Subgraph Matching and Mining in Large Graphs

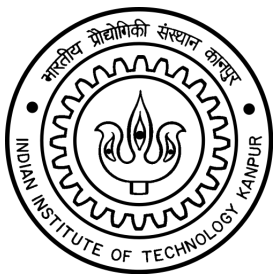
A thesis submitted
in Partial Fulfillment of the Requirements
for the Degree of

Doctor of Philosophy

by

Shubhangi Agarwal

to the



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY KANPUR
April, 2023

POST GRADUATE OFFICE
THESIS SUBMITTED
ON 18/04/23
I. I. T., KANPUR - 16

CERTIFICATE

It is certified that the work contained in the thesis titled **Subgraph Matching and Mining in Large Graphs**, by **Shubhangi Agarwal**, has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

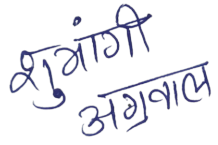
Arnab Bhattacharya

Prof. Arnab Bhattacharya
Department of Computer Science and Engineering
IIT Kanpur

April, 2023

DECLARATION

This is to certify that the thesis titled **Subgraph Matching and Mining in Large Graphs** has been authored by me. It presents the research conducted by me under the supervision of **Prof. Arnab Bhattacharya**. To the best of my knowledge, it is an original work, both in terms of research content and narrative, and has not been submitted elsewhere, in part or in full, for a degree. Further, due credit has been attributed to the relevant state-of-the-art and collaborations (if any) with appropriate citations and acknowledgments, in line with established norms and practices.



Signature

Name: Shubhangi Agarwal

Programme: PhD

Department: Computer Science and Engineering

Indian Institute of Technology Kanpur

Kanpur 208016

ABSTRACT

Name of student: **Shubhangi Agarwal** Roll no: **14111268**

Degree for which submitted: **Doctor of Philosophy**

Department: **Computer Science and Engineering**

Thesis title: **Subgraph Matching and Mining in Large Graphs**

Name of Thesis Supervisor: **Prof. Arnab Bhattacharya**

Month and year of thesis submission: **April, 2023**

The rapid progress in internet connectivity and the continual evolution of data storage and sharing techniques have substantially widened access to vast amounts of information. This surge in data availability is especially evident in the proliferation of linked data across diverse domains such as social networks, chemoinformatics, bioinformatics, and road networks. This expanding landscape underscores the imperative of harnessing the power of *graphs* as an instrumental means towards enhanced comprehension and deeper insights. Graphs provide a strong foundation for analyzing interconnected networks and can be automatically created by extracting entities and relationships. A labeled graph is a way of representing data where entities are modeled by nodes with associated labels, and their relationships are depicted by edges or links between them.

Graph mining is an important field of research for efficiently analyzing large real-world graphs. To extract useful information from graphs or networks various data mining techniques are applied to graphs, like identifying clusters and mining frequent patterns. A popular subfield of graph mining, *Subgraph querying*, focuses on identifying all occurrences of a specific subgraph pattern in a larger graph or a set of graphs. The goal is to identify specific subgraphs of interest, which can help

in answering specific questions about the graph or verifying the presence of a known structure. For instance, the presence of a substructure in a chemical compound can help us understand its properties and behavior.

The applications of subgraph querying across a wide range of domains and over large amounts of data encourage the development of efficient approaches for label and structural pattern matching on large graphs. As subgraph isomorphism is NP-complete, efficient heuristics have been proposed. However, in certain cases the graph may have missing labels or edges, making it unsuitable to apply exact graph querying paradigms. Technological advancements have made possible the automatic construction of graphs from raw data. However, such graph constructing models assign a *probability value* or confidence score to the extracted entities or relationships or their attributes. In such a scenario, an exact subgraph match may not exist and an *approximate subgraph match* may be more suitable.

Approximate subgraph matching (ASM) involves finding subgraph patterns in a larger input graph that are *similar* to a given subgraph pattern, generally referred to as a query, but not necessarily identical. This provides more flexibility and robustness in graph analysis and enables the detection of subgraphs that may have different labels, sizes, or shapes. Applications of ASM range from recommendation systems to medical diagnostics based on symptom-disease association.

Various similarity measures can be used to compute the degree of similarity between the query graph pattern and subgraphs in the input graph, such as graph edit distance, maximum common subgraph or graph based statistical measures like degree distribution. However, methods based on such similarity measures commonly define application dependent thresholds. Moreover, if the input graph is probabilistic, it may be necessary to establish an acceptable probability threshold for existence. Determining an appropriate threshold is necessary to capture important patterns, however, this process can be challenging, especially in cases where the underlying distribution of data is unknown or the data is noisy.

In this work, we address this issue and propose *statistical significance* based sub-

graph querying methods for both deterministic and probabilistic graphs. *Statiscal significance* measures like the *Pearson's chi-squared statistic* allows to integrate the label and the topological similarity of a subgraph as well as the associated probabilities. Another widespread way is the use of neural network-based models to map the input subgraphs and the query to a feature space and compare the embedded vectors to search for answer retrieval. As the embeddings of the graphs are often obtained by aggregating the embeddings of the constituent nodes, we present a *graph neural network* for generating node embedding vectors that capture *global position* of the nodes in the context of the graph with respect to a chosen set of nodes, called *anchors*.

We propose two novel chi-squared based approaches, VerSaChI and ChiSeL, to search for approximate subgraph patterns in deterministic and probabilistic input graphs, respectively. VerSaChI computes a similarity of the neighborhood until two-hops and uses Chebyshev's inequality along with χ^2 measure. On the other hand, ChiSeL takes into account the possible world semantics while avoiding enumeration of all possible instantiations of the (sub)graph. We also conduct a study using VeNoM, a variant of an existing ASM approach, by parametrizing the depth and breadth of the neighborhood considered. We also discuss GraphReach, a random walk based *position-aware graph neural network*. It uses a diversified anchor selection algorithm for a more meaningful node embedding. Extensive experiments demonstrate the robustness of various methods and showcase their efficacy on different datasets.

Dedicated to

Late Dr. Bharati Agarwal, my loving father

and

Smt. Renu Agarwal, my strength.

I hope I make you proud.

Acknowledgements

I would like to express my sincere gratitude to all those who have contributed to the completion of this thesis.

First and foremost, I would like to thank my supervisor, Prof. Arnab Bhattacharya, for his expert guidance and unwavering support. Throughout the course of my research, I faced several personal challenges which slowed down my progress. However, during these difficult times, he was patient and understanding and provided the necessary flexibility and support to help me stay on track, for which I will be eternally grateful. I am also grateful for the mentorship and guidance of Prof. Arnab, which not only helped shape this thesis but also contributed to my personal growth and development.

I would also like to thank my collaborators, Prof. Sayan Ranu, Sourav Dutta and Sunil Nishad. Prof. Sayan was really enthusiastic and his expertise is invaluable to this research. I am also deeply grateful to Sourav Dutta, with whom I collaborated closely. He provided me with guidance and support like an elder brother, always looking out for me and patiently guiding me whenever I faced challenges. Our meetings, which involved insightful discussions and moments of fun, were a delightful blend of academia and rapport. I am incredibly thankful for his support and mentorship throughout this process.

I feel incredibly blessed to have a wonderful group of friends who have supported me throughout my academic journey. Rishabh, Aanandita, Rini, Rutika, Priyam, Ankur and Abhishek have kept me sane since the beginning of my academic progress while matching my craziness at the same time. I am also grateful to Hrishikesh,

Keerti, Rujuta and Awanish, who became my second family and have provided unwavering support in both my personal and professional life. A special thanks to Hrishikesh, for always having my back despite my flaws. This journey wouldn't have been possible without him. I am also thankful to Garima, Abhishek and Umair, for helping me unwind and encouraging me to participate in both indoor and outdoor sports, and helping me grow in dimensions other than academia. A special mention to Shubham, Arpita and Monika for their unadulterated opinions and bits of advice. I would also like to express my gratitude to Susmitha and Pralay for helping me retain my inner child. Their infectious enthusiasm and positive attitude helped to maintain a happy and positive environment.

Finally, I would like to thank my family who stuck by me through this long journey. I am deeply indebted to my mother Smt. Renu Agarwal. Thank you Mom for always keeping me close to my roots. I am thankful to my father (Late) Dr. Bharati Agarwal, for instilling basic human values in me, and for being my inspiration to not give up on this journey and follow in his footsteps. Papa, your child-like curiosity and excitement filled me with happiness and there is not a day I don't miss you. I would also like to thank my elder brother, Shri Asheesh Agarwal for looking out for me in his own way and silently protecting me the best he can. A loving mention to my sister-in-law Smt. Vintee Agarwal for being a sister I never had and to my chipmunks, Akshita, thank you for keeping me sane and entertained during lockdown and for your unconditional love and Ram, for instantly recharging me with your infectious smile.

Just like it takes a village to raise a child, it takes a village to raise a Ph.D. I would like to thank each and everyone, people mentioned above and beyond, for the part you played in this journey.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

There has been a significant increase in linked open data, social communities, and interconnected networks on the World Wide Web. The data available is huge and heterogeneous. For better understanding and analysis, it is best to visualize and represent such large data using a *labeled graph*, where entities are modeled by *nodes* with associated *labels* and their relationships are depicted by *edges* or *links* between them. Graphs form a strong foundation for various domains like social networks, protein and chemical interaction data, bioinformatics, route planning, etc. With the advancement in learning techniques, graphs can be created automatically by extracting entities and relationships, YAGO2 (Suchanek et al., 2007) and DBpedia (Auer et al., 2007) are some examples of automatically created knowledge graphs.

Over the years, researchers have developed various querying and mining techniques to study patterns and properties of graph data. This area of research is popularly known as *graph mining*. A subfield of graph mining is *subgraph querying*, also known as *subgraph matching*. It focuses on identifying all occurrences of a specific subgraph pattern in a larger graph or a set of graphs. A subgraph is a smaller graph that can be found as a connected substructure in a larger graph. The goal of subgraph querying is to identify specific subgraphs of interest, which can help in answering specific questions about the graph or verifying the presence of a known structure. Subgraph querying is useful in various domains, including social network analysis, bioinformatics, chemical informatics, question-answering and image pro-

cessing. For example, in social network analysis, subgraph querying can be used to find all instances of a specific network motif or structure, such as a triangle or a clique, which can help in identifying important features of the network. In bioinformatics, subgraph querying can be used to search for specific patterns in protein or gene interaction networks, which can help in understanding the biological functions of these molecules.

The applications of subgraph querying across a wide range of domains and over large amounts of data encourage the development of efficient approaches for label and structural pattern matching on large graphs (Aggarwal and Wang, 2010). A widely studied approach is the extraction of a deterministic query graph from a large graph with which it shares structural similarity. The label space is shared as well. Conventionally, a query graph is a set of related entities and relationship-based topology where entities, optionally, may have attributes associated. Traditional querying approaches scan the graph data for an exact match to this query pattern graph and are based on graph isomorphism. However, graph isomorphism has been shown to be quasi-polynomial (Babai, 2016), and subgraph isomorphism is known to be NP-complete (Cook, 1971). Given the infeasibility of the exact label and structural matching models, researchers have explored heuristics for efficient subgraph querying Jüttner and Madarasi (2018); Chen et al. (2018); Han et al. (2019).

Further, many times the data obtained from various sources may have missing labels or edges. This can be due to various reasons, multiple data-source merging (Lian et al., 2016), privacy-preserving perturbation (Boldi et al., 2012) or merely due to human judgment error and bias. Among other reasons, automatically created graphs can be inherently *uncertain* as knowledge graph construction models generally assign a confidence score to the extracted entities and relationships. One of the ways to capture this uncertainty in data is to model it through *probabilistic graphs*, where a value is assigned to entities, edges or attributes to quantify their probability of existence. The noisy and non-deterministic nature of real-world graphs makes it unsuitable to apply exact graph querying paradigms. Instead, *approximate sub-*

graph matching techniques are necessary to handle queries in a robust and real-time manner.

1.1 Approximate Subgraph Matching

Approximate subgraph matching or *ASM* is a variant of subgraph matching that involves finding subgraphs in a larger input graph that are similar to a given subgraph pattern, but not necessarily identical. ASM is often used in situations where an exact match of a subgraph pattern may not be available and/or a similar subgraph is still of interest. Various similarity measures can be used to compute the degree of similarity between the subgraph pattern and subgraphs in the input graph, for example, graph edit distance, graph kernels, graph-based similarity measures or statistical significance. These algorithms compare the topology of the subgraph pattern and the larger graph and return a similarity score that reflects the degree of similarity between them. More generally, they employ the notion of a vertex pair, which consists of a node from the query graph and a candidate vertex from the larger input graph. The similarity scores of the vertex pairs capture the overlap between the neighborhood of the vertices and are then combined to compute the match score of the subgraph.

By allowing for the detection of subgraphs that are similar in topology, but may have different labels, sizes, or shapes, ASM algorithms provide more flexibility and robustness in graph analysis. For example, in bioinformatics, it enables the detection of candidate regions in the genome, that might have undergone abnormal mutations, for studying the associated medical effects (Shen and Guda, 2014; Vandin et al., 2011). In image processing, ASM can be used to detect objects or regions in images that are similar to a given template but may have different scales, orientations, or deformations. This has applications in various areas such as object recognition, face detection, and medical image analysis (Sun et al., 2020). Both traditional (Yan et al., 2016; Dinari, 2017) and machine learning based approximate subgraph extraction methods have been extensively researched for various domains (Anwar et al., 2022;

Lou et al., 2020), although no one method is provably the best. Additionally, with the advancement in connectivity and technology, the amount of data to be processed is becoming larger by the day with graphs having millions of nodes and billions of edges. Thus, there is a compelling need to develop new methods to find matching subgraphs which are more efficient, with better runtime and accuracy.

Approximate subgraph matching approaches can be based on the similarity of the edge structure of the graph, e.g., the graph edit distance or the similarity of the labels, e.g., Jaccard similarity, string edit distance etc. However, such methods generally have a threshold associated with each metric, where one may lose out on good matches with high structural similarity if they exhibit label similarity which is marginally below the specified threshold, or vice-versa. Additionally, if a probability value is associated with the elements of the graph, another application-dependent threshold may need to be defined and appropriately tuned to retrieve matching answer subgraphs.

The use of *statistical significance* measures in graph similarity provides an elegant contrast to threshold-based methods by allowing the unification of label and structural overlap measures. The probabilistic nature of graphs can also be captured using these measures, allowing for retrieval of subgraphs that show *statitically significant* similarity to the queried graph pattern, and that considerably deviate from the expected subgraph pattern. Statistical models and measures involve establishing a relation between the empirical (or observed) results of an experiment with factors affecting the system or to pure chance. In such scenarios, an observation is deemed to be statistically significant if its presence cannot be attributed to randomness alone. The classical *p-value* computes the chance of rejecting the null hypothesis. This implies that the observation is drawn from a known probability model characterizing the experimental setup. In other words,a lesser p-value (typically less than 0.05) is considered statistically significant suggesting that the null hypothesis should be rejected. However, the computation of the p-value is generally computationally infeasible since it entails the generation of all possible outcomes.

To alleviate this problem various methods have been studied (Bejerano et al., 2004), however, in systems, a small factor of error can be tolerated, and an approximation of the p-value can be calculated using other statistical measures. The literature hosts numerous statistical models to capture the uniqueness of observations, including *z-score*, *log-likelihood ratio* (G^2), and *Hotelling's* T^2 *measure* (Read and Cressie, 1988).

In a recent work, VELSET (Dutta et al., 2017) proposed the use of the *Pearson's chi-square* distribution for subgraph matching. The chi-square distribution (χ^2) is widely used to compute the goodness-of-fit of a set of observations (to the theoretical model describing the null hypothesis). In most situations, the chi-square distribution provides a good approximation to the p-value (Read and Cressie, 1989). Inspired by the success of statistical analysis we developed two algorithms for subgraph matching based on the chi-square measure. One of the methods discussed in this work assumes deterministic input graphs and makes use of *Chebyshev's inequality* along with *Pearson's chi-squared test*. The second method is well-suited for probabilistic graphs and employs the chi-squared analysis as well. It avoids enumerating all possible worlds, this clever trick results in the efficient retrieval of statistically significant subgraph matches in large probabilistic graphs while eliminating the need for a threshold based measure. In another study, conducted as a part of this thesis, we identify the limitations of a statistical significance based ASM algorithm and propose a variation improving over it. We also explore the effect of different graph and method parameters on its performance.

The approximate subgraph matching techniques discussed so far were based on graph theory and are more popularly referred to as traditional subgraph matching tactics. Another type of subgraph matching approach is the graph embedding method, which maps each graph into a low-dimensional space, where the similarity between graphs can be measured by distance metrics (Bai et al., 2019). Aggregating over node embeddings is a common way of generating graph embeddings, providing flexibility in assigning importance to more significant (e.g., higher degree) nodes.

Several machine learning (Perozzi et al., 2014; Grover and Leskovec, 2016) and neural network (Kipf and Welling, 2017; Velickovic et al., 2018; Xu et al., 2019) based architectures exist in the literature for generating node embedding vectors. Such neural networks are referred to as *graph neural networks (GNN)s*. GNNs are a type of neural network architecture specifically designed to work with graph-structured data. Unlike traditional neural networks, the underlying architecture of a GNN depends on the input graph and varies from graph to graph. GNNs learn a function that maps the input graph to a desired output. They can generalize to unseen nodes, i.e., nodes that were not present during the training phase. This *inductive* learning capability of GNNs is one of the key reasons behind their popularity. GNNs have shown promise in a variety of tasks, including node classification, link prediction, and graph classification.

However, most GNNs consider only the neighborhood of nodes, due to which nodes with isomorphic neighborhoods have the same embedding and cannot be distinguished from one another. To alleviate this issue, the *position* of the node in the larger context of the graph can be encoded into the embedding. The position of a node is generally captured with respect to a chosen set of *anchor nodes*. P-GNN (You et al., 2019) is the first work to address this need and uses the shortest path distance to the anchor nodes to encode the positional information of graph nodes. Having said that, the assumption of shortest path rarely holds in the real world (Gulyás et al., 2018; West and Leskovec, 2012; London, 2017; TransStat, 2016). Additionally, relying on the shortest paths alone ignores the other paths in the graph, which results in information loss while making the model vulnerable. To mitigate this we propose a random walk based distance metric with a diversified anchor node selection mechanism. The model is robust to adversarial attacks and can be used for downstream prediction tasks.

1.2 Contributions

In this work, we focus on the problem of approximate subgraph matching. We make the following contributions in this regard.

- **VerSaChI**[1] (**V**ertex N**e**ighborhood Agg**r**egation for **S**tatistically Signific**a**nt Subgraphs via **Ch**ebyshev **I**nequality)

  It is a scalable and highly accurate paradigm for approximate subgraph querying over deterministic graphs based on 2-hop label and structural similarity with the query graph.

- **ChiSeL**[2] (**Chi**-Square based **Se**arch in Large Probabi**L**istic Graphs)

  It is an ASM paradigm for probabilistic graphs which can search over graphs of billions of edges in a matter of seconds.

- **VeNoM**[3] (**Ve**rtex **N**eighb**o**r **M**atching)

  It improves over an existing ASM algorithm and discusses various factors that affect the matching approach and analyzes their effect on its performance.

- **GraphReach**[4]

  It is a machine learning based model to produce holistic and robust position-aware node embeddings which can be used in downstream tasks for (sub)graph similarity.

1.2.1 VerSaChI

The problem of approximate subgraph matching in a large graph for a smaller query graph, entails search and retrieval of top-k subgraphs of the large graph that *best* match the query graph. The best match is defined as the retrieved answer subgraph that exhibits the *maximum similarity* with the query graph. For a deterministic,

[1]The source code is available at `https://github.com/shubhangiat/VerSaChI`.
[2]The source code is available at `https://github.com/shubhangiat/ChiSeL`.
[3]The source code is available at `https://github.com/shubhangiat/VeNoM`.
[4]The source code is available at `https://github.com/idea-iitd/GraphReach`.

