevin Chen-Chuan Chang. A comprehensive survey of graph embedding: Problems, techniques and applications. *CoRR*, abs/1709.07604, 2017. URL http://arxiv.org/abs/1709.07604.

[49] Shaosheng Cao, Wei Lu, and Qiongkai Xu. Deep neural networks for learning graph representations. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA.*, pages 1145–1152, 2016.

[50] Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodolà, Jan Svoboda, and Michael M. Bronstein. Geometric deep learning on graphs and manifolds using mixture model cnns. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 5425–5434, 2017.

[51] Bernhard Schölkopf and Alexander Johannes Smola. *Learning with Kernels: support vector machines, regularization, optimization, and beyond.* Adaptive computation and machine learning series. MIT Press, 2002.

[52] Thomas Gärtner, Peter A. Flach, and Stefan Wrobel. On graph kernels: Hardness results and efficient alternatives. In *Computational Learning Theory and Kernel Machines, 16th Annual Conference on Computational Learning Theory and 7th Kernel Workshop, COLT/Kernel 2003, Washington, DC, USA, August 24-27, 2003, Proceedings*, pages 129–143, 2003.

[53] László Babai. Graph isomorphism in quasipolynomial time [extended abstract]. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 684–697, 2016.

[54] S. V. N. Vishwanathan, Nicol N. Schraudolph, Risi Kondor, and Karsten M. Borgwardt. Graph kernels. *J. Mach. Learn. Res.*, 11:1201–1242, August 2010. ISSN 1532-4435.

[55] David Haussler. Convolution kernels on discrete structures. Technical report, 1999.

[56] Pinar Yanardag and S. V. N. Vishwanathan. Deep graph kernels. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Sydney, NSW, Australia, August 10-13, 2015*, pages 1365–1374, 2015.

[57] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. Learning convolutional neural networks for graphs. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, pages 2014–2023, 2016.

[58] Antoine Jean-Pierre Tixier, Giannis Nikolentzos, Polykarpos Meladianos, and Michalis Vazirgiannis. Classifying graphs as images with convolutional neural networks. *CoRR*, abs/1708.02218, 2017. URL http://arxiv.org/abs/1708.02218.

[59] Silvio Micali and Zeyuan Allen Zhu. Reconstructing markov processes from independent and anonymous experiments. *Discrete Applied Mathematics*, 200:108–122, 2016.

[60] Quoc V. Le and Tomas Mikolov. Distributed representations of sentences and documents. In *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014*, pages 1188–1196, 2014.

[61] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. A neural probabilistic language model. *Journal of Machine Learning Research*, 3:1137–1155, 2003.

[62] Aditya Grover and Jure Leskovec. Node2vec: Scalable feature learning for networks. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 855–864, New York, NY, USA, 2016. ACM. doi: 10.1145/2939672.2939754.

[63] Annamalai Narayanan, Mahinthan Chandramohan, Lihui Chen, Yang Liu, and Santhoshkumar Saminathan. subgraph2vec: Learning distributed representations of rooted sub-graphs from large graphs. *CoRR*, abs/1606.08928, 2016.

[64] Junchi Yan, Xu-Cheng Yin, Weiyao Lin, Cheng Deng, Hongyuan Zha, and Xiaokang Yang. A short survey of recent advances in graph matching. In *Proceedings of the 2016 ACM on International Conference on Multimedia Retrieval*, pages 167–174. ACM, 2016.

[65] James Mercer. Xvi. functions of positive and negative type, and their connection the theory of integral equations. *Philosophical transactions of the royal society of London. Series A, containing papers of a mathematical or physical character*, 209 (441-458):415–446, 1909.

[66] Bernhard Schölkopf, Alexander J Smola, Francis Bach, et al. *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press, 2002.

[67] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[68] S. V. N. Vishwanathan, Karsten M. Borgwardt, and Nicol N. Schraudolph. Fast computation of graph kernels. In *Proceedings of the 19th International Conference*

*on Neural Information Processing Systems*, NIPS'06, pages 1449–1456, Cambridge, MA, USA, 2006. MIT Press.

[69] Brendan D McKay et al. *Practical graph isomorphism*. Department of Computer Science, Vanderbilt University Tennessee, USA, 1981.

[70] Brendan D McKay and Adolfo Piperno. Practical graph isomorphism, ii. *Journal of Symbolic Computation*, 60:94–112, 2014.

[71] Donald L Kreher and Douglas R Stinson. *Combinatorial algorithms: generation, enumeration, and search*, volume 7. CRC press, 1998.

[72] Harry Wiener. Influence of interatomic forces on paraffin properties. *The Journal of Chemical Physics*, 15(10):766–766, 1947.

[73] Harry Wiener. Correlation of heats of isomerization, and differences in heats of vaporization of isomers, among the paraffin hydrocarbons. *Journal of the American Chemical Society*, 69(11):2636–2638, 1947.

[74] Haruo Hosoya. Topological index. a newly proposed quantity characterizing the topological nature of structural isomers of saturated hydrocarbons. *Bulletin of the Chemical Society of Japan*, 44(9):2332–2339, 1971.

[75] Roberto Todeschini and Viviana Consonni. *Handbook of molecular descriptors*, volume 11. John Wiley & Sons, 2008.

[76] Paul J. Kelly. A congruence theorem for trees. *Pacific J. Math.*, 7(1):961–968, 1957. URL https://projecteuclid.org:443/euclid.pjm/1103043674.

[77] S. V. N. Vishwanathan, Nicol N. Schraudolph, Risi Kondor, and Karsten M. Borgwardt. Graph kernels. *Journal of Machine Learning Research*, 11:1201–1242, 2010.

[78] Hisashi Kashima, Koji Tsuda, and Akihiro Inokuchi. *Kernels for Graphs*. MIT Press, 2004.

[79] SVN Vishwanathan. *Kernel Methods Fast Algorithms and real life applications*. PhD thesis, Indian Institute of Science Bangalore, 2003.

[80] Jin-Yi Cai, Martin Fürer, and Neil Immerman. An optimal lower bound on the number of variables for graph identification. *Combinatorica*, 12(4):389–410, 1992.

[81] Saurabh Verma and Zhi-Li Zhang. Hunt for the unique, stable, sparse and fast feature learning on graphs. In *Advances in Neural Information Processing Systems*, pages 88–98, 2017.

[82] Karsten M Borgwardt and Hans-Peter Kriegel. Shortest-path kernels on graphs. In *Fifth IEEE international conference on data mining (ICDM'05)*, pages 8–pp. IEEE, 2005.

[83] Tamás Horváth, Thomas Gärtner, and Stefan Wrobel. Cyclic pattern kernels for predictive graph mining. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 158–167. ACM, 2004.

[84] Jan Ramon and Thomas Gärtner. Expressivity versus efficiency of graph kernels. In *Proceedings of the first international workshop on mining graphs, trees and sequences*, pages 65–74, 2003.

[85] Pierre Mahé and Jean-Philippe Vert. Graph kernels based on tree patterns for molecules. *Machine learning*, 75(1):3–35, 2009.

[86] Risi Kondor and Karsten M Borgwardt. The skew spectrum of graphs. In *Proceedings of the 25th international conference on Machine learning*, pages 496–503. ACM, 2008.

[87] Risi Kondor, Nino Shervashidze, and Karsten M Borgwardt. The graphlet spectrum. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 529–536. ACM, 2009.

[88] Holger Fröhlich, Jörg K Wegner, Florian Sieker, and Andreas Zell. Optimal assignment kernels for attributed molecular graphs. In *Proceedings of the 22nd international conference on Machine learning*, pages 225–232. ACM, 2005.

[89] Nils M Kriege, Pierre-Louis Giscard, and Richard Wilson. On valid optimal assignment kernels and applications to graph classification. In *Advances in Neural Information Processing Systems*, pages 1623–1631, 2016.

[90] Lu Bai, Peng Ren, Xiao Bai, and Edwin R Hancock. A graph kernel from the depth-based representation. In *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR)*, pages 1–11. Springer, 2014.

[91] Lu Bai, Luca Rossi, Zhihong Zhang, and Edwin Hancock. An aligned subtree kernel for weighted graphs. In *International Conference on Machine Learning*, pages 30–39, 2015.

[92] Nils M Kriege, Fredrik D Johansson, and Christopher Morris. A survey on graph kernels. *arXiv preprint arXiv:1903.11835*, 2019.

[93] Giannis Nikolentzos, Giannis Siglidis, and Michalis Vazirgiannis. Graph kernels: A survey. *arXiv preprint arXiv:1904.12218*, 2019.

[94] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *CoRR*, 2013. URL http://arxiv.org/abs/1301.3781.

[95] Annamalai Narayanan, Mahinthan Chandramohan, Rajasekar Venkatesan, Lihui Chen, Yang Liu, and Shantanu Jaiswal. graph2vec: Learning distributed representations of graphs. In *Proceedings of the 13th International Workshop on Mining and Learning with Graphs (MLG)*, 2017.