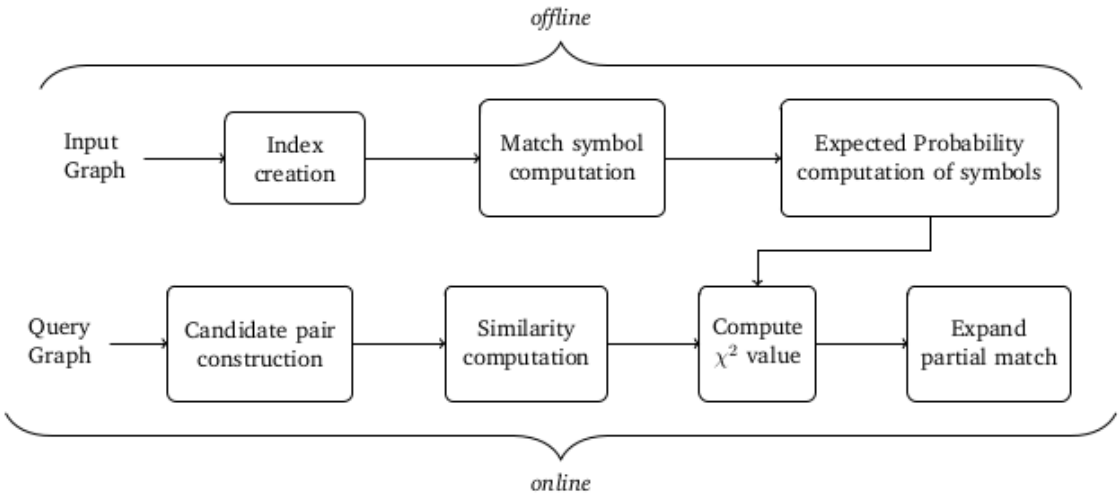


Figure 1.1: VerSaChI - Workflow

undirected and vertex-labeled input graph, with no hyperedges, VerSaChI searches for top-k subgraphs which match with the query graph with high similarity. To estimate similarity, it employs the notion of *statistical significance*. The statistical significance score encodes the degree of similarity of a subgraph of the input graph to the query graph and is computed in a bottom-up manner. The score represents the likelihood that the match is not a random event. To quantify the statistical significance score of a match, the *Pearsons's chi-square statistic (χ^2)* is used which evaluates the deviation of an observed event from the expected event.

To capture its underlying distribution of the input graph, VerSaChI computes the similarity scores between all the vertices of the input graph and creates similarity buckets based on it. It then uses *Chebyshev's inequality* aptly to compute the expected probability of these buckets. The *Chebyshev's inequality* (de Tchébychef, 1867) models the probability of deviation for a random variable from the mean. Such techniques have been studied for sequence mining (Dutta and Bhattacharya, 2010; Sachan and Bhattacharya, 2012; Dutta and Bhattacharya, 2012), substring matching (Dutta, 2015), subgraph similarity (Dutta et al., 2017; Agarwal et al., 2020), and clique finding (Dutta and Lauri, 2019). The similarity between a query node and input vertex is computed based on the label overlap between the immediate and 2-hop neighbors of the query node and the input vertex and is discretized into the buckets. This is the observed similarity. The statistical significance of a

vertex pair match, computed based on the expected probabilities of the buckets and the observed similarity exhibited by the neighbor vertex pairs of the vertex pair, depicts whether there is a significant overlap between the 2-hop neighborhood of the query node and the input vertex. Through a greedy approach, the vertex pair is expanded to a matching subgraph by exploring the neighbor vertex pairs having the largest chi-square score. In this way, k non-overlapping best-matching subgraphs are retrieved from the input graph. Figure 1.1 depicts the workflow of the algorithm.

In this thesis, we showcase the efficacy of this method over several datasets in comparison to other existing approaches. We also analyze the effects of various parameters, like graph size, query type etc on its efficiency. We also study the impact of the bucket size and empirically show that smaller-sized buckets capture the finer differences between the graph substructures better.

1.2.2 ChiSeL

Subgraph querying over deterministic graphs is generally more straightforward than in probabilistic graphs. In deterministic graphs, nodes and edges are either present or not, however, in probabilistic graphs, nodes and edges can have probability values associated with them, indicating the likelihood of their existence. An important aspect of probabilistic data is the *possible world semantics*. The aim of the approximate subgraph querying, now, is to find a subgraph that is not only highly *similar* to the query graph but is also the one with the largest existential probability across all the possible worlds. A *possible world* is a deterministic instance of the non-deterministic input graph, created after considering the probabilities associated with its nodes, edges and attributes. Each world, thus, has an associated probability of existence. For this reason, the direct application of subgraph matching approaches designed for deterministic graphs becomes infeasible as the computation of similarity between an input graph substructure and the query graph requires scavenging through exponential possibilities.

Probabilistic graph querying is used regularly in biology. For example, protein-

protein interaction networks have been modeled as stochastic graphs, where statistical significance measures have been used for motif discovery (Jiang et al., 2006). In the domain of information extraction and natural language processing for automated question-answering platforms such as Jeopardy (Chandrasekar, 2014), involves finding the closest matching subgraph with the highest possible probability of truthfulness and existence of facts. Large graphs pose the need to be efficiently queried for the user to be able to effectively extract relevant information. Since a query in general retrieves multiple answers, it is further important to rank them. However, the probabilistic nature of the graph makes it challenging for deterministic approaches to determine which result is better and, hence, methods that work directly with probabilistic graphs are preferable.

With probabilities involved, the choice of the *best* matching subgraph is influenced by two factors, label and structural similarity, and existential probability. Most ASM methods for probabilistic graphs make a decision based on application-specific fine-tuned thresholds, and matching subgraphs must have a similarity and/or existential probability that exceeds this predetermined threshold value. Thus, there is a trade-off between the label and structural similarity and the existential probability.

ChiSeL proposes an integrated measure based on statistical significance that tries to capture the possible world semantics and this trade-off for approximate matching subgraphs. ChiSeL returns the top-k subgraphs similar to the specified deterministic query graph pattern, from an undirected and vertex-labeled input graph which is void of hyperedges. The input graph is also assumed to be edge-probabilistic with independent probabilities, however, the method can be quickly adapted to various scenarios of uncertainty in the graph structure and attributes.

To compute the similarity of a vertex pair, ChiSeL tries to match the label distribution of the query node with that of the input vertex, two labels at a time, with all possible world instances of the subgraph. The two labels may find a complete match, no match or a partial match, in different possible world scenarios, based on

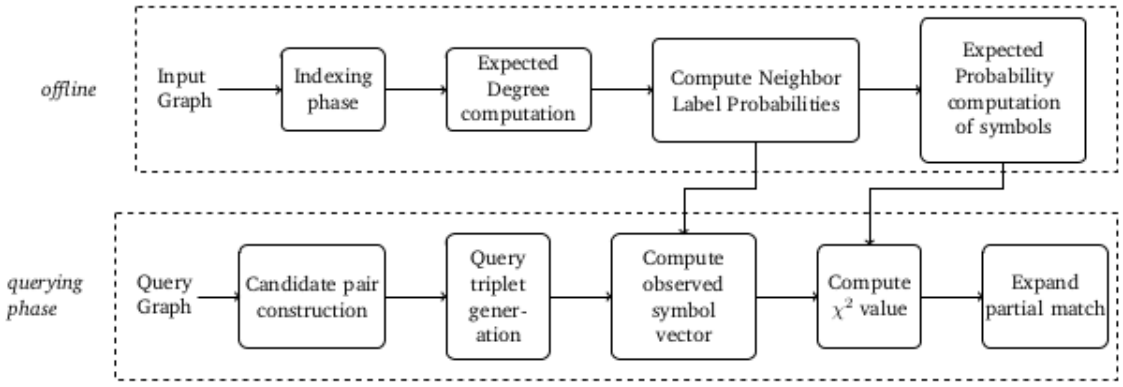


Figure 1.2: Overview of ChiSeL

which each is assigned a probability of match. It is important to observe that there are only these three possible scenarios, and they are exclusive and exhaustive. The summation of probabilities for these three events over all pairs of neighbor-labels of the query node aggregates the amount of label and structural match in different worlds. Although, enumerating all possible worlds is inefficient and computationally expensive. ChiSeL calculates the probabilities of the three match events, for each pair of neighbor-labels of the query node, using insightful observations which overcome the combinatorial nature of the possible world semantics.

The expected computation is based on the number of unique labels in the graph, to incorporate label similarity of neighbors; and, the degree of a vertex, for structural match. However, since the graph is probabilistic, the existence of neighbors of a vertex is uncertain. To address this issue, we propose the concept of *expected degree* which amalgamates the probabilistic characteristics of the graph into the expected value computation of the match events.

The statistical significance score of the vertex pair estimates the similarity of the vertex pair using the χ^2 measure. The vertex pair is then greedily expanded preferring its neighbor vertex pairs with high χ^2 values. A high-level overview of the algorithm is shown in Figure 1.2. Experiments conducted on billion-sized graphs and various synthetic datasets showed that ChiSeL has a very low runtime with high accuracy. Extensive experiments were done on various graph parameters for a deeper analysis of the algorithm. In comparison to deterministic ASM approaches,

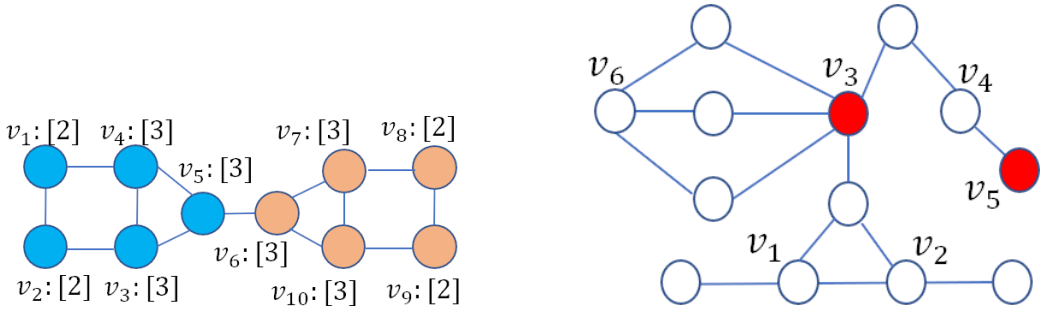
for real-life use cases, the subgraphs returned were highly relevant and accurate. The paradigm is flexible enough to adapt to other flavors of uncertain graphs as well.

1.2.3 VeNoM

Despite the considerable amount of research done in approximate subgraph querying methods, for both deterministic and probabilistic graphs, there is still room for researchers to conduct more in-depth analysis of ASM paradigms. Different factors can affect the subgraph querying algorithms, such as the degree of nodes, label distribution, missing edges, etc. The breadth and depth of the neighborhood considered is, perhaps, one of the most important aspects of an algorithm, while computing the similarity of two nodes.

For this analytical research, VeNoM chooses a state-of-the-art ASM paradigm, VELSET (Dutta et al., 2017) and improves over it. The approach of VELSET captures the underlying characteristics of a deterministic, node-labeled graph by defining its expectation for each vertex based on their degrees. However, the empirical analysis revealed that for input graphs with a lower number of unique labels, it results in an inversion of chi-square values, with bad matches receiving a higher similarity score than good matches. VeNoM proposes to use the query degree in expectation computation which is shown experimentally to perform better than VELSET.

It further delves into two of the several factors affecting an algorithm, the depth of the neighborhood considered when comparing the input graph vertex for a match with the query node, and the breadth of the neighborhood considered. We identify that the improved algorithm takes into account only 1-hop neighborhood while matching and matches only two labels of the neighborhood at a time. Algorithms are developed for different values of the depth and breadth of the neighborhood considered for the match. Through experiments, we discuss and analyze their performance with respect to the base algorithm on real-life graphs. Apart from the



(a) (b)

Figure 1.3: (a) The color of the node indicates its class label. Each node is also characterized by a numerical attribute. (b) The red nodes, v_3 and v_5, represent the anchor nodes.

design parameters of an algorithm, we also study the effect of graph parameters on the performance of the algorithms.

1.2.4 GraphReach

Although traditional graph mining techniques have shown promising results, the success of graph neural networks based methods inspired researchers to also explore machine learning based models for various graph mining problems. Traditional GNNs aggregate over neighbor attribute information to encode the information of a node into embedding vectors. However, such methods fail to capture the position of the node which may lead to identical node embeddings. For instance, nodes v_1 and v_8 in Figure 1.3a, belong to two different classes, denoted by the color of the node. However, since their 2-hop neighborhoods are isomorphic to each other, their learned embeddings in a 2-layer Graph Convolutional Network (GCN) are identical. Hence, the GCN will be incapable of correctly predicting that v_1 and v_8 belong to different class labels. Consequently, for predictive tasks that rely on the position of a node with respect to the graph, the performance suffers.

One of the first works to address the need for an inductive GNN that encodes position information is P-GNN (You et al., 2019), based on the shortest path to a set of selected nodes. P-GNN randomly chooses a small set of nodes, *anchor nodes*, and computes the shortest path of all nodes to these anchor nodes. This information

is then embedded in a low-dimensional space. However, if two nodes are equidistant to the anchor nodes, it would lead to ambiguous node embeddings. For example, in Figure 1.3b, the shortest distance of v_1, v_2 and v_6 is the same to the anchor nodes v_3 and v_5. The use of shortest paths also implies that the remaining paths of the graph are ignored and there is a loss of information. Further, in case of an adversarial attack, the addition or deletion of a few edges in the graph can alter the shortest path distances for target nodes, and, in turn, their embeddings. It has been shown that in real-life scenarios *many* short paths are preferred over the shortest path.

To this end, we present GraphReach, which is a holistic approach using random walks for *reachabilitity estimation* between anchors and nodes, which is a non-symmetric distance metric. It also employs a diversified anchor selection algorithm based on a greedy hill-climbing approach, that maximizes the graph coverage. In the neural architecture, GraphReach is initialized with the node attributes. For each graph node and anchor node pair, the attribute information of the two nodes is amalgamated with the to-and-fro reachability estimates, thus, encoding the positional information of the nodes with respect to the anchors, these are generally called *messages*. In the intermediate layers, messages corresponding to all the anchors are aggregated using an aggregation method, e.g., mean-pooling or attention-based aggregation, and passed on to the next layer as the node attribute. At the output layer, the message vectors are linearly transformed using a trainable weight matrix to lower dimensional embedding, each dimension of which encodes the positional information of the node corresponding to an anchor.

The node embeddings were tested on various datasets for different tasks and showed very high relative improvement of up to 40% over P-GNN and other existing GNN architectures. We also show that for the standard black-box adversarial setup GraphReach performed better with a negligible drop in performance compared to the state-of-the-art.

1.3 Outline

The thesis is organized into seven chapters. Chapter 1 provides an introduction to the topic and an overview of the problem statement. Chapter 2 reviews the existing state-of-the-art research works on subgraph matching. In Chapter 3, we present the methodology of the ASM approach VerSaChI over deterministic graphs. We identify the difficulties associated with probabilistic graphs and explore ChiSeL, the approximate subgraph querying technique, ChiSeL, which effectively addresses these challenges, in Chapter 4. In Chapter 5, we discuss different aspects and their effects analytically for an existing approximate subgraph matching approach. In Chapter 6, we discuss GraphReach, an inductive model for learning node-embeddings that encodes their global positioning in the graph. We conclude by presenting the conclusions of our study in Chapter 7 and discuss possible future work directions.

Chapter 2

Related work

2.1 Deterministic Graph Matching

Graph and subgraph matching provide fundamental primitives for applications pertaining to graph analytics and pattern mining in network structures (Conte et al., 2004). Classical studies in this domain include tree-pruning (Ullmann, 1976), Swift-Index (Shang et al., 2008) and VF2 (Cordella et al., 2004). Since they depend majorly on backtracking and tree-search algorithms, they are computationally costly for large modern-day graphs. Using a set of feasibility rules defined in (Cordella et al., 2004) for structural match, the VF2++ algorithm (Jüttner and Madarasi, 2018) improves over VF2. In (Larrosa and Valiente, 2002), the tree search method for isomorphism is sped up by another heuristic derived from constraint satisfaction. The NP-completeness of subgraph isomorphism (Cook, 1971) led to efforts towards graph edit distance (GED) measures for exact matches; however, the optimal solution for GED was shown to be NP-hard (Zeng et al., 2009). These approaches used state space representation and feasibility rules for graph isomorphism.

Shang et al. (Shang et al., 2008) proposed QuickSI for testing, and Swift-Index for filtering using prefix tree indexing. Most methods that target biological networks, such as PathBlast (Kelley et al., 2004), SAGA (Tian et al., 2006), NetAlign (Liang et al., 2006) and IsoRank (Singh et al., 2008), work mostly for small networks. Tsai and Fu (Tsai and Fu, 1979) proposed an ordered-search algorithm

for determining error-correcting isomorphism and pattern classification combining both structural and statistical techniques. GraphGrep (Giugno and Shasha, 2002), a graph querying algorithm, uses hash-based fingerprinting for subsequent filtering. Such *filtering-and-verification* based approaches worked with threshold-based distance computation, identifying common substructures, and candidate fragment computation. Identifying graphs in a graph database containing a query subgraph have also been studied (Sun and Luo, 2019). However, our target applications require the identification of the precise locations in the graph(s) where the best match of the query is found.

In general, such methods involved only exact subgraph matching, whereas this work involves approximate matching to extract the best matching subgraphs.

2.2 Approximate Graph Matching

Subsequently, research in subgraph matching has mostly focused on approximate subgraph similarity matches (Aggarwal and Wang, 2010), wherein a small amount of mismatch is tolerable. An efficient index-based approximate subgraph matching tool, TALE (Tian and Patel, 2008), uses maximum weighted bipartite graph matching. Different heuristics based on predefined graph distance and radius thresholds GADDI (Zhang et al., 2009b), set cover (SIGMA) (Mongiovi et al., 2010), edge edit-distance (SAPPER) (Zhang et al., 2010), regular expressions (Barcelo et al., 2011), etc. have been proposed. APGM (Jia et al., 2011) proposes a method to mine useful patterns from noisy graph databases. C-Tree (Zou et al., 2004) proposes a generalized representation of graphs that can be used for both subgraph querying and subgraph similarity computation. However, these methods are computationally quite expensive and are, hence, infeasible for modern web-scale graphs.

A semantic-based search algorithm using a sequencing method to capture the semantics of the underlying graph data was proposed in GString (Jiang et al., 2007). S^4 system (Yu et al., 2012) finds the subgraphs with identical structure and semantically similar entities of query subgraph. Other relevant works in the subgraph

matching domain are gIndex (Yan et al., 2005a), FG-Index (Cheng et al., 2007), iGraph (Han et al., 2010), Grafil (Yan et al., 2005b), Gcoding (Zou et al., 2008), GPTree (Zhang et al., 2009a), and cIndex (Chen et al., 2007). An extensive survey about graph matching algorithms was presented in (Conte et al., 2004). Random walks to find the best matching in large graphs were used in (Tong and Faloutsos, 2006; Tong et al., 2007). Subgraph matching considering the similarity between objects associated with two matching vertices (Zou et al., 2007), and a maximum likelihood estimation approach (Armiti and Gertz, 2014) were also proposed.

Recent approaches (Arora et al., 2014; Dutta et al., 2017) employ statistical analysis methods to find *statistically significant subgraph* matches that deviate from the expected subgraph pattern significantly. Dutta et al. (Dutta et al., 2017) improve upon NeMa (Khan et al., 2013) and SIM-T (Kpodjedo et al., 2014a) approaches based on neighborhood search. A concise description and comparison of various techniques available for graph matching in large graphs is given by (Mahmood et al., 2017). An interesting survey of existing graph-based structural and pattern matching approaches across diverse applications such as computer vision, biology, networks, etc., has been presented in (Gallagher, 2006). Inspired from (Dutta et al., 2017), VeNoM proposes an enhancement. Our proposed method, VerSaChI, employs Chebyshev's inequality to capture the underlying graph characteristics which captures finer differences between potential matches during statistical analysis.

These approaches consider deterministic graphs and are difficult to generalize for probabilistic setting wherein varying types of uncertainties might be present.

2.3 Probabilistic Graph Matching

Recently, with the advent of uncertain and probabilistic graphs such as knowledge graphs, RDF stores, etc., algorithms for probabilistic graph querying are being studied. Initially, convex optimization methods were proposed (Hintsanen and Toivonen, 2008). However, they found it difficult to handle large and noisy real-life graphs.

Inexact graph matching for uncertain graphs majorly comprises a three-phase framework: *structural pruning*, *probabilistic pruning* and *verification*. Often the uncertainty information in the graph is ignored and candidate answers are searched using conventional structural pruning algorithms, followed by probabilistic pruning of candidates and verification of the answer candidates. Utilizing this framework, (Yuan et al., 2012, 2015) compute tight probabilistic bounds for pruning of approximate subgraph matching based on a threshold for uncertain graphs with local correlations and adhering to the possible world semantics. Efficient upper and lower bounds were computed for relaxed query graphs by transforming the problem to set cover and integer quadratic programming problems in (Yuan et al., 2015). Additionally, (Yuan et al., 2016) constructs an optimal probabilistic index based on edge-cuts of target graph to compute an upper bound on *pattern matching probability* for pruning. The above *filter-and-search* framework performs well in subgraph matching. On arrival of a query, it retrieves and sorts the promising positions in the underlying deterministic graph with the help of index structures incorporating PWS, and verifies the results by checking for subgraph isomorphism. However, given a database of probabilistic graphs, it only returns graphs that contain the entire query subgraph and does not report the exact location of the query subgraph.

The direct approach described in (Gu et al., 2016) extends TreeSpan by efficient incremental similarity computation mechanism intertwined with structural pruning, with no attention to uncertainty information. Sampling has been shown to be effective in dealing with the hardness of managing and mining uncertain graphs (Leskovec and Faloutsos, 2006). The performance of sampling-based approaches depends heavily on the samples considered, though. Weighted subgraph matching algorithms were considered using the probability values as weights in (Hua and Pei, 2010; Khan and Chen, 2015). However, such techniques were unable to generalize to other possible uncertain scenarios.

Hua et al. (Hua and Pei, 2010) proposed three novel types of probabilistic path queries using basic principles. For probabilistic graph settings, a host of diverse

problems such as frequent subgraph mining (Chen et al., 2019; Li et al., 2012; Papapetrou et al., 2011; Zou et al., 2010), clustering (Kollios et al., 2011), reliable subgraphs (Liu et al., 2012), shortest-path (Hua and Pei, 2010), and maximum flow (Han et al., 2014) have been actively worked upon. Potamias et al. (Potamias et al., 2010) studied k-nearest neighbor queries over uncertain graphs, and propose sampling algorithms for tackling #P-completeness of reachability problems. Lian et al. (Li et al., 2018) proposed customization over existing tree-indexing strategies based on uncertain graph decomposition for k-NN queries. Reverse k-NN queries for uncertain graphs were studied in (Gao et al., 2017). Approximate subgraph matching queries on fuzzy RDF graphs using *path decomposition* was recently shown to be efficient (de Virgilio et al., 2015; Li et al., 2019a). A systematic introduction to the topic of managing and mining uncertain data can be found in (Yuan et al., 2011), while (Kassiano et al., 2016) provides a detailed overview of state-of-the-art methods on uncertain graph mining. Such path decomposition-based similarity techniques have been shown to be effective in the past, and, hence, we compare the empirical performance of ChiSeL with them. Interestingly, the neighborhood search step in ChiSeL can be considered to be similar to such path-based approaches taking into account the structure around a vertex.

2.4 Machine Learning based Graph Matching

Although the traditional graph mining techniques based on theoretical grounds have shown promising results, there is still scope for improvement. The increasing size and complexity of data and graphs, and the need for a smaller response time further motivate newer solutions. With the popularity of neural networks, data scientists have made several attempts at solving the problem of graph matching by learning node embeddings. Such neural networks are generally called *Graph Neural Networks (GNNs)*. As discussed previously, GNNs are neural networks designed to directly work on graphs. The objective is to learn a function for task prediction, e.g., node classification, link prediction, graph classification, etc.

Several GNNs exist in the literature. DeepWalk (Perozzi et al., 2014) learns node representation inspired by the skip-gram model while Node2vec (Grover and Leskovec, 2016) takes it a step further with more sophisticated random walks. GCN (Kipf and Welling, 2017) is a semi-supervised variant of convolutional networks. These methods are *transductive* in nature, i.e., it requires complete graph structure at the training time and cannot generalize to unseen nodes. In contrast, GAT (Velickovic et al., 2018) is an inductive model that expands on GCN by assigning attention weights to important connections. GraphSAGE (Hamilton et al., 2017) proposed sampling the neighborhood is more efficient and robust for large graphs. GIN (Xu et al., 2019) is another GNN that claims to be as powerful as the Weisfeiler-Lehman graph isomorphism test.

These models can be used for various downstream tasks, like graph matching. SimGNN (Bai et al., 2019) uses GCN node embeddings to predict graph similarity based on the graph edit distance. NeuroMatch (Lou et al., 2020) proposes subgraph matching based on graph decomposition and learns embeddings for subgraphs. A matching matrix based training for graph matching is presented in (Caetano et al., 2009), while (Baskararaja et al., 2012) presents a GNN to identify a subgraph that matches the query graph. Sub-GMN (Lan et al., 2021) discusses a subgraph matching network for node embeddings and models the relationship between matched nodes. IsoNet (Roy et al., 2022) proposes subgraph matching based on neural edge alignment of query and input graph while (Liu et al., 2020) uses deep learning methods to predict the count of isomorphic subgraphs.

Most GNNs, in general, work on the message passing principle from neighbors alone, i.e., the embedding of a node depends only on its neighborhood. For cases, where the location of the node in the context of the graph structure is important for the prediction task, such methods would inadvertently fail. Position-aware GNNs try to bridge this gap, by incorporating messages from nodes in other parts of the graph, which is generally referred to as the *anchor nodes*. P-GNN (You et al., 2019) is the first work to propose this idea. It randomly selects a set of anchor nodes and

learns low dimensional node embeddings based on Lipschitz embeddings (Bourgain, 1985). A-GNN (Liu et al., 2019a) proposes a modified algorithm for anchor-selection based on the importance of their position in the graph. In our work, we discuss the shortcomings of relying on shortest paths alone and propose a random walk based approach for a more holistic and robust distance metric and anchor set selection technique. More recently, PO-GNN (Waikhom et al., 2023) proposed a truncated distance metric and an improvised anchor-selection mechanism over GraphReach without significant gains.

Chapter 3

VerSaChI

In this Chapter, we present the workings of our proposed approximate subgraph matching algorithm for deterministic graphs, *Vertex Neighborhood Aggregation for Statistically Significant Subgraphs via Chebyshev Inequality* - **VerSaChI** - for efficient top-k subgraph matching. We show how deviations from the underlying graph characteristics, modeled using probabilistic bounds, can efficiently provide label and structural similarity measures for approximate subgraph matching. Experimental results on various real and synthetic datasets showcase how the proposed algorithm outperforms existing techniques in accuracy and is robust to noise.

3.1 Preliminaries

Before discussing the algorithm, for ease of understanding let us familiarize ourselves with some of the terms and notations that will be commonly used in this work.

Definition 3.1. *Target graph ($\mathcal{G}$). A target or input graph $\mathcal{G} = (\mathcal{V}_\mathcal{G}, \mathcal{E}_\mathcal{G}, \mathcal{L}_\mathcal{G}, \Gamma)$, is a large graph where, $\mathcal{V}_\mathcal{G}$ refers to the vertex set, $\mathcal{E}_\mathcal{G}$ refers to the set of edges, $\mathcal{L}_\mathcal{G}$ is a function that maps vertices of the graph to a label, and Γ is the set of all labels.*

Definition 3.2. *Query graph ($\mathcal{Q}$). A query graph $\mathcal{Q} = (\mathcal{V}_\mathcal{Q}, \mathcal{E}_\mathcal{Q}, \mathcal{L}_\mathcal{Q})$, is a small graph where, $\mathcal{V}_\mathcal{Q}$ refers to the vertex set, $\mathcal{E}_\mathcal{Q}$ refers to the set of edges, and $\mathcal{L}_\mathcal{Q}$ is a function that maps the vertices of the graph to a label.*

The input and the query graph share the label space, i.e., $\mathcal{L}_{\mathcal{G}} : \mathcal{V}_{\mathcal{G}} \rightarrow \Gamma$ and $\mathcal{L}_{\mathcal{Q}} : \mathcal{V}_{\mathcal{Q}} \rightarrow \Gamma$.

Definition 3.3. *Candidate Vertex Pair $\langle q, v \rangle$. When a query node $q \in \mathcal{V}_{\mathcal{Q}}$ and an input vertex $v \in \mathcal{V}_{\mathcal{G}}$ have a similar label, i.e., $\mathcal{L}_{\mathcal{Q}}(q) \approx \mathcal{L}_{\mathcal{G}}(v)$, then v is a candidate match for q.*

In general, a vertex pair implies candidate vertex pair with a query node and a target vertex. For ease of understanding, as a convention, any random pair of vertices is referred to using parentheses, e.g., (u, v), while angular brackets, e.g., $\langle q, v \rangle$, imply a candidate vertex pair with a query node and input vertex. Also, we refer to a vertex of the query graph as a *node* and the target vertex as *vertex*. The similarity of labels is an application dependent metric, and any standard technique can be used to compute it, Jaccard similarity, string edit distance, etc. However, for our purpose, without loss of generality, we assume exact label matching.

3.1.1 Methodology overview

VerSaChI uses *statistical significance* to quantify the similarity of a vertex pair. We use *Pearson's chi-square statistic* (χ^2) to measure the probability of attributing an observed event to chance or randomness. Mathematically,

$$\chi^2 = \sum_{\forall i} \frac{(O_i - E_i)^2}{E_i} \tag{3.1}$$

where O_i and E_i are the observed and expected number of occurrences for all outcomes, respectively.

To aid in χ^2 computation we use the *Chebyshev's inequality* (de Tchébychef, 1867). It models the probability of deviation for a random variable in terms of the number of standard deviations from the mean. For a random variable X with finite mean μ and varianceδ^2, $\Pr(|X - \mu| \geq t \cdot \delta) \leq 1/t^2$ for any $t > 0$ ($t \in \mathbb{R}$). Intuitively, the degree of matching (i.e., similarity) between a query graph and its matching subgraph would demonstrate significant deviations (due to high similarity) from the

expected characteristics (considering a random subgraph). Chebyshev's inequality is used to characterize such *deviations* for computing the statistical significance of candidate matching subgraphs.

3.2 VerSaChI Framework

The working of VerSaChI comprises the following stages. The first five steps are *offline* and compute the different characteristics for each vertex – done only once for the target graph. The next 4 steps, the *online* phase upon query arrival, compute characteristics for the query nodes and identifies candidate neighborhood regions matching the query by using *two-hop* label and structural overlap. The deviation of the observed similarity, from the underlying distribution is then characterized by Chebyshev's inequality and represented as symbols. Based on statistical significance, matching candidate regions are identified and explored greedily, to obtain the best matching subgraphs.

3.2.1 Offline Phase

The offline phase of VerSaChI consists of index creation and expectation definition for efficient computations in the online phase.

1. **Index Creation.**

   Given a target graph $\mathcal{G}$, VerSaChI initially constructs two indexing lists summarizing the labels of the vertices and their neighbors. The first is an *inverted list*, $IL_{\mathcal{G}}$, that maps vertex labels to the corresponding list of vertices having the label. The second index, the *label neighbor list*, $LNL_{\mathcal{G}}$, stores the label information of the neighbors for each vertex in $\mathcal{G}$. A *label count vector* index, $LCV_{\mathcal{G}}(u), \forall u \in \mathcal{V}_{\mathcal{G}}$ is also constructed. It stores the count of occurrence of each label (for $|\Gamma|$ labels) in the neighborhood of u. This enables efficient computation of similarity between vertices as described next (step 4 onwards in offline and throughout the online phase).

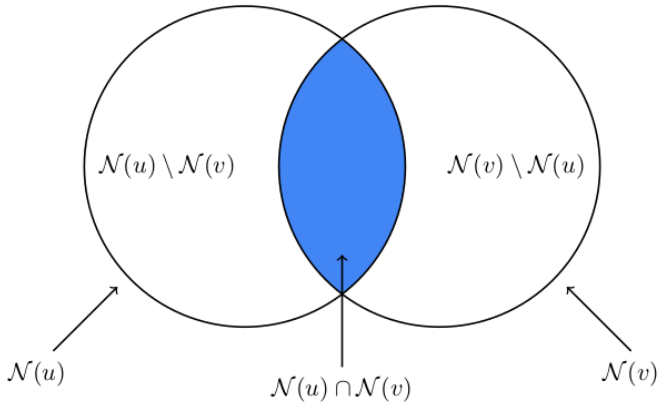


Figure 3.1: Venn diagram representing the neighborhood overlap and mismatch considered for similarity between two vertices

2. **Similarity Measure.**

   For *any* pair of vertices (u, w), we use a modified *Tversky index* (Tversky, 1977) to compute similarity in their neighborhood.

$$\eta_{(u,w)} = \frac{|\mathcal{N}(u) \cap \mathcal{N}(w)|}{|\mathcal{N}(u) \cap \mathcal{N}(w)| + |\mathcal{N}(u) \setminus \mathcal{N}(w)|^{\gamma}} \tag{3.2}$$

   where $\mathcal{N}(u)$ is the multiset of labels in the neighborhood of u including the label of u itself. Observe, by setting $\gamma = 1$, we obtain the original Tversky index[1] with $\alpha = 1$ and $\beta = 0$. Intuitively, the similarity of w is maximized w.r.t. u when all neighbor labels of u are present in the neighbors of w (i.e., $\mathcal{N}(u) \subseteq \mathcal{N}(w)$). For ease of understanding, Figure 3.1 shows a Venn diagram of the multisets involved in the Equation 3.2. To elaborate, on the arrival of the query graph, for a candidate pair $\langle q, v \rangle$, we argue that the presence of additional neighbors of v should not affect the final similarity score negatively, hence, we set $\beta = 0$ in the Tversky index. In essence, the equation captures the *neighborhood recall* of u provided by w (thus, $\alpha = 1$). The exponential *penalty factor*, γ, penalizes increasing mismatches in the neighborhood label overlap between vertex pairs. It captures fine differences in the neighborhoods by accentuating even smaller mismatches. Empirically, $\gamma = 3$ gave the best

[1] Tversky index:
$S(X, Y) = |X \cap Y| / \big(|X \cap Y| + \alpha|X \setminus Y| + \beta|Y \setminus X|\big)$, for sets X and Y with $\alpha, \beta \geq 0$.

results.

3. **Initialization.**

   Using $LCV_{\mathcal{G}}(u)$ and similarity measure $\eta_{(u,w)}$, VerSaChI computes the vertex similarity scores for every pair of vertices in $\mathcal{G}$. This captures the underlying distribution of the input graph. The expected similarity distribution across random neighborhoods of $\mathcal{G}$ is captured via three characteristics computed using $\eta_{(u,w)} : \forall(u \in \mathcal{G}, w \in \mathcal{G})$

   i. ***Average*** vertex similarity score for all vertex pairs in $\mathcal{G}$:

   $$\psi(\mathcal{G}) = \frac{\sum_{u,w \in \mathcal{V}_{\mathcal{G}}} \eta_{(u,w)}}{|\mathcal{V}_{\mathcal{G}}|} \tag{3.3}$$

   ii. ***Standard deviation*** of the vertex similarity scores of $\mathcal{G}$:

   $$\delta(\mathcal{G}) = \left(\frac{\sum_{u,w \in \mathcal{V}_{\mathcal{G}}} (\eta_{(u,w)} - \psi(\mathcal{G}))^2}{|\mathcal{V}_{\mathcal{G}}| - 1} \right)^{1/2} \tag{3.4}$$

   iii. ***Maximum*** deviation of vertex similarity score from the average among all the vertex pairs in terms of standard deviations in $\mathcal{G}$, this can also be referred to as the maximum *z-score*:

   $$\triangle(\mathcal{G}) = \max_{u,w \in \mathcal{V}_{\mathcal{G}}} \left\{ \frac{|_{(u,w)} - \psi(\mathcal{G})|}{\delta(\mathcal{G})} \right\} \tag{3.5}$$

4. **Symbol Categorization.**

   The degree of matching between a query node and target graph vertex is captured by the amount of deviation of the vertex pair similarity score (in terms of the number of standard deviations) from the underlying expected distribution (computed in the previous step). The standard deviations are discretized using the *step size* parameter, κ. It also determines the total number of possible buckets or *symbols*, $\tau = \lceil (\triangle(\mathcal{G}) - 1)/\kappa \rceil$. The set of category symbols, therefore, is $\Sigma = \{\sigma_1, \sigma_2, \cdots, \sigma_\tau\}$. Smaller values of κ are

preferred for differentiating between finer-grained structural mismatches.

For a pair of vertices (u, w), its similarity is characterized using the symbol σ_i, $1 \leq i \leq \tau$. The first symbol, σ_1, spans the range of standard deviations up to $1 + \kappa$, i.e.,

$$\sigma_1 : 0 \leq \frac{|\eta_{(u,w)} - \psi(\mathcal{G})|}{\delta(\mathcal{G})} < 1 + \kappa \tag{3.6}$$

Subsequent symbols cover step size standard deviations each, for $2 \leq i \leq \tau$,

$$\sigma_i : 1 + (i-1) \cdot \kappa \leq \frac{|\eta_{(u,w)} - \psi(\mathcal{G})|}{\delta(\mathcal{G})} < 1 + i \cdot \kappa \tag{3.7}$$

5. **Expected Probabilities of Symbols.**

The expected probability of occurrence of the category symbols is next computed using the *Chebyshev's inequality*. Observe, the deviation of similarity of a vertex pair from the mean can be in negative or positive direction.

Since we are interested in vertices that have higher similarity than the mean (to capture a higher degree of matching), we only discretize the similarity (into symbols) when it is greater than the mean. For all similarities that are lesser than the mean, we fold them into the symbol σ_1 (Figure 3.2 shows the span of each symbol for easier understanding). Thus, assuming a symmetric one-sided Chebyshev's inequality, the occurrence probability of symbol σ_i is,

$$\Pr(\sigma_i) \approxeq \frac{1}{2}\Big[\frac{1}{(1 + (i-1) \cdot \kappa)^2} - \frac{1}{(1 + i \cdot \kappa)^2}\Big] \tag{3.8}$$

for $2 \leq i \leq \tau$, and

$$\Pr(\sigma_1) = 1 - \sum_{j=2}^{\tau} \Pr(\sigma_j) \tag{3.9}$$

We also empirically evaluated the variant where the deviation where both the positive and negative side of the mean are considered (i.e., without folding). However, it produced no changes in our results. Since a very low similarity (large negative deviation) can potentially have large chi-square values and,

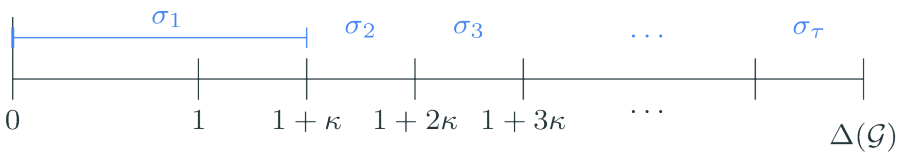


Figure 3.2: A representation of the span of symbol categories

thus, produce false matching results, VerSaChI uses the one-sided version.

3.2.2 Online phase

The above steps are *offline*, and performed only once for a target graph. Once a query arrives, the following *online* steps take place.

1. **Candidate Pair Mapping.**

   Upon arrival of a query graph $\mathcal{Q}$, the *online* processing starts with the construction of indexes $IL_{\mathcal{Q}}$, $LNL_{\mathcal{Q}}$, and $LCV_{\mathcal{Q}}$, analogously to $\mathcal{G}$. For each *label* in $\mathcal{Q}$, VerSaChI creates *candidate pairs* between the query nodes and target vertices having the same label. These candidate pairs form the initial seed vertex for extracting matching subgraphs (to the query) via greedy neighborhood search. Formally, the candidate pairs generated are

   $$CP = \{\langle q \in \mathcal{Q}, v \in \mathcal{G}\rangle \mid \mathcal{L}_{\mathcal{G}}(v) = \mathcal{L}_{\mathcal{Q}}(q)\} \tag{3.10}$$

2. **Vertex Symbol Sequence.** For a candidate pair $\langle q, v\rangle$, VerSaChI computes the vertex pair similarity score, $\eta_{\langle q,v\rangle}$, and characterizes the similarity score by assigning a category symbol based on the deviation from the expected similarity distribution (as discussed previously). The category symbol $\sigma_{\langle q,v\rangle}$ captures the one-hop neighborhood similarity for vertices q and v.