[96] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1263–1272. JMLR. org, 2017.

[97] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203*, 2013.

[98] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in neural information processing systems*, pages 3844–3852, 2016.

[99] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[100] Ruoyu Li, Sheng Wang, Feiyun Zhu, and Junzhou Huang. Adaptive graph convolutional neural networks. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[101] Chenyi Zhuang and Qiang Ma. Dual graph convolutional networks for graph-based semi-supervised classification. In *Proceedings of the 2018 World Wide Web Conference*, pages 499–508. International World Wide Web Conferences Steering Committee, 2018.

[102] Alessio Micheli. Neural network for graphs: A contextual constructive approach. *IEEE Transactions on Neural Networks*, 20(3):498–511, 2009.

[103] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018.

[104] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[105] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[106] Oriol Vinyals, Samy Bengio, and Manjunath Kudlur. Order matters: Sequence to sequence for sets. *arXiv preprint arXiv:1511.06391*, 2015.

[107] Thomas N Kipf and Max Welling. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308*, 2016.

[108] Rafael Gómez-Bombarelli, Jennifer N Wei, David Duvenaud, José Miguel Hernández-Lobato, Benjamín Sánchez-Lengeling, Dennis Sheberla, Jorge Aguilera-Iparraguirre, Timothy D Hirzel, Ryan P Adams, and Alán Aspuru-Guzik. Automatic chemical design using a data-driven continuous representation of molecules. *ACS central science*, 4(2):268–276, 2018.

[109] Martin Simonovsky and Nikos Komodakis. Graphvae: Towards generation of small graphs using variational autoencoders. In *International Conference on Artificial Neural Networks*, pages 412–422. Springer, 2018.

[110] Aleksandar Bojchevski, Oleksandr Shchur, Daniel Zügner, and Stephan Günnemann. Netgan: Generating graphs via random walks. *arXiv preprint arXiv:1803.00816*, 2018.

[111] Hongwei Wang, Jia Wang, Jialin Wang, Miao Zhao, Weinan Zhang, Fuzheng Zhang, Xing Xie, and Minyi Guo. Graphgan: Graph representation learning with generative adversarial nets. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[112] Jiaxuan You, Rex Ying, Xiang Ren, William L Hamilton, and Jure Leskovec. Graphrnn: Generating realistic graphs with deep auto-regressive models. *arXiv preprint arXiv:1802.08773*, 2018.

[113] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. Computational capabilities of graph neural networks. *IEEE Transactions on Neural Networks*, 20(1):81–102, 2008.

[114] Barbara Hammer, Alessio Micheli, and Alessandro Sperduti. Universal approximation capability of cascade correlation for structures. *Neural Computation*, 17 (5):1109–1159, 2005.

[115] Haggai Maron, Heli Ben-Hamu, Nadav Shamir, and Yaron Lipman. Invariant and equivariant graph networks. *arXiv preprint arXiv:1812.09902*, 2018.

[116] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S Yu. A comprehensive survey on graph neural networks. *arXiv preprint arXiv:1901.00596*, 2019.

[117] Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, and Maosong Sun. Graph neural networks: A review of methods and applications. *arXiv preprint arXiv:1812.08434*, 2018.

[118] Ajith Abraham. *Computational Social Networks: Security and Privacy*. Springer Publishing Company, Incorporated, 2012.

[119] Silvio Micali and Zeyuan Allen Zhu. Reconstructing markov processes from independent and anonymous experiments. *Discrete Applied Mathematics*, 2016.

[120] Winston Yang. Bell numbers and k-trees. *Discrete Math.*, 1996.

[121] Alain Hertz and Hadrien Mélot. Counting the number of non-equivalent vertex colorings of a graph. *Discrete Appl. Math.*, 2016.

[122] Richard P. Stanley. *Enumerative Combinatorics: Volume 1*. 2011.

[123] Daniel Berend and Tamir Tassa. Improved bounds on bell numbers and on moments of sums of random variables. *Probability and Math. Statistics*, 2010.

[124] Jack Edmonds and Ellis L. Johnson. Matching, euler tours and the chinese postman. *Math. Program.*, 1973.

[125] H. Tubig. The number of walks and degree powers in directed graphs. Technical report, Computer Science Department, Rutgers University, TUM-I123, TU Munich, 2012.

[126] Tomas Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems NIPS*, pages 3111–3119, 2013.