   We next compute "second-order" candidate pairs between the vertex sets adjacent to q and v. A *greedy* best mapping based on the vertex pair similarity score is used to compute the second-order candidate pairs. Similar to $\langle v, q\rangle$, each second-order candidate pair is assigned a category symbol based on the deviation of its similarity score from the expected. The initial category symbol

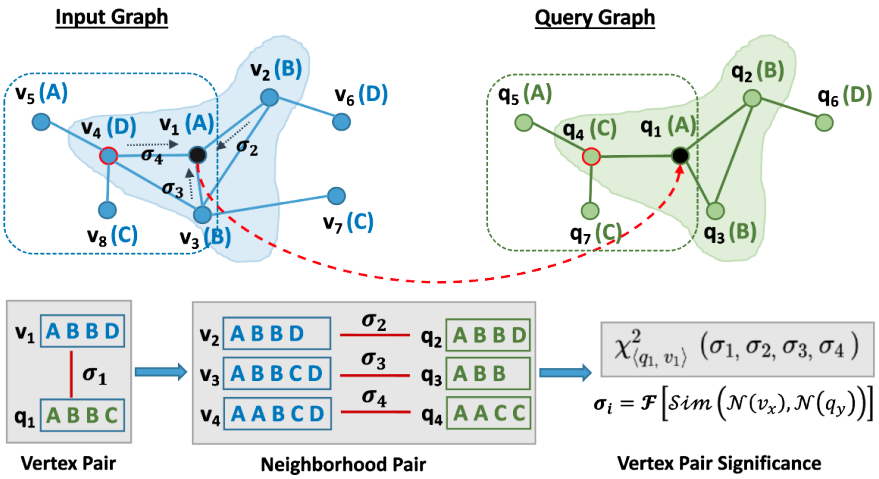


Figure 3.3: Two-hop neighborhood similarity based computation of χ^2 statistical significance for vertex match in VerSaChI.

$\sigma_{\langle q,v\rangle}$ is aggregated with the second-order category symbols to form the *vertex symbol sequence*, $O_{\langle q,v\rangle}$, for the candidate pair $\langle q, v\rangle$.

As an example, consider Figure 3.3 depicting an initial candidate pair between vertices q_1 and v_1 (both having label A) with category symbol σ_1 assigned to it. The adjacent vertices of v_1 ($\{v_2, v_3, v_4\}$) and the neighbors of q_1 ($\{q_2, q_3, q_4\}$) are then greedily best-matched based on vertex pair similarity to obtain the "second-order" candidate pairs. For instance, v_2 and q_2 provide the best match with the same label and the same neighborhood labels and, thus, form the next candidate pair (with, say, category symbol σ_2). Subsequently, v_3 and q_3 are matched having the same label and partial neighborhood overlap (consider to be assigned symbol σ_3). Finally, the candidate pair $\langle q_4, v_4\rangle$ is obtained with category symbol σ_4. The corresponding vertex symbol sequence, $O_{\langle q_1,v_1\rangle} = \{\sigma_1, \sigma_2, \sigma_3, \sigma_4\}$, associated to $\langle q_1, v_1\rangle$, captures the two-hop similarity between the candidate pair vertices q_1 and v_1.

3. **Statistical Significance.**

The computed symbol sequence $O_{\langle q,v\rangle}$ signifies the degree of matching between the two-hop neighborhoods of q and v. Assuming d to be the *degree* of q, since mapping is performed for the neighbors of q, the length of $O_{\langle q,v\rangle}$ is d. Thus, the *expected occurrence counts* of category symbol σ_i is $E[\sigma_i] = d \cdot \Pr(\sigma_i)$.

The *observed occurrence counts* of the category symbols are directly obtained from $O_{\langle q,v\rangle}$. Using the observed and expected counts, VerSaChI computes the *chi-square statistics*, $\chi^2_{\langle q,v\rangle}$, for all the candidate pairs obtained in CP (Equation 3.10).

4. **Approximate Matching.**

   The candidate pairs along with their computed chi-square values, $\langle q, v, \chi^2_{\langle q,v\rangle}\rangle$, are inserted into a *primary max-heap* structure. The candidate pair with the largest χ^2 value is extracted (assume $\langle v, q\rangle$) for initializing the top-1 matching subgraph, $Match^{(1)}$, and is considered as the seed vertex pair for greedy expansion to find a matching subgraph for the query $\mathcal{Q}$.

   Next, candidate pairs between adjacent vertices of the extracted seed pair, i.e., between neighbors of q and v, are constructed (as in step 1 of the online phase) and pushed into a *secondary max-heap* structure. As before, vertex symbol sequences of the candidate pairs in the secondary heap are constructed, and their statistical significance scores are computed. The pair with the highest χ^2 value is extracted, and added to $Match^{(1)}$. This process is iterated till the secondary heap is empty, or the size of $Match^{(1)}$ equals the number of vertices in $\mathcal{Q}$. The subgraph obtained in $Match^{(1)}$ is reported as the *top-1 best approximate matching subgraph* for $\mathcal{Q}$.

   Vertices extracted from the primary and secondary heaps are marked as "done" to prevent duplicate entries in the heap structures, and ensure that the same region is not repeatedly explored.

To retrieve *top-k* approximate matches for a query, the secondary heap is reset and the process is re-run, starting from picking the currently best candidate pair (with the highest statistical significance) from the primary heap. This is repeated until k matches are obtained.

Dataset	# Vertices	# Edges	$\|\Gamma\|$
Human	4.7K	86.2K	44
HPRD	9.5K	37K	307
Protein	43.5K	81K	3
Flickr	80.5K	5.9M	195
IMDb	428.4K	1.7M	22

Table 3.1: Summary of the characteristics of the datasets

3.2.3 Complexity Analysis

For graph $\mathcal{G} = (\mathcal{V}_\mathcal{G}, \mathcal{E}_\mathcal{G}, \mathcal{L}_\mathcal{G}, \Gamma)$, assume $n = |\mathcal{V}_\mathcal{G}|$ and $m = |\mathcal{E}_\mathcal{G}|$, $|\Gamma|$ denotes the number of unique labels. The overall space complexity of VerSaChI, is $O(n + m + \tau + n \cdot |\Gamma|)$, since during index construction (offline phase), $O(n)$ space is required for $IL_\mathcal{G}$, $O(m)$ for $LNL_\mathcal{G}$, $O(n \cdot |\Gamma|)$ for $LCV_\mathcal{G}$, and symbol probabilities require $O(\tau)$. The time taken for index construction are $O(n)$ for $IL_\mathcal{G}$, and $O(m)$ for both $LNL_\mathcal{G}$ and $LCV_\mathcal{G}$. Computing the target graph underlying distribution requires traversal of $LCV_\mathcal{G}$ for each vertex pair in $\mathcal{G}$. Thus, total *offline* time is $O(n+m+n^2 \cdot |\Gamma|) \approx O(n^2)$.

Once a query arrives, for each query node, candidate pairs (with the same label) are constructed using the inverted indices. Assuming uniform label distribution in $\mathcal{G}$, the number of candidate pairs is $O(n_\mathcal{Q} \cdot n/|\Gamma|)$, where $n_\mathcal{Q}$ is the number of nodes in $\mathcal{Q}$. For each candidate vertex pair, vertex symbol sequence generation (both initial and "second-order") takes $O(\rho \cdot |\Gamma|)$ time where ρ is the maximum degree in $\mathcal{G}$. Since χ^2 computation takes $O(\tau)$ time, the overall runtime of VerSaChI is $O(n_\mathcal{Q} \cdot n/|\Gamma|)$.

3.3 Experiments

In this section, we discuss the empirical setup and evaluation of VerSaChI with other approaches.

3.3.1 Setup

All experiments were implemented in C++ and were conducted on an Intel(R) Xeon(R) 2.60GHz CPU E5-2697v3 with 500GB of RAM.

Baselines

We compare the performance of VerSaChI algorithm against the following:

- *VELSET* (Dutta et al., 2017), a statistical significance based approach for exploring candidate regions with partial label match, and
- *G-Finder* [2](Liu et al., 2019b), a graph traversal based indexing for dynamic filtering and refinement of matching neighborhoods.

Datasets

We evaluate the algorithms on real datasets from three different domains:

i. *Biological Networks.*
   Protein-protein graphs of Human, HPRD (Bi et al., 2016) and Protein (Rossi and Ahmed, 2015).

ii. *Social Interaction.* user interaction network on the hosting site Flickr (Rossi and Ahmed, 2015) with the label of each user (vertex) denoting the group that they belong to.

iii. *Knowledge Graph.*
   Movie relationship graph IMDb (Rossi and Ahmed, 2015) containing named entities like movies, actors, etc., along with their relationships.

The dataset characteristics are shown in Table 3.1, $|\Gamma|$ denotes the number of unique labels in the graph. Using synthetically generated *Barabási-Albert* graphs we study the scalability of VerSaChI.

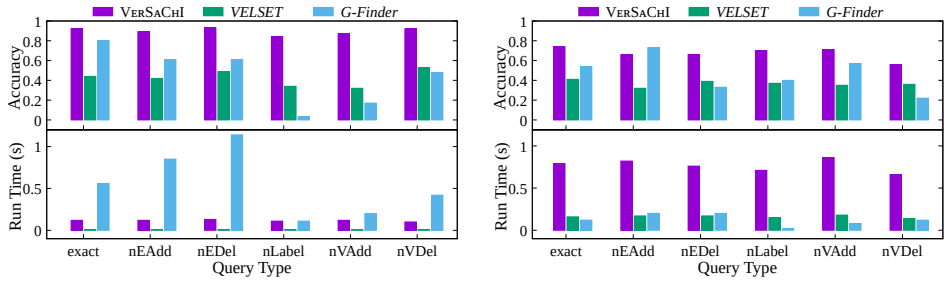
Query Generation

Query graphs (connected) are constructed (from the datasets) by initially selecting a random vertex, and exploring its neighborhood till n_Q vertices are visited. These

[2]G-Finder was obtained from `github.com/lihuiliullh/GFinder` and evaluated on a Visual Studio 2015 C++ platform.

Dataset / Algorithm	Accuracy					Running Time (sec)				
	Human	HPRD	Protein	Flickr	IMDb	Human	HPRD	Protein	Flickr	IMDb
VELSET	0.42	0.65	0.37	0.75	0.53	0.01	0.01	0.16	0.13	1.31
G-Finder	0.45	0.12	0.47	out of memory		0.55	0.01	0.12	out of memory	
VerSaChI	**0.90**	**0.81**	**0.67**	**0.84**	**0.87**	0.12	0.06	0.77	1.98	6.90

Table 3.2: Overall accuracy and runtime performance of the algorithms on the different datasets.



(a) Human **(b)** IMDb

Figure 3.4: Performance for different query types on Human and IMDb datasets.

are referred to as *exact queries*. To study the performance of the algorithms in the presence of noise, exact query graphs were perturbed by introducing structural and label noise randomly by (i) modifying node labels (*nLabel*), or (ii) inserting or deleting nodes (*nVAdd* and *nVDel* resp.), or (iii) adding or deleting edges (*nEAdd* and *nEDel* resp.). The number of perturbations is limited to 2. Further, for each scenario, we generate queries with sizes varying from 3 to 13 (at intervals of 2), with 20 query graphs extracted for each size. Thus, for each dataset, we consider $(6 \times 6 \times 20) = 720$ queries, and report average results across all the queries for a dataset. For *Barabási-Albert* graphs only exact queries were considered.

Evaluation Metrics

The efficiency of the algorithms is measured in terms of edge retrieval accuracy (using the labels of end vertices), that is, the fraction of edges of the query graph $\mathcal{Q}$ that are present in the matching subgraph retrieved. Additionally, we report the average runtime required (per query graph) by the different approaches to extract the approximate matching subgraphs. Since the introduced perturbations do not exist in the original graph, the exact query (for obtaining the noisy query) is considered as the ground truth.

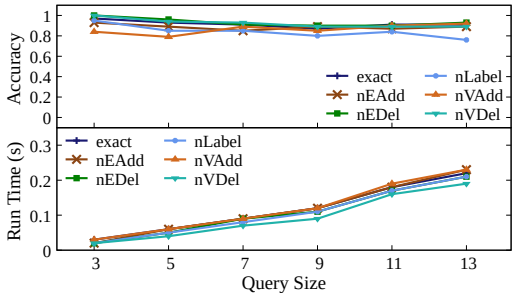


Figure 3.5: Performance over query size on Human dataset.

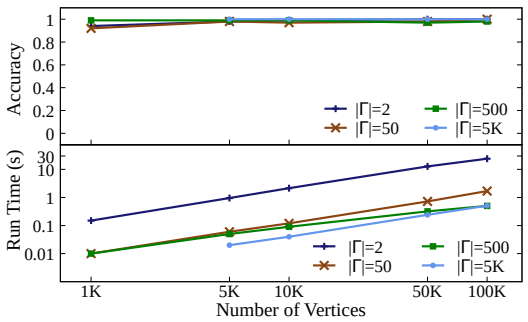
3.3.2 Results

From Table 3.2, we observe that VerSaChI has significantly better accuracy than the competing algorithms for finding the best matching subgraphs with more than 20% accuracy improvements (averaged across varying query types and sizes). The run-time of VerSaChI is slightly more than the other approaches due to the two-hop neighborhood similarity computation. In absolute terms, though, it is quite practical. Overall, with a slight increase in compute time, VerSaChI offers a substantial accuracy gain. *G-Finder* crashed due to out-of-memory issues for Flickr and IMDb datasets.

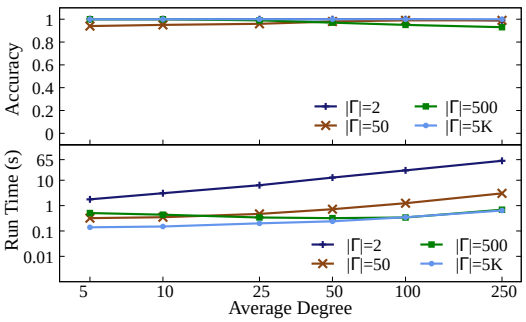
Figure 3.4 depicts the performance for different query types. Results on the other datasets were observed to be similar. VerSaChI achieves better accuracy across all the different query types, with a slight increase in runtime. Figure 3.5 shows that with an increase in query size, the runtime increases linearly (across query types), while the accuracy remains largely unaffected.

Figure 3.6 studies the scalability of VerSaChI using synthetic *Barabási-Albert* graphs. The runtime is seen to increase linearly with the number of vertices and average degree, conforming to the analysis of Section 3.2. The accuracy of VerSaChI is unaffected in these scenarios. With an increase in the *step size* κ, accuracy decreases, as the number of symbols decreases, limiting the power of VerSaChI to differentiate between finer differences in neighborhood mismatches between the graphs, while the runtime remains mostly constant.

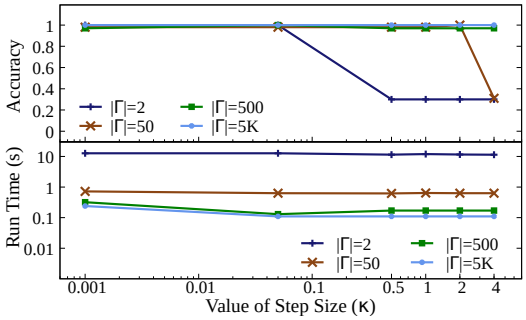
The maximum index size taken by VerSaChI in our experiments was 1.4 GB for



(a) Number of Vertices
(Avg. Degree = 50, $\kappa = 0.001$)



(b) Average Degree
($|V| = 50K$, $\kappa = 0.001$)



(c) Step Size (κ) ($|V| = 50K$, Avg. Degree = 50)

Figure 3.6: Performance of VerSaChI on Barabási-Albert graphs with varying (a) number of vertices, n, (b) average degree, and (c) step size, κ.

the Flickr graph, while the highest offline computation time was 32783.44 seconds, for the IMDb dataset.

Chapter 4

ChiSeL

In real-world scenarios, the obtained data is often inherently *uncertain* due to noisy measurements (e.g., missing edges or labels), hardware limitations, inference models, multiple data-source merging, privacy-preserving perturbation, etc. For example, pairwise protein interactions are usually derived by statistical models while potential interactions in social networks are based on trust and influence factors. Similarly, confidence values may be associated with extracted facts by information extraction systems. Further, in many scenarios information is often vague or ambiguous, expressing subjective opinions and judgments concerning market analysis, medical diagnosis or even personal evaluation. Such uncertainties can be classified into three categories (Moustafa et al., 2014):

- *identity uncertainty* where an entity is represented by multiple objects or references in the data,
- *attribute uncertainty* about the attribute values of entities (e.g., vertex existence or label uncertainty), and
- *relationship uncertainty* about whether a particular relationship exists (e.g., edge existence or label uncertainty).

A natural way to capture graph uncertainty is to represent them as *probabilistic graphs* (Jin et al., 2011; Valiant, 1979), where each entity, relation, or characteristic

is associated with a probability quantifying the existence likelihood. For example, in the automated creation of knowledge bases such as YAGO and DBpedia, extraction of entities, facts and relationships have an associated confidence depending on the veracity of the data source, information extraction technique, and errors in the pipeline. Without loss of generality, in this work, we only consider the existence of *edge uncertainties*, i.e., each relationship between entities has an associated probability of confidence of existence.

Existing literature defines two main representations of edge uncertainty in probabilistic graphs (Hua and Pei, 2010; Maniu et al., 2017):

- *edge-existential model*, where each edge is quantified with a probability indicating the chance of edge existence, and
- *weight-distribution model*, where each edge is associated with a probability distribution of weight values.

We adopt the former representation model in this work. Later, in Section 4.9, we discuss how various other scenarios involving vertex, edge, and label uncertainties can be handled by our algorithm.

We start the discussion by formally defining the problem at hand.

Problem Statement

We borrow the terminologies defined in the previous chapter in Section 3.1, with a slight modification to the definition of the target graph. Formally, without loss of generality, we represent an input graph a $\mathcal{G} = (\mathcal{V}_\mathcal{G}, \mathcal{E}_\mathcal{G}, \mathcal{L}_\mathcal{G}, \Gamma, \mathcal{P})$ where $\mathcal{V}_\mathcal{G}$ is the set of vertices and $\mathcal{E}_\mathcal{G}$ is the set of edges. The vertex labels $\mathcal{L}_\mathcal{G} : \mathcal{V}_\mathcal{G} \rightarrow \Gamma$ is a mapping of vertices to labels drawn from a finite set $\Gamma = \{l_1, l_2, \cdots l_{|\Gamma|}\}$ of cardinality $|\Gamma|$. The mapping $\mathcal{P} : \mathcal{V}_\mathcal{G} \rightarrow [0, 1]$ is defined on the set of edges to obtain the associated probability of existence for each edge. We assume that G is undirected and does not contain any hyperedge. The query graph, represented by $\mathcal{Q} = (\mathcal{V}_\mathcal{Q}, \mathcal{E}_\mathcal{Q}, \mathcal{L}_\mathcal{Q})$, specifies the structure and the labels on the vertices. Since users generally specify a query

completely, there is no uncertainty in the query edges. Without loss of generality, as previously, Γ is assumed to include the query labels.

Given a deterministic query $\mathcal{Q}$ and a large probabilistic graph $\mathcal{G}$ as above, our aim is to find the *top-k statistically significant subgraphs* of $\mathcal{G}$ that are the best approximate matches of $\mathcal{Q}$.

4.1 ChiSeL Framework

The ChiSeL algorithm has two distinct phases, namely, the *indexing* and the *querying* phases. In the indexing phase, several *inverted index lists* for mapping between vertices, labels, and neighbors are constructed. Observe that this phase is a one-time pre-processing step for an input graph (to boost the performance of querying later) and is independent of the query (i.e., performed even before the queries arrive). The querying phase is initiated when a query arrives, and the index structures are then used to efficiently compute statistical significance scores to guide the search process for finding the best matching subgraph. We next describe in detail the working of the two phases in ChiSeL.

4.2 Indexing Phase

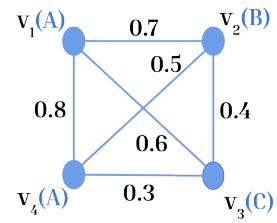
In the offline indexing phase, ChiSeL constructs several indexes to store vertex and neighborhood information from the input target graph G. Specifically, we construct the following.

- **Vertex-Label Inverted Index.**

  The vertex-label inverted index stores the mapping between the labels and the vertices of the input graph, $\mathcal{G}$. Here, the labels present in $\mathcal{L}_{\mathcal{G}}$ are considered to be keys, while the vertices associated with those labels form the values.

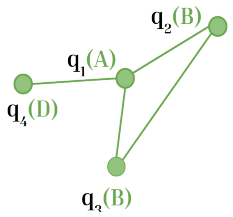
- **Neighbor Labels List.**

  For each vertex in $\mathcal{G}$, ChiSeL stores the neighboring vertices along with their



Target Graph - $\mathcal{G}$

Vertex	Label	Neighbors
v1	A	(A, 0.8), (B, 0.7), (C, 0.6)
v2	B	(A, 0.7), (A, 0.5), (C, 0.4)
v3	C	(A, 0.6), (A, 0.3), (B, 0.4)
v4	A	(A, 0.8), (B, 0.5), (C, 0.3)



Query Graph - $\mathcal{Q}$

Vertex	Label	Neighbors	Triplet-list
q_1	A	B,C,D	(B, A, C), (C, A, D), (B, A, D)
q_2	B	A,C	(A, B, C)
q_3	C	A,B	(B, C, A)
q_4	D	A	(A, D, ϕ)

Figure 4.1: Running example of a subgraph matching query graph $\mathcal{Q}$ for the probabilistic target graph $\mathcal{G}$.

labels and the corresponding edge existential probabilities by using an adjacency list structure. Figure 4.1 depicts the construction of the above indexes for our example graphs.

4.2.1 Expected Degree Computation

Since the graph is probabilistic, the existence of neighbors of a vertex is uncertain. Assuming that the existence of each edge is an *independent* event, the *expected degree* of a vertex is computed as the sum of probabilities of edges incident on it. Thus, if a vertex v has n_v neighbors connected with edges with probabilities $p_1, p_2, \cdots, p_{n_v}$, the expected degree, δ_v, of v is given by,

$$\delta_v = E[deg(v)] = \sum_{i=1}^{n_v} p_i \tag{4.1}$$

We prove this by induction. Assume the base case where there is only one neighbor with probability p_1. The expected number of neighbors, therefore, is $\delta = p_1 \times 1 + (1-p_1) \times 0 = p_1$. Assume that for $n-1$ neighbors, the expected degree is $\delta = \sum_{i=1}^{n-1} p_i$. If another vertex with edge probability p_n is added, then with probability p_n, the number of neighbors is $\delta + 1$, while with probability $1 - p_n$, it remains δ. Hence, the new expected degree is $\delta' = p_n \times (\delta + 1) + (1 - p_n) \times \delta = p_n + \delta = \sum_{i=1}^{n} p_i$.

Hence, the expected degree of vertex v_1 in the example of Figure 4.1 is $\delta_{v_1} = 0.8 + 0.7 + 0.6 = 2.1$.

4.2.2 Neighbor Label Probabilities

For each vertex in $\mathcal{G}$, we also compute the probability that a particular label would occur in its neighborhood. This is used during the querying phase and, hence, ChiSeL performs the computation during the offline pre-processing phase for faster querying.

Consider vertex v to have n_v neighbors with corresponding labels $l_1, l_2, \cdots, l_{n_v}$, each connected by edges with existential probabilities $p_1, p_2, \cdots, p_{n_v}$ respectively. Note that vertex labels may not be unique and may repeat.

Assume label l_j to be associated with ψ neighbors of v, which are connected via edges with probabilities $p_1^{l_j}, p_2^{l_j}, \cdots, p_\psi^{l_j}$. The event that *no instance* of label l_j is present in the neighborhood of v occurs when none of these vertices are neighbors of v, i.e., these edges do not exist. Denoting the number of instances of label l_j in the neighborhood by $\#l_j$, the probability of this event is

$$P(\#l_j = 0) = (1 - p_1^{l_j}) \times \cdots \times (1 - p_\psi^{l_j}) = \prod_{i=1}^{\psi} \bar{p}_i^{l_j} \tag{4.2}$$

where $\bar{p}_i^{l_j}$ denotes the probability of the edge between the i^{th} neighbor and v not existing. Using the above equation, the probability that label l_j occurs *at least once* in the neighborhood of v is,

$$P(\#l_j \geq 1) = 1 - P(\#l_j = 0) \tag{4.3}$$

Similarly, the probability that label l_j occurs *exactly once* in the neighborhood of v is given by

$$P(\#l_j = 1) = \sum_{i=1}^{\psi} \left[p_i^{l_j} \cdot \prod_{\iota \neq i} \bar{p}_\iota^{l_j} \right] = P(\#l_j = 0) \cdot \sum_{i=1}^{\psi} \frac{p_i^{l_j}}{\bar{p}_i^{l_j}} \tag{4.4}$$

Possible worlds	Neighbors of v_1(A) v_2(B)	v_3(C)	v_4(A)	$P(W_x)$	Best matches for query triplets of q_1(A) and symbols $\langle B, A, C\rangle$	Sym.	$\langle C, A, D\rangle$	Sym.	$\langle B, A, D\rangle$	Sym.
W_0	0	0	0	$\overline{0.7} \times \overline{0.6} \times \overline{0.8} = 0.024$	$\langle \phi, A, \phi \rangle$	s_0	$\langle \phi, A, \phi \rangle$	s_0	$\langle \phi, A, \phi \rangle$	s_0
W_1	0	0	1	$\overline{0.7} \times \overline{0.6} \times 0.8 = 0.096$	$\langle \phi, A, \phi \rangle$	s_0	$\langle \phi, A, \phi \rangle$	s_0	$\langle \phi, A, \phi \rangle$	s_0
W_2	0	1	0	$\overline{0.7} \times 0.6 \times \overline{0.8} = 0.036$	$\langle C, A, \phi \rangle$	s_1	$\langle C, A, \phi \rangle$	s_1	$\langle \phi, A, \phi \rangle$	s_0
W_3	0	1	1	$\overline{0.7} \times 0.6 \times 0.8 = 0.144$	$\langle C, A, \phi \rangle$	s_1	$\langle C, A, \phi \rangle$	s_1	$\langle \phi, A, \phi \rangle$	s_0
W_4	1	0	0	$0.7 \times \overline{0.6} \times \overline{0.8} = 0.056$	$\langle B, A, \phi \rangle$	s_1	$\langle \phi, A, \phi \rangle$	s_0	$\langle B, A, \phi \rangle$	s_1
W_5	1	0	1	$0.7 \times \overline{0.6} \times 0.8 = 0.224$	$\langle B, A, \phi \rangle$	s_1	$\langle \phi, A, \phi \rangle$	s_0	$\langle B, A, \phi \rangle$	s_1
W_6	1	1	0	$0.7 \times 0.6 \times \overline{0.8} = 0.084$	$\langle B, A, C \rangle$	s_2	$\langle C, A, \phi \rangle$	s_1	$\langle B, A, \phi \rangle$	s_1
W_7	1	1	1	$0.7 \times 0.6 \times 0.8 = 0.336$	$\langle B, A, C \rangle$	s_2	$\langle C, A, \phi \rangle$	s_1	$\langle B, A, \phi \rangle$	s_1
Distribution of $[s_0, s_1, s_2]$ for each query triplet					$[0.12, 0.46, 0.42]$		$[0.40, 0.60, 0.00]$		$[0.30, 0.70, 0.00]$	

Table 4.1: Computing observed values of symbols s_0, s_1 and s_2 for vertex pair $\langle q_1, v_1 \rangle$ for the example in Figure 4.1.

If there is no neighbor of v with the label l_j, then it cannot occur in the neighborhood of v and, consequently, $P(\#l_j = 0) = 1$ and $P(\#l_j \geq 1) = P(\#l_j = 1) = 0$.

The occurrence probabilities of labels in the neighborhood of a vertex is computed using equations 4.2 - 4.4 during pre-processing.

4.3 Querying Phase

After the indexing phase is completed, the *querying phase* commences on the arrival of a query graph $\mathcal{Q} = (\mathcal{V}_\mathcal{Q}, \mathcal{E}_\mathcal{Q}, \mathcal{L}_\mathcal{Q})$. We next describe the details of the steps involved in this phase.

4.3.1 Inverted Lists and Neighborhood Information

Similar to the input target graph $\mathcal{G}$, as described in Section 4.2, the vertex-label inverted index and neighbor label list structures are computed for the query graph $\mathcal{Q}$ as well. Since query graphs are generally relatively small in size (in the order of tens of vertices), this step is fast.

4.3.2 Vertex Pair Generation

The querying phase proceeds by constructing similar matching vertex pairs between graphs $\mathcal{G}$ and $\mathcal{Q}$. Specifically, a *vertex pair*, $\langle q, v \rangle$, is constructed with vertex $q \in \mathcal{V}_\mathcal{Q}$ and $v \in \mathcal{V}_\mathcal{G}$ having the same label, i.e., $\mathcal{L}_\mathcal{Q}(q) = \mathcal{L}_\mathcal{G}(v)$. Given a query node q, such vertex pairs can be easily formed by using the vertex-label inverted index structure

of $\mathcal{G}$. Formally, the entire vertex pair set for query $\mathcal{Q}$ is,

$$\mathcal{VP} = \{\langle q, v\rangle \mid q \in \mathcal{V}_\mathcal{Q}, v \in \mathcal{V}_\mathcal{G}, \mathcal{L}_\mathcal{Q}(q) = \mathcal{L}_\mathcal{G}(v)\} \tag{4.5}$$

The vertex pair set for the example in Figure 4.1 is $\{\langle v_1, q_1\rangle, \langle v_2, q_2\rangle, \langle v_3, q_3\rangle, \langle v_4, q_1\rangle\}$.

Vertex pairs provide initial seeds for neighborhood exploration for finding matching subgraphs. The ChiSeL framework works on the computation of statistical significance scores of these vertex pairs to obtain the top-k potential candidate regions in $\mathcal{G}$ for extracting the best (approximate) matching subgraph to $\mathcal{Q}$. The next section explains how the χ^2-value of a vertex pair is computed.

4.4 Vertex Pair Chi-Square Computation

4.4.1 Label Triplet Generation

For each query node q, ChiSeL initially constructs *triplets* of the form $\langle x, q, y\rangle$ where x and y are the neighboring nodes of q in $\mathcal{Q}$. The corresponding label triplet, $\langle l_x, l_q, l_y\rangle$, is computed, where l_x, l_q, and l_y denote the labels of the vertices x, q and y respectively in $\mathcal{Q}$, i.e., $l_x = \mathcal{L}_\mathcal{Q}(x), l_q = \mathcal{L}_\mathcal{Q}(q)$, and $l_y = \mathcal{L}_\mathcal{Q}(y)$. The neighbors are considered to be symmetric, i.e., $\langle l_x, l_q, l_y\rangle$ is equivalent to $\langle l_y, l_q, l_x\rangle$. Therefore, without loss of generality, we order the labels in a label triplet alphabetically.

Considering vertex $v \in \mathcal{G}$ that forms the vertex pair $\langle q, v\rangle$ with q, similar neighbor triplets $\langle u, v, w\rangle$ and their corresponding label triplets $\langle l_u, l_v, l_w\rangle$ of v are extracted ($l_u = \mathcal{L}_\mathcal{G}(u), l_v = \mathcal{L}_\mathcal{G}(v)$, and $l_w = \mathcal{L}_\mathcal{G}(w)$).

4.4.2 Triplet Pair Matching

For the vertex pair $\langle q, v\rangle$, ChiSeL next characterizes the similarity between the label triplets obtained from q and v. By definition of vertex pair construction, we have $l_q = l_v$. However, the other two neighboring vertex labels (in the triplets) may or may not match. Based on the degree of label matching between the obtained label

triplets (of a vertex pair), i.e., label triplet pairs $\langle l_x, l_q, l_y \rangle$ and $\langle l_u, l_v, l_w \rangle$, the *triplet pair similarity* is characterized into three different classes:

- s_2: When both the neighboring labels in the triplets match.

$$s_2 : (l_x = l_u \wedge l_y = l_w) \tag{4.6}$$

- s_1: When only one of the neighboring triplet labels matches.

$$s_1 : ((l_x = l_u \wedge l_y \neq l_w) \vee (l_x \neq l_u \wedge l_y = l_w)) \tag{4.7}$$

- s_0: When none of the neighboring labels match.

$$s_0 : (l_x \neq l_u \wedge l_y \neq l_w) \tag{4.8}$$

Since a vertex in the target graph has neighbors connected by probabilistic edges, it will have a varying number of triplets in different "possible worlds". We next explain how the above triplet pair matching assigns symbols in this probabilistic setting.

4.4.3 Possible Worlds

We adopt the *possible world semantics* (PWS) graph uncertainty model, where each vertex v with d neighbors is connected by edges with probabilities of existence. The existence of edges is considered to be independent of each other. Thus, in each possible world, either an edge exists or it does not exist. There are 2^d possible worlds, and each such possible world occurs with a particular probability (as shown in Table 4.1).

4.4.4 Observed Match Symbol Vector

Assume vertices $t_1, t_2, \cdots, t_d$ to be the neighbors of vertex v, with corresponding labels $l_1, l_2, \cdots, l_d$ respectively. Consider a particular possible world W_i where edges to neighbors $t_1, t_2, \ldots, t_w$ exist but those to $t_{w+1}, t_{w+2}, \cdots, t_d$ do not exist, for some $1 \leq w \leq d$. The probability of world W_i is, thus,

$$P(W_i) = p_1 \times \cdots \times p_w \times \bar{p}_{w+1} \times \cdots \times \bar{p}_d \tag{4.9}$$

The vertex triplets of v that exist in W_i are $\mathcal{T}_i = \{\langle l_1, l_v, l_2\rangle, \langle l_1, l_v, l_3\rangle, \cdots, \langle l_{w-1}, l_v, l_w\rangle\}$.

Assuming the vertex pair $\langle q, v\rangle$, the query triplet $\langle l_x, l_q, l_y\rangle$ is now matched with all the possible triplets in $\mathcal{T}_i$. The matching of a query triplet is considered to be the one that produces the *best* scenario, i.e., where there are more triplet label matches. The order of preference of match classes (or symbols) is, thus, $s_2 \succ s_1 \succ s_0$.

For each possible world W_i, a match symbol s_j (either s_2, s_1, or s_0) is, hence, associated with the query triplet, denoted as $s_j \odot W_i$. The overall probability of a particular match symbol is the *sum* of probabilities of the possible worlds to which it gets associated:

$$P(s_j) = \sum_{\forall W_i,\ s_j \odot W_i} P(W_i) \tag{4.10}$$

Thus, corresponding to each query triplet, and a vertex pair, there is a probability distribution of the symbols s_2, s_1 and s_0. This forms the *observed counts* for the match symbols that implicitly incorporate the PWS concept based on different possible worlds.

Based on the above formulation, Table 4.1 shows the observed symbol vectors for query triplets corresponding to the vertex pair $\langle q_1, v_1\rangle$ with label A. Since v_1 has three neighbors, the number of possible worlds is $2^3 = 8$. They are denoted by $W_0, \ldots, W_7$.

Consider the possible world W_1 where only neighbor v_4 with label A occurs but neighbors v_2 and v_3 do not occur. The probability of this world is $P(W_1) = (1 - 0.7) \times (1 - 0.6) \times 0.8 = 0.096$. The label triplets around v_1 are $\langle A, A, \phi \rangle$ and $\langle \phi, A, \phi \rangle$ where ϕ is a dummy label that is used when at most one neighbor label exists. Comparing any label with ϕ, including ϕ itself results in a mismatch. For the query triplet $\langle B, A, C \rangle$, the best match, therefore, is $\langle \phi, A, \phi \rangle$. Since none of the neighbors match, this results in the symbol s_0. Similarly, for the world W_2, the query triplet $\langle B, A, C \rangle$ is best matched with the vertex triplet $\langle C, A, \phi \rangle$ yielding the symbol s_1. In the world W_7, the possible vertex triplets are $\langle A, A, B \rangle$, $\langle A, A, C \rangle$ and $\langle B, A, C \rangle$. The highest match corresponds to $\langle B, A, C \rangle$ and it yields the symbol s_2. Thus, the world W_7 contributes to the symbol s_2.

Adding up the probabilities of the worlds in which the symbols occur, the observed symbol vector for the query triplet $\langle B, A, C \rangle$ is $[0.12, 0.46, 0.42]$. Since the probabilities of the possible worlds add up to 1, so do the probabilities of the symbols for a query triplet.

4.4.5 Multiple Query Triplets

When there are multiple query triplets corresponding to a vertex pair, which is generally the case, the observed counts of the match symbols are added to form the cumulative *observed count vector*, O, for the vertex pair. In Table 4.1, considering all the three query triplets pertaining to query node q_1, the cumulative observed symbol vector obtained for the vertex pair $\langle q_1, v_1 \rangle$ is $O_{\langle q_1, v_1 \rangle} = [0.82, 1.76, 0.42]$.

ChiSeL considers the matching symbols of each of the query triplets to find the best matching subgraph.

4.4.6 Efficiently Computing Observed Vectors

However, in the above process of computing the cumulative observed count vectors, enumerating all the possible worlds is computationally inefficient. For some vertices with large degrees, it is practically infeasible. Fortunately, since we are only con-

cerned with the presence or absence of relevant neighbor vertex labels in the worlds, ChiSeL efficiently computes the observed symbol vectors based on the probabilities obtained in the indexing phase.

Consider a query triplet $\langle l_x, l_q, l_y \rangle$ and a corresponding vertex v in $\mathcal{G}$ with label $l_v = l_q$. The vertex v can have multiple neighbors with the label l_x (or l_y). Equation 4.2 to Equation 4.4 in Section 4.2.2 give the probabilities of l_x (or l_y) occurring various number of times in the neighborhood.

We first consider the case when $l_x \neq l_y$. The symbol s_2 occurs in those worlds where both the labels l_x and l_y occur at least once. The occurrence of such an event has the probability

$$O_{\langle q,v \rangle}[s_2] = P(\#l_x \geq 1) \times P(\#l_y \geq 1) \quad \text{[using Eq. 4.3]} \tag{4.11}$$

Similarly, symbol s_0 occurs when none of the instances of the labels l_x and l_y occur:

$$O_{\langle q,v \rangle}[s_0] = P(\#l_x = 0) \times P(\#l_y = 0) \quad \text{[using Eq. 4.2]} \tag{4.12}$$

Since the probability of the match symbols add up to 1,

$$O_{\langle q,v \rangle}[s_1] = 1 - O_{\langle q,v \rangle}[s_2] - O_{\langle q,v \rangle}[s_1] \tag{4.13}$$

We next consider the case when $l_x = l_y$. The symbol s_0 occurs in those worlds where no instance of label l_x occurs:

$$O_{\langle q,v \rangle}[s_0] = P(\#l_x = 0) \quad \text{[using Eq. 4.2]} \tag{4.14}$$

Similarly, symbol s_1 occurs when exactly one instance of l_x occurs:

$$O_{\langle q,v \rangle}[s_1] = P(\#l_x = 1) \quad \text{[using Eq. 4.4]} \tag{4.15}$$

Consequently,

$$O_{\langle q,v\rangle}[s_2] = 1 - O_{\langle q,v\rangle}[s_0] - O_{\langle q,v\rangle}[s_1] \tag{4.16}$$

The above computation avoids the exponential enumeration of the possible worlds and is only *linear* in terms of the number of neighbors of a vertex. More importantly, equations 4.2-4.4 are computed in the *offline* phase before any query arrives. The querying phase only uses the information and is, therefore, very fast in practice.