[127] Michael Gutmann and Aapo Hyvärinen. Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2010, Chia Laguna Resort, Sardinia, Italy, May 13-15, 2010*, pages 297–304, 2010.

[128] Sébastien Jean, KyungHyun Cho, Roland Memisevic, and Yoshua Bengio. On using very large target vocabulary for neural machine translation. In *ACL 2015*, pages 1–10, 2015.

[129] Giannis Nikolentzos, Polykarpos Meladianos, and Michalis Vazirgiannis. Matching node embeddings for graph similarity. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA.*, pages 2429–2435, 2017.

[130] Mahito Sugiyama and Karsten M. Borgwardt. Halting in random walk kernels. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 1639–1647, 2015.

[131] J Paul Hamilton, Michael C Chen, and Ian H Gotlib. Neural systems approaches to understanding major depressive disorder: an intrinsic functional organization perspective. *Neurobiology of disease*, 52:4–11, 2013.

[132] Michael J Robinson, Sara E Edwards, Smriti Iyengar, Frank Bymaster, Michael Clark, and Wayne Katon. Depression and pain. *Front Biosci*, 14(503):l–5051, 2009.

[133] Gilles Ambresin, Patty Chondros, Christopher Dowrick, Helen Herrman, and Jane M Gunn. Self-rated health and long-term prognosis of depression. *The Annals of Family Medicine*, 12(1):57–65, 2014.

[134] Athanasios Gaitatzis, Kevin Carroll, Azeem Majeed, and Josemir W Sander. The epidemiology of the comorbidity of epilepsy in the general population. *Epilepsia*, 45(12):1613–1622, 2004.

[135] JJ Barry, AB Ettinger, P Friel, FG Gilliam, CL Harden, B Hermann, AM Kanner, R Caplan, S Plioplys, J Salpekar, et al. Advisory group of the epilepsy foundation as part of its mood disorder. consensus statement: the evaluation and treatment of people with epilepsy and affective disorders. *Epilepsy Behav*, 13(Suppl 1):S1–29, 2008.

[136] Dale C Hesdorffer, W Allen Hauser, John F Annegers, and Gregory Cascino. Major depression is a risk factor for seizures in older adults. *Annals of Neurology: Official Journal of the American Neurological Association and the Child Neurology Society*, 47(2):246–249, 2000.

[137] Y.K. Kim and K.S. Na. Application of machine learning classification for structural brain mri in mood disorders: Critical review from a clinical perspective. *Prog. Neuro-Psychopharmacology Biol. Psychiatry*, 80:71–80, 2018.

[138] Cynthia HY Fu and Sergi G Costafreda. Neuroimaging-based biomarkers in psychiatry: clinical opportunities of a paradigm shift. *The Canadian Journal of Psychiatry*, 58(9):499–508, 2013.

[139] Andrew T Drysdale, Logan Grosenick, Jonathan Downar, Katharine Dunlop, Farrokh Mansouri, Yue Meng, Robert N Fetcho, Benjamin Zebley, Desmond J Oathes, Amit Etkin, et al. Resting-state connectivity biomarkers define neurophysiological subtypes of depression. *Nature medicine*, 23(1):28, 2017.

[140] M. Sharaev, A. Andreev, A. Artemov, E. Burnaev, E. Kondratyeva, S. Sushchinskaya, I. Samotaeva, V. Gaskin, and A. Bernstein. Pattern recognition pipeline for neuroimaging data. In Luca Pancioni, Friedhelm Schwenker, and Edmondo Trentin, editors, *Artificial Neural Networks in Pattern Recognition (ANNPR-2018). Lecture Notes in Computer Science*, volume 11081, pages 306–319. Springer, 2018.

[141] M. Sharaev, A. Artemov, E. Kondratyeva, S. Sushchinskaya, E. Burnaev, A. Bernstein, R. Akzhigitov, and A. Andreev. Mri-based diagnostics of depression concomitant with epilepsy: in search of the potential biomarkers. In *2018 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*. IEEE, 2018.

[142] Meenal J Patel, Alexander Khalaf, and Howard J Aizenstein. Studying depression using imaging and machine learning methods. *NeuroImage: Clinical*, 10:115–123, 2016.

[143] Gaël Varoquaux, Alexandre Gramfort, Jean-Baptiste Poline, and Bertrand Thirion. Brain covariance selection: better individual functional connectivity models using population prior. In *Advances in neural information processing systems*, pages 2334–2342, 2010.