4.4.7 Expected Symbol Vector for Triplet Match

The chi-square statistic computes the deviation of the observations from the expectations. The expected count of triplet match symbols for a vertex triplet is computed as follows.

Suppose the input target graph contains $|\Gamma|$ labels that are assumed to be equally likely in terms of occurrence. Consider a vertex v with *expected degree* δ_v (as computed in Section 4.2.1). The chance that a neighbor of v has a particular label l_x is $1/|\Gamma|$. Therefore, the chance that none of the neighbors of v has label l_x is given by

$$\bar{p}_v = \left(1 - \frac{1}{|\Gamma|}\right)^{\delta_v} \tag{4.17}$$

.

Now, considering a label triplet of v, if the highest match symbol (to a query label triplet) is s_0, then none of the triplets have label l_x. Since there are two neighboring vertices in a triplet, assuming the independence of labels, the probability of the event s_0 for v is,

$$P_v(s_0) = \bar{p}_v^2 = \left(\left(1 - \frac{1}{|\Gamma|}\right)^{\delta_v}\right)^2 \tag{4.18}$$

The chance that there is at least one neighbor with label l_x is $(1 - \bar{p})$. Thus, the

probability that v has a triplet where both the neighboring vertices match is given by,

$$P_v(s_2) = (1 - \bar{p}_v)^2 = \left(1 - \left(1 - \frac{1}{|\Gamma|}\right)^{\delta_v}\right)^2 \tag{4.19}$$

The event s_1 occurs when there is one vertex in a triplet that matches a label while the other does not. Since there are two ways of enumerating vertex labels in a triplet, the probability of the event s_1 is,

$$P_v(s_1) = 2 \cdot \bar{p}_v \cdot (1 - \bar{p}_v) \tag{4.20}$$

$$= 2 \cdot \left(1 - \frac{1}{|\Gamma|}\right)^{\delta_v} \cdot \left(1 - \left(1 - \frac{1}{|\Gamma|}\right)^{\delta_v}\right) \tag{4.21}$$

Observe that the match symbols are mutually exhaustive, i.e., the probabilities add up to 1. If a query node q has len_q triplets, the expected counts of the match symbols ($i = 0, 1, 2$) for the triplet $\langle q, v \rangle$ are,

$$E_{\langle q,v \rangle}[s_i] = len_q \cdot P_v(s_i) \tag{4.22}$$

The expected counts of the match symbols are represented as a vector, referred to as the *expected symbol vector*. Importantly, the calculation of the above probabilities is *independent* of the query (except for Equation 4.22). Hence, they are done in the *offline* pre-processing phase of ChiSeL, contributing to the efficiency of the querying phase.

The vertex v_1 in Figure 4.1, as shown earlier in Section 4.2.1, has an expected degree of 2.1. Assume the total number of labels in $\mathcal{L}$ to be $L = 4$. Hence, $P(s_0) = ((1 - 1/4)^{2.1})^2 = 0.30$, $P(s_1) = 0.49$, and $P(s_2) = 0.21$. Thus, the expected symbol vector for the vertex pair $\langle q_1, v_1 \rangle$ having three query triplets is computed as $E = 3 \cdot [0.30, 0.49, 0.21] = [0.90, 1.47, 0.63]$.

4.4.8 Chi-Square of a Vertex Pair

Using the *observed symbol vector,* $O_{\langle q,v\rangle}[s_i]$ and the *expected symbol vector,* $E_{\langle q,v\rangle}[s_i]$ as computed above, ChiSeL finally computes the *chi-square* value, $\chi^2_{\langle q,v\rangle}$, of each candidate vertex pair $\langle q, v\rangle$ as:

$$\chi^2_{\langle q,v\rangle} = \sum_{i=0}^{2} \frac{(O_{\langle q,v\rangle}[s_i] - E_{\langle q,v\rangle}[s_i])^2}{E_{\langle q,v\rangle}[s_i]} \tag{4.23}$$

In the example in Figure 4.1, the chi-square of the vertex pair $\langle q_1, v_1\rangle$ is,

$$\chi^2_{\langle q_1,v_1\rangle} = \frac{(0.82 - 0.90)^2}{0.90} + \frac{(1.76 - 1.47)^2}{1.47} + \frac{(0.42 - 0.63)^2}{0.63} = 0.13$$

.

4.5 Top-k Subgraph Search

ChiSeL next computes the χ^2 scores for each of the vertex pairs obtained from the query Q. The best candidate regions for subgraph matching are then explored to obtain the final answer by use of two heap structures as discussed next.

4.5.1 Primary Heap

All the vertex pairs along with their χ^2 values are inserted into a *max-priority heap* called the *primary heap.* To compute the similarity for subgraph matching, vertex pairs are picked up from the primary heap in the priority order, i.e., the one with the largest chi-square score is picked first, and so on, to form the *seed* of the neighborhood search process. A vertex that has been matched earlier is not picked up any further. This ensures that the subgraphs returned are disjoint. This is done to avoid duplicate answers.

4.5.2 Secondary Heap

For each vertex pair picked from the primary heap, its neighborhood is searched to see if the query subgraph can be completed. Suppose, $\langle q, v \rangle$ is chosen. The neighbors of vertices v and q that share the same label (i.e., form a vertex pair themselves) are extracted, constructed into a vertex pair, and inserted into another priority-max heap, referred to as the *secondary heap*.

Assume a neighbor vertex pair $\langle n_q, n_v \rangle$. In addition to its chi-square value, $\chi^2_{\langle n_q, n_v \rangle}$, the probability, $p(v, n_v)$, of the edge connecting v to n_v is also inserted in the secondary heap. The neighbor vertex pair having the maximum value of $p(v, n_v) \cdot \chi^2_{\langle n_q, n_v \rangle}$ is then picked and the neighborhood search continues using this as the new seed. The edge probability is used along with the chi-square so that more probable edges are preferred.

4.5.3 Top-k Search

The growth of the neighborhood continues till the query is completely matched or the subgraph in the target graph cannot grow any further due to the lack of matching vertex pairs.

The above process is initiated k times to find the top-k matching subgraphs. The subgraphs are constrained to be disjoint from each other, by marking vertex pairs as "visited" in both the heaps.

4.6 Summation of Chi-Square Values

For every matching vertex pair of a query, the chi-square value follows the χ^2 *distribution* with two *degrees of freedom* (since there are three possible match symbols). The χ^2 values for the vertices of a matching subgraph are added to produce the total chi-square statistical significance score of the match. Since the addition of chi-square distributions results in another chi-square distribution (Hall, 1983), for a query graph $\mathcal{Q}$ of size $|\mathcal{V}_\mathcal{Q}|$, the total chi-square value of the matching subgraph

follows the chi-square distribution with degrees of freedom $2 \cdot |\mathcal{V}_\mathcal{Q}|$. This enables approximating the *p-value* of the match if required. The top-k matching subgraphs are sorted based on their total χ^2 values.

On one hand, the observed match symbol vector models the PWS concept by considering the match symbol (of triplets) across all the "possible" worlds. On the other hand, the χ^2 computation and the greedy neighborhood search takes into account the structural and label match. Thus, our proposed framework provides an integrated measure to obtain the best approximate subgraph matches.

4.7 Complexity Analysis

Assume that $\mathcal{G}$ has n vertices and m edges while $\mathcal{Q}$ has r nodes and s edges. Building the query graph indexes require $O(s)$, while creating the vertex pairs $\mathcal{VP}$ require at most $n \cdot r = O(n)$ time, since r and s are small and considered to be constants in practice. For each vertex pair in $\mathcal{VP}$, computing the expected vector requires $O(1)$ time, while that for the observed vector requires $O(d_v)$ time, where d_v is the degree of vertex v. The complexity of the above operations for all vertex pairs is approximately $O(m)$.

Considering $m \leq n^2$, the average degree of vertices in $\mathcal{G}$ is $a = 2m/n \sim O(n)$. The size of the primary heap is bounded by $O(n)$. For a vertex in the primary heap, the secondary heap is initially populated by $O(a)$ neighboring vertex pairs. Subsequently, the best candidate from the secondary heap is extracted and its $O(a)$ neighboring vertex pairs are added to the heap. Observe, at most r such iterations are performed on the secondary heap to extract the matching subgraph, providing a total time complexity of $O(r \cdot a) \sim O(n)$ for the heap operations. Hence, for top-k subgraph matches, the time complexity of ChiSeL is $O(m + k \cdot n)$.

The size of the primary heap is $O(n)$, while each secondary heap can grow to a maximum of another $O(a) \sim O(n)$ neighbors. The extra space overhead, hence, is at most $O(n)$.

Empirical evaluation, however, shows that the sizes of primary and secondary

heaps are mostly small constants in general.

4.8 Experimental Evaluation

In this section, we empirically evaluate the efficacy of the proposed algorithm, and benchmark its performance against state-of-the-art competing algorithms on large real-life datasets.

4.8.1 Setup

All the algorithms were implemented in C++. The experiments were performed on an Intel(R) Xeon(R) 2.6GHz CPU E5-2697v3 processor with 504GB RAM running CentOS Linux 7.2.1511.

Datasets

We use the following datasets as our input target graphs:

- *STRING DB* or *PPI* v10.5 (`version-10-5.string-db.org`): a database of known and predicted protein-protein interactions created automatically by collecting information from various sources. We extract the COG mappings of proteins and their links. The proteins are considered as the vertex set with the orthologous groups as their labels, and the links form the edges. Importantly, each protein link is annotated with a confidence score (from 0 to 1) that represents how likely it is that the interaction exists. The total graph consists of around 7.6M vertices and 1.2B edges with around 200K unique vertex labels. This dataset is referred to as *PPI-complete* henceforth.

  - *PPI-small*: We also randomly extract a smaller graph from PPI-complete consisting of 12K vertices and 10.7M edges with around 2.4K unique vertex labels.

Dataset	# Vertices	# Edges	# Labels	Avg. Degree
PPI-complete	7.6M	1.2B	0.2M	316
PPI-small	12.0K	10.7M	2.4K	1789
YAGO	4.3M	11.5M	4.0M	5
IMDb	3.0M	11.0M	3.0M	7

Table 4.2: Characteristics of datasets used.

- *YAGO* (`www.yago-knowledge.org`): an open-source knowledge graph consisting of extracted entities and relations from Wikipedia, WordNet and GeoNames. Each relationship is associated with a confidence value. It comprises of nearly 4.3M vertices and 11.5M edges, with 4M unique labels.

- *IMDb* (`www.imdb.com/interfaces`): dataset with information on movies, actors, directors, etc., with around 3M uniquely labeled vertices and 11M edges. We randomly assign edge probabilities to model scenarios where the edge existential probability distribution is not known apriori.

Table 4.2 summarizes the characteristics of the various datasets.

Query Generation

Exact. The query graphs were constructed from each of the above datasets by initially selecting a random vertex and then exploring its neighborhood using a random walk till q vertices are visited. Finally, the subgraph induced by the visited vertices is considered as the query Q. Without loss of generality, we consider the query graphs to be connected; else, the algorithms can be independently executed on the disjoint components.

We generated query graphs of different sizes, with the number of vertices varying from 3 to 13, with a step size of 2. For PPI-complete, the query graph size varied from 3 to 25. The number of query edges varied from 2 to 80. Further, for each query graph size, 20 different graphs were randomly extracted from each of the datasets. We refer to this set of query graphs as *exact* queries.

Noisy. To study the performance of the algorithms in the presence of noise, we further create a *noisy* query graph set by introducing structural noise as follows.

For each of the above exact queries obtained, we randomly insert, delete or modify the probabilities of some edges such that the number of edit operations is 33% of the edges present in the exact query.

Thus, for each dataset, in general, we considered $(6 \times 20 \times 2) = 240$ queries. For PPI-complete, the number of queries were $(12 \times 20 \times 2) = 480$. Unless otherwise mentioned, results presented henceforth are averages over the corresponding query sets.

Competing Methods

Two major strategies are used in the literature: distance/error bounded pruning approaches and tree/graph traversal based methodologies. Thus, in the same spirit, we compare the ChiSeL algorithm against two recent state-of-the-art methods:

(i) *PBound* (Gu et al., 2016), that performs maximal subgraph matching by incrementally computing similarity probabilities, and using probability upper bounds for pruning.

(ii) *Fuzzy* (Li et al., 2019a), that uses path-based graph decomposition and k-partite based joining techniques to obtain the best matching approximate subgraph on the LUBM benchmark, and has been shown to outperform SPath (Zhao and Han, 2010) and SAPPER (Zhang et al., 2010).

Parameter Setting

For the above competing methods, we performed the best-effort implementation as the original code was not available publicly or from the authors.

We experimentally studied the performance of the baselines with different parameter settings. Fuzzy was tested on varying edit-distance parameter combinations from the set {0, 0.25, 0.5, 0.75, 1.0}, and different distance and probability threshold parameters for PBound. For the best performance, the parameters of PBound were set to their default values (as reported in (Gu et al., 2016)), while in Fuzzy

Method	YAGO		IMDb		PPI-small	
	Acc.	**Time (s)**	**Acc.**	**Time (s)**	**Acc.**	**Time (s)**
ChiSeL	0.89	1.62	0.87	0.05	0.96	16.69
PBound	0.26	560.92	0.57	101.95	0.29	3134.09
Fuzzy	0.61	1.82	0.62	2.19	0.01	13970.94

Table 4.3: Performance comparison of the algorithms averaged over both exact and noisy query graph sets.(The results for PPI-complete are not shown since PBound and Fuzzy could not be run on them. ChiSeL achieves an accuracy of **0.84** on PPI-complete with a running time of **0.14**s.)

the probability threshold was set to 0.1 with the insertion, deletion and replacement costs for string edit-distance set to 1.0 each.

We set $k = 10$ for the number of top-k subgraphs returned in our framework.

Evaluation Metrics

We evaluate the quality of the matching subgraphs reported and the performance of the algorithms using the following measures:

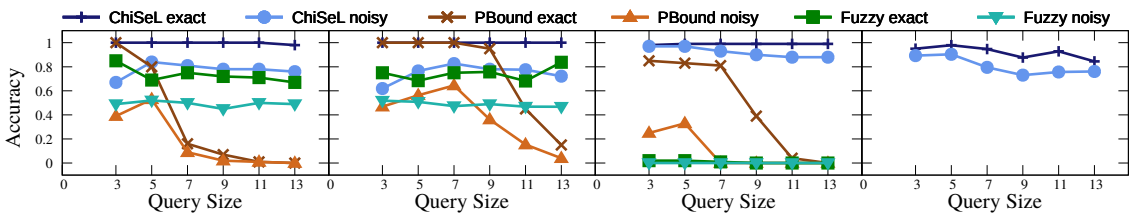
(i) *Mean maximum accuracy* – reports the average over the maximum accuracy of a subgraph match found, for each query, within the top-k results reported. We define the *accuracy* as the number of matching edges present in the answer subgraph against the number of edges required for a complete match (i.e., edges in the query).

(ii) *Runtime* – compares the computation efficiency of the algorithms by reporting the wall clock running times.

4.8.2 Results

The overall performances of the algorithms on the different datasets are tabulated in Table 4.3.

PBound enumerates all minimum spanning trees and Fuzzy lists all source-destination paths. Since these numbers are extremely large for PPI-complete that contains more than a billion edges, none of the queries for PBound and Fuzzy could



(a) YAGO (b) IMDb (c) PPI-small (d) PPI-complete

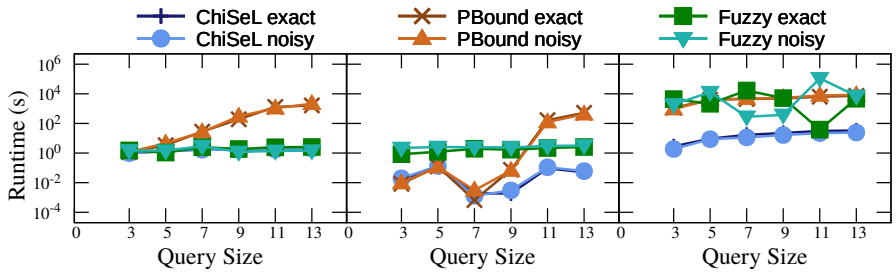
Figure 4.2: Accuracy comparison of different algorithms over all datasets.

finish within a reasonable amount of time (1 hour) for PPI-complete. ChiSeL could easily scale to such massive graphs along with high accuracy and achieves an accuracy of 0.84 on PPI-complete with a running time of 0.14s. For a fair evaluation of all three algorithms, a smaller dataset, PPI-small, was, therefore, randomly extracted from PPI-complete.

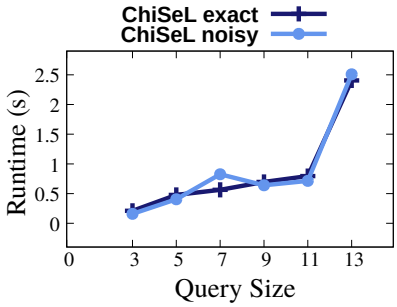
In terms of runtime, on all the datasets, ChiSeL is the fastest. PBound particularly suffers in compute time as it iterates over all possible minimum spanning trees. While running time for Fuzzy is low owing to the enumeration of all the source-destination paths, it requires a very high indexing time. ChiSeL requires considerably more running time for YAGO due to the fact that the vertex labels in YAGO are very long and can have special characters and, therefore, label matching requires a lot of time. Interestingly, ChiSeL took the longest time for PPI-small, almost 15 times more than even the total PPI-complete dataset. The reason was the very large average degree of vertices in PPI-small. To understand this effect further, we did more experiments on the average degree (see Section 4.8.5).

The quality of ChiSeL was also the best for all the datasets. While Fuzzy produced medium quality results for YAGO and IMDb, it completely failed for PPI-small. Fuzzy had too many paths to search since the number of labels were low and, hence, it ended up not finding the desired subgraphs for almost all the queries.

ChiSeL, thus, efficiently extracts subgraphs with better structural similarity (edge and label matches) to the query.



(a) YAGO (b) IMDb (c) PPI-small



(d) PPI-complete

Figure 4.3: Runtime comparison of different algorithms over all datasets.

4.8.3 Detailed Analysis

To further analyze the performance of the algorithms, we conducted experiments to capture the accuracy and runtime over different query types and query sizes. Figure 4.2 and Figure 4.3 report the accuracy and the runtime performance of all the algorithms.

ChiSeL was seen to perform the best in terms of quality, with nearly 100% accuracy for exact queries for almost all the query sizes for all the datasets except PPI-complete. The accuracy numbers were also observed to be stable across different query sizes, thereby demonstrating the robustness of our proposed framework. Even for the noisy query scenario, the accuracy was around 0.8 for most of the datasets and more than 0.9 for PPI-small.

PBound depicted a sharp drop in accuracy with an increase in query size. This is due to its inability to iterate over minimum spanning trees with distinct vertex sets within a reasonable runtime. Fuzzy showed a steady performance and was not affected much by the query size. It, however, performed very poorly for PPI-small.

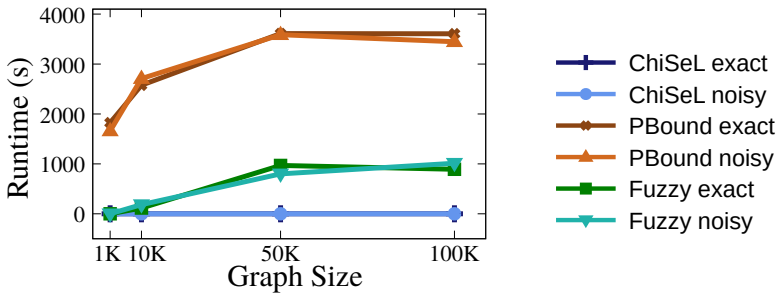


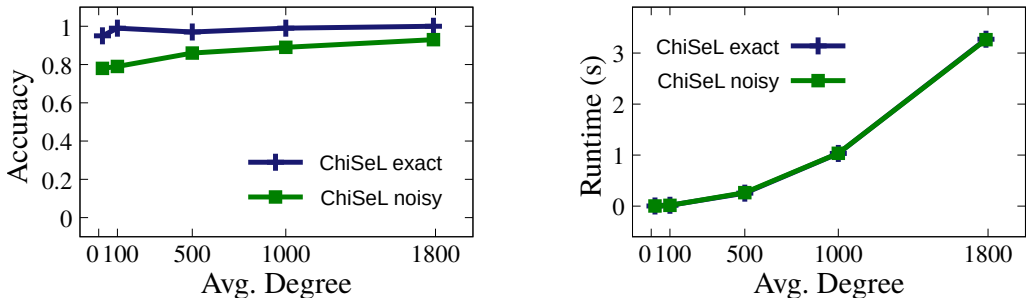
Figure 4.4: Scalability of ChiSeL, PBound and Fuzzy on graphs sampled from PPI-complete dataset.

Figure 4.3 shows that ChiSeL was the fastest for almost all the scenarios. Fuzzy quickly becomes impractical for larger query sizes and requires a very large time for query sizes 9 and above. Both PBound and Fuzzy performed very poorly on PPI-small due to the extremely high average degree. The number of paths and subgraphs are too many when the density of the graph is high, and, hence, these algorithms do not scale for such dense graphs.

An interesting anomalous behavior of runtime with query sizes was observed for IMDb and PPI-complete datasets for ChiSeL. This is explored and explained in detail later in Section 4.8.5.

4.8.4 Scalability Study

We next study the scalability of the different algorithms with respect to input graph size. For that, different-sized subsets from the PPI-complete dataset were created maintaining the average degree in each subset to be around 250. We then pose queries of size 5, for both exact and noisy scenarios, on these subsets. Figure 4.4 reports the running times observed on these sampled subgraphs. PBound performs poorly due to its enumeration of all minimum spanning trees. While for very small graphs, Fuzzy is comparable in runtime with ChiSeL, the overall scalability of ChiSeL is better. For graphs with a size greater than 50K, the runtime of ChiSeL is orders of magnitude lesser than the competing approaches. Overall, the scalability over graph size for all algorithms is at most linear.



(a) Accuracy (b) Running time

Figure 4.5: Effect of average degree.

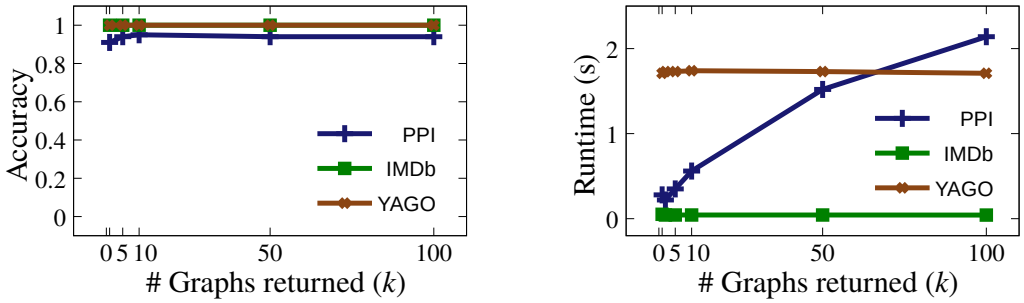
4.8.5 Analysis of Effect of Parameters

In this section, we empirically explore the robustness of performance for our proposed ChiSeL algorithm. We vary different characteristics of the input and query graphs to study their impact on the accuracy and runtime of ChiSeL.

Average Graph Degree

We first analyze the effect of the average degree of vertices in the input graph over the performance of ChiSeL. To that end, we took the entire PPI-small dataset and created various edge subsets of it. We created different samples of decreasing density from it by randomly deleting edges. The number of unique labels were more or less conserved. Queries of size 5 were then posed on these graphs.

While we observed no appreciable effect on accuracy, the runtime increased considerably with an increase in average degree (Figure 4.5). This is due to the increase in the number of neighborhoods, affecting the time taken by ChiSeL to explore and populate the secondary heap. To understand it further, we measured the size of the secondary heap, averaged across different queries, for the different scenarios. We see that while the secondary heap size is only 9 when the average degree is 20, it increases steadily across the increasing average degree and reaches 543 when the average degree is 1789. Consequently, the running time increases considerably.



(a) Accuracy (b) Running time

Figure 4.6: Effect of the number of subgraphs returned.

Number of Subgraphs Returned, Top-k

The next set of experiments measures the effect of the number of subgraphs returned, i.e., the parameter k for the top-k search. Figure 4.6 shows that there is negligible effect of k on accuracy for all the datasets and on runtime for IMDb and YAGO datasets. The runtime increases only slightly for PPI-complete mainly because of the increase in the number of secondary heap initializations.

Perturbation of Edge Probabilities

Since edge probabilities model the uncertainty in structure for our problem setting, we next study how changes in edge existence probabilities affect the performance of ChiSeL. For this study, we use PPI-complete, albeit with a slight modification. The input graph edges that are present in the query are modified to be *certain*, i.e., the probabilities of those edges in the original graph are set to 1.0. We refer to this modified dataset as *perturbed*, while *unperturbed* denotes the original un-altered PPI-complete graph. Further, as before, we consider both the *exact* and *noisy* query scenarios. Figure 4.7 depicts the accuracy of ChiSeL with changes in the probability model of an input graph with varying query sizes.

The accuracy increases considerably for the *perturbed* version of PPI-complete. This intuitively follows from the *possible world scenario* concept. Since the exact matches to the query edges are modified to have probability 1.0, the exact query subgraph is extracted and returned by ChiSeL with a higher chi-square (at the

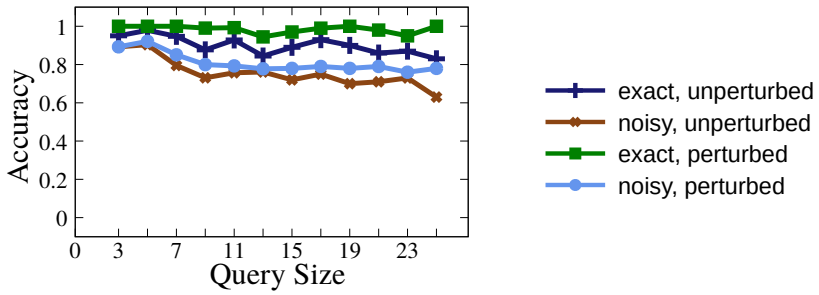


Figure 4.7: Effect of perturbation of edge probabilities.

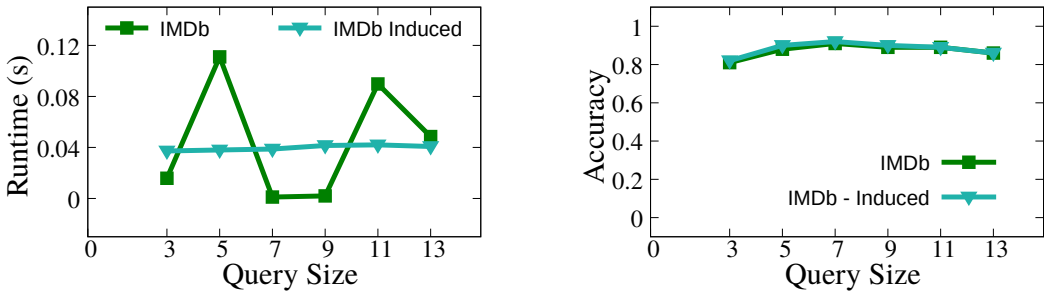
top of the ranking) in most cases, thereby leading to an increase in accuracy. This provides valuable insight that our proposed framework for subgraph matching based on statistical significance inherently takes into account the possible world modeling.

Possible World Semantics Modeling

To understand the effects of sampling possible worlds, we did the following experiment. From the PPI-small dataset, we created various "certain" graphs by sampling the edges according to their existence probabilities. In other words, in each such sampled possible world, an edge is present with its corresponding existence probability. Each such sampled world, therefore, contains a subset of the original set of edges, but is *certain* in nature and no longer a probabilistic graph. For each sampled certain graph, we ran an approximate graph querying algorithm that works for certain graphs. We chose the NAGA algorithm (Dutta et al., 2017) since it reports fairly accurate results and works on the same principles of statistical significance.

For each scenario, we created several sampled possible world graphs, varying from 100 to 10000. We then ran NAGA for queries of size 5 on every possible world graph, and report the *best* accuracy obtained over any possible world graph. Despite creating as large as 10000 possible worlds, the best accuracy obtained over a query graph was only 0.33. On average, the best accuracy over all the queries was only 0.15.

Since the number of possible worlds for PPI-small is extremely large, it may be that 10000 samples were not enough. We, thus, chose a very small graph – the input graph $\mathcal{G}$ shown in the example in Figure 4.1. Since there are only 6 edges



(a) Running time (b) Accuracy

Figure 4.8: Effect of degree distribution over labels on runtime and accuracy for original and controlled degree distribution.

in it, the total number of possible worlds is only $2^6 = 64$. We sampled 20 possible world graphs ($\sim 31\%$) from it and ran the query $\mathcal{Q}$ in Figure 4.1 against the possible worlds using NAGA. For these samples, the best accuracy achieved by NAGA was only 0.50, while ChiSeL demonstrated an accuracy of 0.75. The best accuracy level of 0.75 was attained by NAGA only after sampling 41 worlds ($\sim 64\%$).

This shows that sampling the possible worlds and running an approximate graph matching algorithm that works only for certain graphs is not enough to obtain good results, and is neither effective nor scalable. ChiSeL successfully avoids this expensive sampling procedure by utilizing the possible world modeling directly in its framework to find good matches.

Degree Distribution over Labels

An interesting erratic effect was observed over different query sizes for runtime for both IMDb and PPI-complete datasets (as earlier pointed out in Section 4.8.3). As shown in Figure 4.8(a), the query processing time was highest for queries with 5 vertices while particularly low for the larger queries of sizes 7 and 9. On further analysis, it was observed that queries, consisting of vertices whose label-matching vertices in the input graph exhibit large degrees, adversely affected the matching subgraph computation time. Since the query sets for each query size were chosen *randomly*, the set for 5-vertex size had a particularly high number of such high-degree graphs than 7- and 9-vertex sets and, hence, the "anomaly".

Measures	PPI-complete	YAGO	IMDb
Time (s)	5616.28	66.71	60.16
Memory (GB)	440.00	8.40	7.00

Table 4.4: Indexing time and memory consumption of ChiSeL.

To confirm the above behavior and to alleviate the above occurrence, we sampled a query set from IMDb with similar vertex degree distribution. We refer to this query set as *IMDb-Induced*. The query set was chosen progressively as follows. First, random queries of size 13 were chosen. Then, queries of size 11 were chosen by considering subgraphs from this set, and so on. This ensures that the degree distribution of the different query sets does not vary widely. On this modified query set, the query processing time increases only slightly and smoothly across the query sizes (Figure 4.8(b)). This confirms the effect of query degree distribution on the runtime complexity of ChiSeL. Thus, it was the presence of such anomalous high-degree query vertices (in the randomly generated queries) that attributed to the erratic behavior at some points of Figure 4.3. Removing such anomalous vertices resulted in a smoother curve, as shown in Figure 4.8(a).

Figure 4.8(b) shows that there is no appreciable effect of query degree distribution on the accuracy of the algorithms. Similar results were observed for the PPI-complete dataset as well.

4.8.6 Indexing Requirements

Table 4.4 tabulates the indexing time and memory requirements of ChiSeL for the different datasets. Even for the very large PPI-complete dataset with more than a billion edges, the memory footprint is not very high and the indexing time is less than 2 hours. The smaller-sized YAGO and IMDb datasets require only a minute and less than 10GB of memory. This shows that ChiSeL is applicable for diverse applications using commodity hardware.

4.8.7 Real World Use Case: StringDB

In this section, we show the performance of ChiSeL in a real-life use case for example queries from String DB (`version10.string-db.org/cgi/input.pl`) containing synthetases and regulators connected to long-chain fatty acyl-CoA synthetase. We obtain the "most confident" gold annotated result for two different queries (having 11 and 16 vertices) obtained by varying the number of interactors. The result consists of the exact locations (in terms of vertex IDs) of the PPI-complete graph where this query structure is important (in terms of score). Detection of such sites is useful in identifying mutation regions for early detection of cancer and other diseases.

We compare the performance of ChiSeL and NAGA in extracting subgraphs similar to the queries. Figure 4.9 depicts the obtained matching subgraphs from the algorithms. (Node identifiers shown in the figure are protein names and are different from labels of orthologous groups that are queried.) ChiSeL demonstrates a high structural similarity for both the queries and is seen to outperform NAGA. The accuracy gap of ChiSeL is more for the larger query since the discriminative power from the probabilistic modeling and structural similarity via statistical significance increases when more the number of vertices and edges are involved. The inherent modeling of PWS by ChiSeL (by taking the edge existential probabilities into account) enables it to accurately identify the location of the query subgraphs that are most important in terms of the protein interactions. It needs to be noted that there are several occurrences of the same query subgraph based on labels and edges in the PPI-complete graph. However, the vertex IDs provided as part of the ground truth provide the exact list of vertices that are the most important. As shown in Figure 4.9, ChiSeL correctly finds most of them and outperforms NAGA. The average query processing time for the two queries for NAGA was 27.6s, while ChiSeL took only 0.31s.

Hence, it can be concluded that ChiSeL provides an effective and efficient algorithm for approximate subgraph matching in uncertain graphs for real-life use cases.

(a) Query

(b) ChiSeL

(c) NAGA

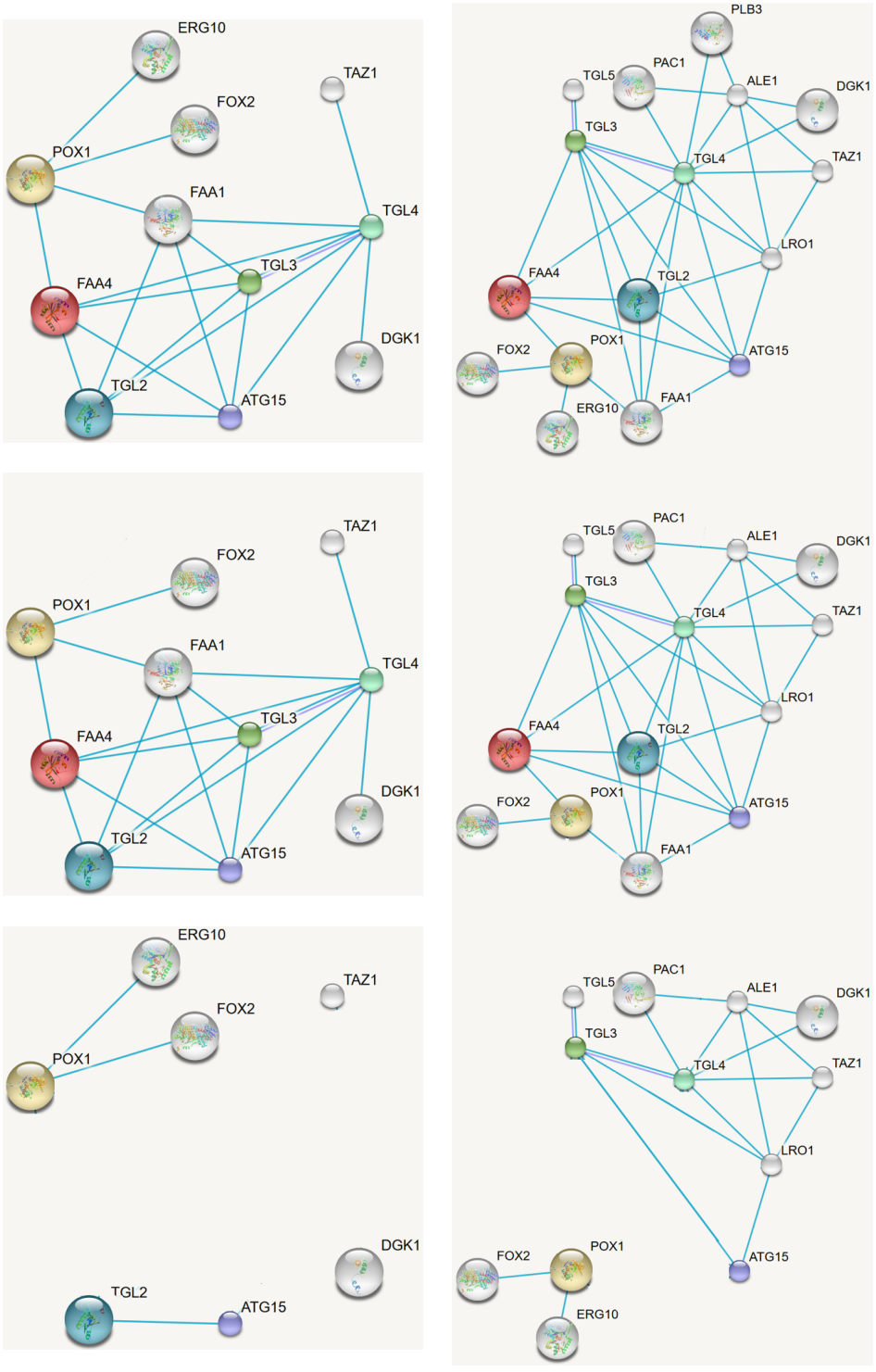


Figure 4.9: Evaluation on real dataset: (a) String DB queries, and corresponding results from (b) ChiSeL and (c) NAGA.

4.9 Discussion

We saw that the working of ChiSeL hinges on the principle of *statistical significance* and it handles the problem of approximate subgraph querying in large vertex-labeled edge-probabilistic graphs efficiently. Using it for a real-life use case showed that the results found were highly relevant and accurate. In this section, we briefly outline how other cases of uncertainty and noise can be handled within our framework.

Edge Labels

In a graph with labeled edges, for a vertex to match the corresponding edge labels also need to match. To handle this easily, for a neighboring vertex, we can prepend the corresponding edge label to the vertex label of the neighbor. This ensures that the edge labels are also taken into account during triplet matching and symbol generation.

Uncertain Vertices

Consider the vertices in the target input graph to be also uncertain, i.e., each vertex exists with a probability. This scenario can be handled in the current model by absorbing the vertex probabilities into the probabilities of the edges incident on it. Thus, if vertices u, v, w exist with probabilities p_u, p_v, p_w respectively, and the edges $e_1 = (u, v), e_2 = (w, v)$ exist with probabilities p_1, p_2 respectively, then the graph can be appropriately modified such that the edge probabilities are updated to $p'_1 = p_1 \cdot p_u \cdot p_v$ and $p'_2 = p_2 \cdot p_w \cdot p_v$. The vertices are then no longer treated as uncertain, and the rest of the framework remains the same.

Noisy Labels

For scenarios where the labels in the vertices might be noisy (e.g., misspelled), similarity measures such as *Jaccard similarity* can be used for matching purposes. For example, vertex labels having a Jaccard similarity score greater than a pre-

defined threshold will be considered to be a match. Also, *semantic similarity* of the vertex labels can be considered in such cases.

Label Uncertainty

The label of a vertex or an edge might also be uncertain, that is, there exists a probability distribution over a set of labels for each vertex or edge. For example, suppose vertex v has label l_{1_v} with probability 0.6 and label l_{2_v} with probability 0.4. Given a query vertex q with label l_q, a suitable distance function (such as Jensen-Shannon divergence measure) can be used to ascertain the similarity between the two label distributions. Similar to noisy labels, only if this distance is below a threshold, the vertices are said to match.

Uncertain Query Graphs

Finally, if the query graph is also uncertain (this uncertainty may be in the edges and/or vertices and/or labels), we can adopt deviation thresholds of probability for each uncertainty, i.e., only candidate matches having a probability deviation from the query vertex/edge matches within the threshold will be considered.

Chapter 5

VeNoM

Many *approximate subgraph matching (ASM)* algorithms (Khan et al., 2013; Dutta et al., 2017; Agarwal et al., 2021) have been developed. Different factors affect the algorithms, such as the degree of nodes, label distribution in the graph, missing edges, etc. One of the most important aspects is, perhaps, the size of the neighborhood considered when comparing two nodes. Despite the considerable amount of research done in approximate subgraph querying methods, there is still room for researchers to conduct a more in-depth study of the effect of these factors on the subgraph querying methods.