[144] Xin Wang, Yanshuang Ren, and Wensheng Zhang. Depression disorder classification of fmri data using sparse low-rank functional brain network and graph-based features. *Computational and mathematical methods in medicine*, 2017, 2017.

[145] Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M. Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12:2539–2561, 2011.

[146] Sergey Ivanov and Evgeny Burnaev. Anonymous walk embeddings. In *Proceedings of the 35th International Conference on Machine Learning*, 2018.

[147] Silvio Micali and Zeyuan Allen Zhu. Reconstructing markov processes from independent and anonymous experiments. *Discrete Applied Mathematics*, 200:108–122, 2016.

[148] Jonathan D Cohen, Nathaniel Daw, Barbara Engelhardt, Uri Hasson, Kai Li, Yael Niv, Kenneth A Norman, Jonathan Pillow, Peter J Ramadge, Nicholas B Turk-Browne, et al. Computational approaches to fmri analysis. *Nature neuroscience*, 20(3):304, 2017.

[149] D. Smolyakov and E. Burnaev. One-class SVM with privileged information and its application to malware detection. In Carlotta Domeniconi, Francesco Gullo, Francesco Bonchi, Josep Domingo-Ferrer, Ricardo A. Baeza-Yates, Zhi-Hua Zhou, and Xindong Wu, editors, *IEEE International Conference on Data Mining Workshops, ICDM Workshops 2016, December 12-15, 2016, Barcelona, Spain.*, pages 273–280. IEEE Computer Society, 2016. doi: 10.1109/ICDMW.2016.0046. URL https://doi.org/10.1109/ICDMW.2016.0046.

[150] D. Smolyakov, P. Erofeev, and E. Burnaev. Model selection for anomaly detection. In A. Verikas, P. Radeva, and D. Nikolaev, editors, *Proc. SPIE 9875, Eighth International Conference on Machine Vision, Barcelona, Spain (December 8, 2015)*, volume 9875. SPIE, 2015.

[151] A. Papanov, P. Erofeev, and E. Burnaev. Influence of resampling on accuracy of imbalanced classification. In A. Verikas, P. Radeva, and D. Nikolaev, editors, *Proc. SPIE 9875, Eighth International Conference on Machine Vision, Barcelona, Spain (December 8, 2015)*, volume 9875. SPIE, 2015.

[152] Daniel Yekutieli and Yoav Benjamini. Resampling-based false discovery rate controlling multiple test procedures for correlated test statistics. *Journal of Statistical Planning and Inference*, 82(1-2):171–196, 1999.

[153] S.S. Chernova and E.V. Burnaev. On an iterative algorithm for calculating weighted principal components. *Journal of Communications Technology and Electronics*, 60(6):619–624, Jun 2015.

[154] Benson Mwangi, Tian Siva Tian, and Jair C. Soares. A review of feature reduction techniques in neuroimaging. *Neuroinformatics*, 12(2):229–244, 2014.

[155] Antti Airola, Tapio Pahikkala, Willem Waegeman, Bernard De Baets, and Tapio Salakoski. A comparison of auc estimators in small-sample studies. In *Machine learning in systems biology*, pages 3–13, 2009.

[156] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.

[157] David Kempe, Jon M. Kleinberg, and Éva Tardos. Maximizing the spread of influence through a social network. In *KDD*, pages 137–146, 2003.

[158] L. de Vries, S. Gensler, and Peter S.H. Leeflang. Popularity of brand posts on brand fan pages: An investigation of the effects of social media marketing. *Journal of Interactive Marketing*, 26(2):83–91, 2012.

[159] I. P. Cvijikj and F. Michahelles. Online engagement factors on facebook brand pages. *Social Network Analysis and Mining*, 3(4):843–861, 2013.

[160] E. Katz. Mass communications research and the study of popular culture: An editorial note on a possible future of this journal. *Studies in Public Communication*, 2:1–6, 1959.

[161] S. Aral and D. Walker. Creating social contagion through viral product design: A randomized trial of peer influence in networks. *Management Science*, 57(9): 1623–1639, 2011.

[162] N. Barbieri and F. Bonchi. Influence maximization with viral product design. In *SDM*, 2014.

[163] N. Barbieri, F. Bonchi, and G. Manco. Topic-aware social influence propagation models. In *ICDM*, 2012.

[164] W. Chen, C. Wang, and Y. Wang. Scalable influence maximization for prevalent viral marketing in large-scale social networks. In *KDD*, 2010.

[165] Wei Chen, Laks V. S. Lakshmanan, and Carlos Castillo. *Information and Influence Propagation in Social Networks*. Morgan & Claypool Publishers, 2013. ISBN 1627051155, 9781627051156.

[166] J. Leskovec, A. Krause, C. Guestrin, C. Faloutsos, J. M. VanBriesen, and N. S. Glance. Cost-effective outbreak detection in networks. In *KDD*, 2007.

[167] Brian Karrer and M. E. J. Newman. Stochastic blockmodels and community structure in networks. *Phys. Rev. E*, 83:016107, Jan 2011.

[168] Tiago P. Peixoto. Model selection and hypothesis testing for large-scale network models with overlapping groups. *Phys. Rev. X*, 5, 2015.

[169] Jure Leskovec, Andreas Krause, Carlos Guestrin, Christos Faloutsos, Jeanne VanBriesen, and Natalie Glance. Cost-effective outbreak detection in networks.

[170] David Kempe, Jon Kleinberg, and Éva Tardos. Maximizing the spread of influence through a social network. SIGKDD 2003.

[171] Sergey Ivanov and Evgeny Burnaev. Anonymous walk embeddings. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, 2018.

[172] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. ICLR '17.

[173] Maximillian Nickel and Douwe Kiela. Learning continuous hierarchies in the lorentz model of hyperbolic geometry. In *ICML '18*.

[174] Sergey Ivanov and Evgeny Burnaev. Anonymous walk embeddings. ICML '18.

[175] Akhil Arora, Sainyam Galhotra, and Sayan Ranu. Debunking the myths of influence maximization: An in-depth benchmarking study. SIGMOD '17.

[176] P. Domingos and M. Richardson. Mining the network value of customers. In *KDD*, 2001.

[177] M. Richardson and P. Domingos. Mining knowledge-sharing sites for viral marketing. In *KDD*, 2002.

[178] W. Chen, Y. Yuan, and L. Zhang. Scalable influence maximization in social networks under the linear threshold model. In *ICDM*, 2010.

[179] S. Cheng, H. Shen, J. Huang, W. Chen, and X. Cheng. IMRank: influence maximization via finding self-consistent ranking. In *SIGIR*, 2014.

[180] A. Goyal, F. Bonchi, and L. V. S. Lakshmanan. A data-based approach to social influence maximization. *PVLDB*, 5(1):73–84, 2011.

[181] Y. -C. Chen, W. -C. Peng, and S. -Y. Lee. Efficient algorithms for influence maximization in social networks. *Knowl. Inf. Syst.*, 33(3):577–601, 2012.

[182] C. Borgs, M. Brautbar, J. T. Chayes, and B. Lucier. Maximizing social influence in nearly optimal time. In *SODA*, 2014.

[183] Y. Tang, X. Xiao, and Y. Shi. Influence maximization: near-optimal time complexity meets practical efficiency. In *SIGMOD*, 2014.

[184] C. Aslay, N. Barbieri, F. Bonchi, and R. A. Baeza-Yates. Online topic-aware influence maximization queries. In *EDBT*, 2014.

[185] W. Chen, T. Lin, and C. Yang. Efficient topic-aware influence maximization using preprocessing. *CoRR*, abs/1403.0057, 2014.

[186] S. Chen, J. Fan, G. Li, J. Feng, K.-L. Tan, and J. Tang. Online topic-aware influence maximization. *Proc. VLDB Endow.*, 8(6):666–677, 2015.

[187] Y. Li, D. Zhang, and K.-L. Tan. Real-time targeted influence maximization for online advertisements. *Proc. VLDB Endow.*, 8(10):1070–1081, 2015.

[188] L. Hong and B. D. Davison. Empirical study of topic modeling in twitter. In *SOMA*, 2010.

[189] Wei Chen, Yajun Wang, and Siyu Yang. Efficient influence maximization in social networks. In *KDD*, pages 199–208, 2009.

[190] Wei Chen, Chi Wang, and Yajun Wang. Scalable influence maximization for prevalent viral marketing in large-scale social networks. In *KDD*, pages 1029–1038, 2010.

[191] Annamalai Narayanan, Mahinthan Chandramohan, Rajasekar Venkatesan, Lihui Chen, and Yang Liu. graph2vec: Learning distributed representations of graphs.

[192] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. KDD '14.

[193] Hung T. Nguyen, My T. Thai, and Thang N. Dinh. Stop-and-stare: Optimal sampling algorithms for viral marketing in billion-scale networks. SIGMOD '16.