VeNoM (Vertex Neighbor Matching), which is an enhancement over VELSET (Dutta et al., 2017), a state-of-the-art ASM approach for deterministic graphs. To analyze the effect of the size of the neighborhood considered during matching, we instantiate four versions of VeNoM, namely, *VeNoM-(1,1)*, *VeNoM-(2,1)*, *VeNoM-(3,1)* and *VeNoM-(2,2)*. We elaborate on them in later sections. The experiments provide an understanding of the effects of different graph parameters on the performance of the individual instances. This would help in tuning and comparing the approximate subgraph matching algorithms, for example, the trade-off between a smaller (respectively, larger) running time and lesser (respectively, higher) matching accuracy. Extensive experiments show that accuracy improves as the depth of the neighborhood comparison is increased at the cost of a larger runtime. Experiments with benchmark algorithms, VELSET (Dutta et al., 2017) and VerSaChI (Agar-

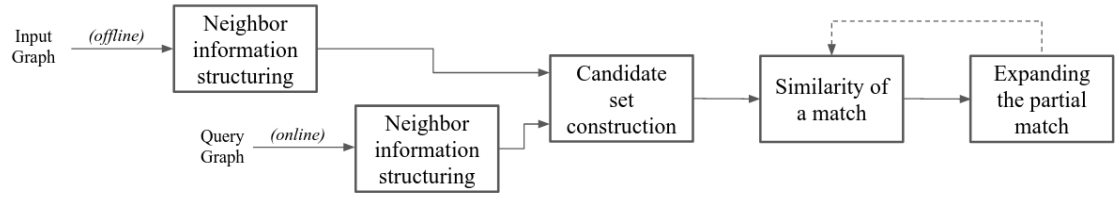


Figure 5.1: Generalised flowchart used in VeNoM

wal et al., 2021), show that a holistic approach, incorporating both the depth and breadth of the neighborhood has a better performance.

We first present a succinct summary of VELSET and outline the scenarios where its effectiveness is limited (Section 5.2). We next present our algorithm, VeNoM, which is a variant of VELSET, in Section 5.3. Different extension flavors, based on the depth and breadth of a neighborhood, are discussed in Sections 5.5- 5.7. Figure 5.1 shows an overview of VeNoM.

5.1 Notations

Consider a *target graph* $\mathcal{G}(\mathcal{V}_\mathcal{G}, \mathcal{E}_\mathcal{G}, \mathcal{L}_\mathcal{G})$, with vertex set $\mathcal{V}_\mathcal{G}$, set of edges $\mathcal{E}_\mathcal{G}$ and a function $\mathcal{L}_\mathcal{G} : \mathcal{V}_\mathcal{G} \rightarrow \Sigma_\mathcal{G}$ that assigns labels to vertices, where $\Sigma_\mathcal{G}$ is a finite set of labels. A *query graph* $\mathcal{Q}(\mathcal{V}_\mathcal{Q}, \mathcal{E}_\mathcal{Q}, \mathcal{L}_\mathcal{Q})$ is searched for in $\mathcal{G}$, where $\mathcal{V}_\mathcal{Q}$ and $\mathcal{E}_\mathcal{Q}$ represent the set of query vertices and edges, respectively, and $\mathcal{L}_\mathcal{Q}$ is the query vertex label function. We use the notation $\mathcal{N}_G(v)$ and $\mathcal{N}_Q(q)$ to denote the set of one-hop neighbors of $v \in \mathcal{V}_\mathcal{G}$ and $q \in \mathcal{V}_\mathcal{Q}$, respectively. Additionally, we use the notation $\langle q, v \rangle$ to denote a candidate vertex pair, where $q \in \mathcal{V}_\mathcal{Q}$ is a query vertex and $v \in \mathcal{V}_\mathcal{G}$ is a candidate match for q. A candidate match is any target vertex v that satisfies the condition $\mathcal{L}_\mathcal{Q}(q) \approx \mathcal{L}_\mathcal{G}(v)$, i.e., the label of the query vertex q is "similar" to that of v. The similarity is based on a user-defined similarity metric, which may be exact or approximate, e.g., Levenshtein distance, Jaccard similarity, etc.

5.2 Overview of VELSET

VELSET is an ASM framework for deterministic labeled graphs based on statistical analysis. To compute the similarity between two nodes, it compares the 1-hop neighborhood of the individual vertices. For this purpose, VELSET creates a set of *triplets* for each vertex of the target graph and query graph. A triplet for a vertex $q \in \mathcal{V}_\mathcal{Q}$ is of the form $\langle \mathcal{L}_\mathcal{Q}(q_i), \mathcal{L}_\mathcal{Q}(q), \mathcal{L}_\mathcal{Q}(q_j) \rangle$, where $q_i, q_j \in \mathcal{N}_Q(q)$. Triplets for target vertices are defined in the same way. A list of such triplets represents the label distribution in the neighborhood of the query and target vertices.

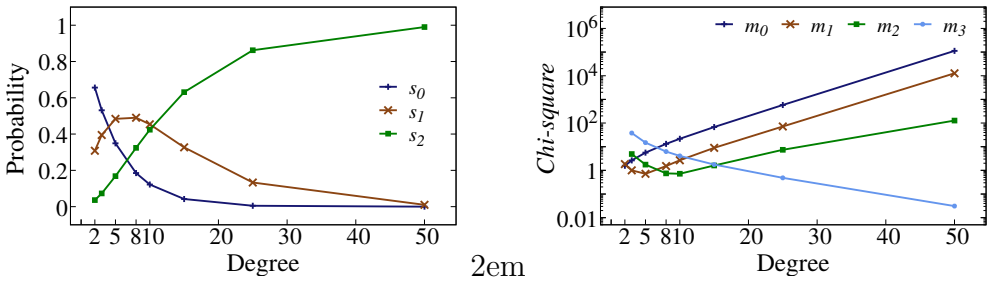
In VELSET, the candidate vertex pairs are formed based on the Jaccard similarity. To characterize the neighborhood similarity of a vertex pair $\langle q, v \rangle$, the triplets of q are matched against those of v. Triplet matches are categorized into three categories, denoting the quality or degree of the match: s_0, when no neighbor label is found to be similar; s_1, when only one of the neighbor labels is similar; and s_2, when both the neighbor labels are similar. The categories are ordered as $s_2 \succ s_1 \succ s_0$, i.e., s_2 is preferred over s_1, and s_1 is preferred over s_0.

To quantify the match similarity of a candidate pair, it uses the *Pearson's chi-squared statistical significance test*. The observed value for a vertex pair $\langle q, v \rangle$ is the count of the categorical matches obtained by matching triplets. To compute the expected value of the categorical matches, VELSET assumes uniform label distribution and argues that the probability of a label (from query triplet) not matching is $(1 - 1/L)^{d_v}$, where L is the number of unique labels and d_v is the degree of vertex $v \in \mathcal{V}_\mathcal{G}$. Based on this, the probability for each of the categories is defined as:

$$\begin{gathered} P_v(s_0) = \left((1 - 1/L)^{d_v}\right)^2; \quad P_v(s_1) = 2 \cdot (1 - 1/L)^{d_v} \cdot \left(1 - (1 - 1/L)^{d_v}\right); \\ P_v(s_2) = \left(1 - (1 - 1/L)^{d_v}\right)^2 \end{gathered} \tag{5.1}$$

The expected values are computed by scaling the probability values appropriately for a vertex pair and is used in chi-squared value computation to capture it similarity.

In the final step, the highest scoring candidate pair is greedily expanded until a



2em

(a) Probability trends of s_0, s_1 and s_2

(b) χ^2 values for match scenarios m_0 - m_3 for query degree 3

Figure 5.2: Expected value and χ^2 trends for VELSET for $L = 10$.

match is found or the search space of candidate neighbor pairs is exhausted. The readers can refer to the original paper (Dutta et al., 2017) for more details.

5.3 VeNoM-(2,1)

The expected value calculation in VELSET is influenced by two factors, the number of unique labels and the degree of target graph vertices. The intuition is that the expected value for the best match category would be very low and any vertex pair exhibiting a *perfect* triplet match would thus get a higher chi-square value. We verify this by putting different values of L and d_v in Equation 5.1. We also assume a query vertex q of degree 3 and enumerate four possible scenarios based on number of labels that find a match to analyze the chi-square trends (observed values of categories are denoted as $(\#s_0, \#s_1, \#s_2)$):

- m_0: when none of the neighbor labels match, $(3, 0, 0)$
- m_1: when exactly one neighbor label matches, $(1, 2, 0)$
- m_2: when exactly two of the neighbor labels match, $(0, 2, 1)$
- m_3: when all the neighbor labels match, $(0, 0, 3)$

Figure 5.2a shows the probability of different match categories for a fixed number of unique labels, $L = 10$ and, different degrees of the vertex. It is observed that $P_v(s_2) < P_v(s_0)$ for lower degrees of v and, for higher degrees of the vertex, the

expected values for the s_2 category were higher than for s_0. Due to this reason an inversion in the chi-square values can be seen in the chi-square values can be seen in Figure 5.2b. Note that m_3 is the best possible matching scenario for q, followed by m_2, m_1 and m_0 in that order. It can be seen that for the χ^2 for the best scenario decreases with increase in the input vertex degree. This essentially penalizes a good input vertex match for having a higher number of connections.

In general, the inversion is observed when the label set size is much smaller compared to the vertex degree (roughly, a degree to label ratio of > 0.65). To overcome this issue, we propose VeNoM, which modifies the expected value calculation by replacing the input vertex degree d_v in Equation 5.1 with the degree of the query vertex d_q, since in general, the query graphs are much smaller and have lower degrees. Thus, the probabilities of the match categories in VeNoM are computed as,

$$P_q(s_0) = (p_q)^2\,; \qquad P_q(s_1) = 2 \cdot p_q \cdot (1 - p_q)\,; \qquad P_q(s_2) = (1 - p_q)^2 \tag{5.2}$$

where, $p_q = (1 - 1/L)^{d_q}$.

For ease of discussing different extensions of VeNoM in this article, we define two terms, *unit* and *group*. Note that both the terms are defined with respect to a vertex.

Definition 5.1. Unit. *An ordered collection of neighbor labels of a vertex which form a path of length h.*

Definition 5.2. Group. *A set of k units of a vertex along with its label where the units correspond to different neighbors.*

The terms *unit* and *group* help us in extending the triplet format to extensions of VeNoM. As discussed previously, a triplet consists of *two* 1-hop neighbor labels of a vertex. In the proposed terminology, we refer to each of these neighbor labels as a *unit*. A triplet is then essentially a *group of two units* along with the label of the vertex. Hence, we denote the modified version of VELSET, as an instance VeNoM-(2,1), in the VeNoM framework. The first number in the bracket denotes

the total number of units in a group, while the second number denotes the number of vertices in a unit.

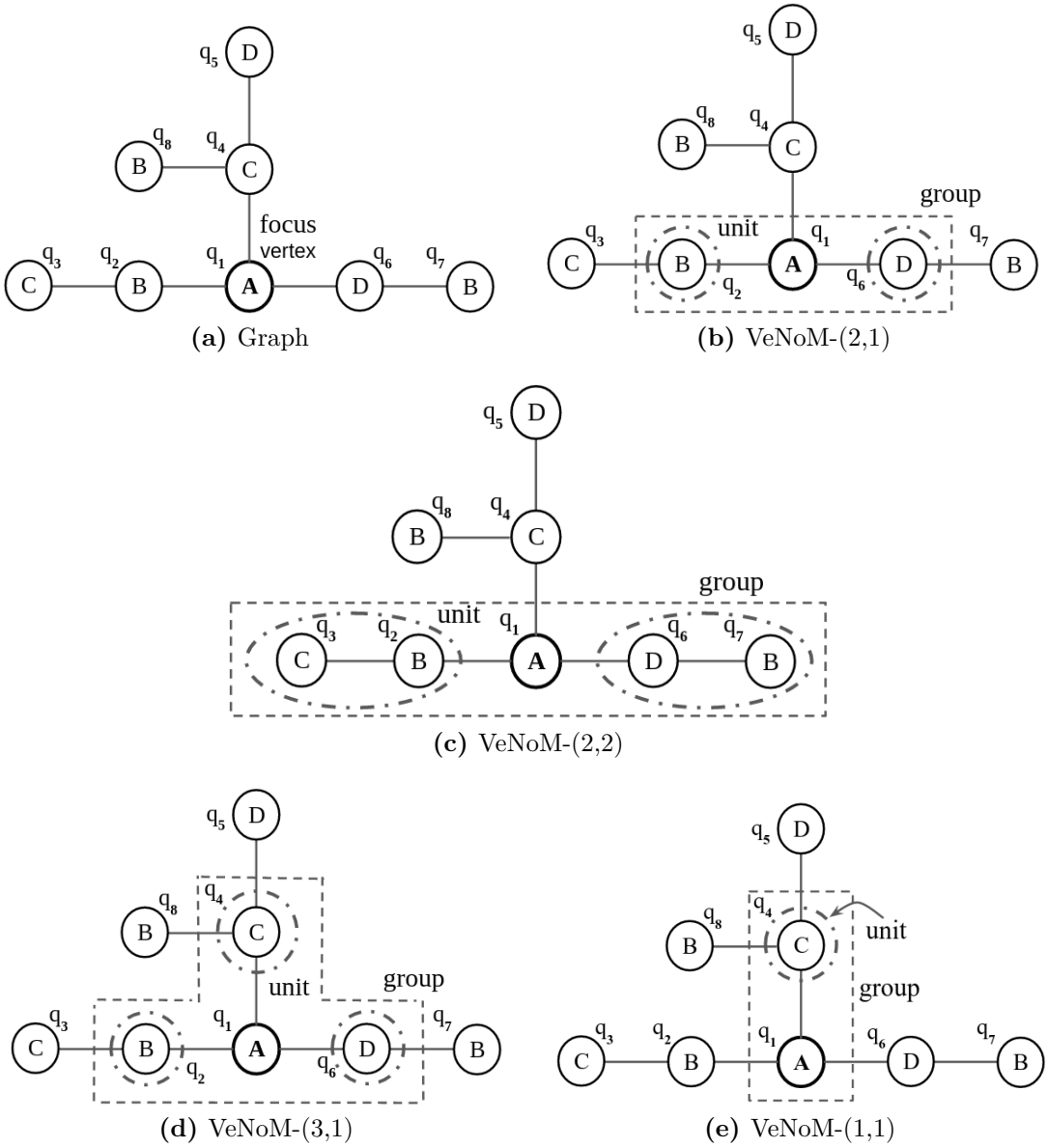
We use the same notation scheme VeNoM-(k,h) for extensions and reductions of VeNoM-(2,1). Essentially, h controls the depth (or hops) of the neighborhood considered around a vertex while k controls the breadth of its neighborhood that is considered during a match.

5.4 Extensions

We describe two methods through which VeNoM-(2,1) can be extended to create a family of algorithms: (1) *multi-hop*, where the depth of a unit is increased to include farther neighbors (Section 5.5); (2) *multi-neighbor* (Section 5.6), the number of units in a group is increased. Additionally, we also discuss how VeNoM-(2,1) can be reduced to a single vertex unit and group version, (Section 5.7).

Figure 5.3 shows the difference between the coverage of neighborhoods in different instances of VeNoM. The dash-dotted line depicts a unit, while the dotted rectangular line outlines a group. For example, for vertex q_1 with label A, in VeNoM-(1,1), VeNoM-(2,1) and VeNoM-(3,1), labels of only its one-hop neighbors are considered as units, i.e., the size of each unit of q_1 is 1. Hence, in Figure 5.3 (b), (d) and (e), q_1 has three units, ($\langle B\rangle, \langle C\rangle$ and $\langle D\rangle$). While for VeNoM-(2,2) a unit is an ordered pair of two vertex labels, $\langle$1-hop vertex label, 2-hop vertex label $\rangle$. Likewise, in Figure 5.3 (c), q_1 has four units, each composed of two vertices: $\langle B, C\rangle, \langle C, D\rangle, \langle C, B\rangle$ and $\langle D, B\rangle$.

In a group, VeNoM-(2,1) and VeNoM-(2,2) use two units, VeNoM-(3,1) groups are composed of three units while groups in VeNoM-(1,1) contain only one unit. In Figure 5.3 (b), there are three possible groups for vertex q_1: $\big\langle A, \langle B\rangle, \langle D\rangle\big\rangle, \big\langle A, \langle B\rangle, \langle C\rangle\big\rangle$ and $\big\langle A, \langle C\rangle, \langle D\rangle\big\rangle$. We discuss the structure of units and groups for other instances in the subsequent sections.



(a) Graph **(b)** VeNoM-(2,1)

(c) VeNoM-(2,2)

(d) VeNoM-(3,1) **(e)** VeNoM-(1,1)

Figure 5.3: Units and groups in different instances of VeNoM, for vertex q_1. Units are represented by dash-dotted elliptical lines and groups by dashed rectangular lines.

5.5 VeNoM-(2,2)

The VeNoM-(2,1) approach can be extended to h-hops by increasing the depth of the neighborhood considered for each vertex to h-hops, thereby increasing the number of entities in a unit. For brevity, we describe the framework for $k = 2$ and $h = 2$.

For VeNoM-(2,2), a unit consists of labels of its one and two-hop neighbors. A unit for $q \in \mathcal{V}_\mathcal{Q}$ is a tuple of the form $\langle \mathcal{L}_\mathcal{Q}(q_i^1), \mathcal{L}_\mathcal{Q}(q_j^2) \rangle$, where $q_i^h \in \mathcal{V}_\mathcal{Q}$ is the i^{th} node exactly h hops away from q. (A similar notation is used corresponding to the

target vertex.) Note that $q_j^2 \in \mathcal{N}_Q(q_i^1) \setminus \{q\}$, i.e., q_j^2 is a neighbor of q_i^1 other than q.

A group in VeNoM-(2,2) is constructed for each vertex using two of the units of the vertex and the label of the vertex itself. We restrict a group to have units that correspond to different 1-hop neighbors of the vertex to ensure a better structural match. A group for q is of the form $\big\langle \mathcal{L}_Q(q), \langle \mathcal{L}_Q(q_i^1), \mathcal{L}_Q(q_j^2)\rangle, \langle \mathcal{L}_Q(q_k^1), \mathcal{L}_Q(q_l^2)\rangle \big\rangle$, with $i \neq k$.

In Figure 5.3c, q_1 has five valid groups:

- $\big\langle A, \langle B, C\rangle, \langle D, B\rangle \big\rangle$,
- $\big\langle A, \langle B, C\rangle, \langle C, B\rangle \big\rangle$,
- $\big\langle A, \langle B, C\rangle, \langle C, D\rangle \big\rangle$,
- $\big\langle A, \langle C, B\rangle, \langle D, B\rangle \big\rangle$, and
- $\big\langle A, \langle C, D\rangle, \langle D, B\rangle \big\rangle$.

An example of an *invalid* group of q_1 in Figure 5.3c is $\big\langle A, \langle C, B\rangle, \langle C, D\rangle \big\rangle$, since both the units $\langle C, B\rangle$ and $\langle C, D\rangle$, correspond to the same neighbor q_4. The units and groups are similarly constructed for the target vertices.

5.5.1 Similarity of a Vertex Pair

To compute the similarity of a vertex pair $\langle q, v\rangle$, we match each group of q against an unmatched group of v. The similarity of two groups is defined based on the similarity of their respective constituent units.

In VeNoM-(2,2), there are three unit-match categories possible, with $u_2 \succ u_1 \succ u_0$:

- u_2: When both the vertex labels of the query unit have an exact match.
- u_1: When exactly one of the vertex labels of the query unit matches.
- u_0: When none of the vertex labels of the query unit find a match.

As maximum matching is preferred, the best possible symbol is assigned.

Similarly, five group-match levels, s_0-s_4, are defined. The level s_0 corresponds to the case when none of the constituent neighbor labels of the query group find a match; s_1 when exactly one of them matches and so on. Again, to maximize overlap the group-match categories are ordered, $s_4 \succ s_3 \succ s_2 \succ s_1 \succ s_0$, i.e., s_4 is the best match level. The group-match categories can be defined based on the match categories assigned to the constituent units. For instance, a group match would be assigned to the category s_0 when none of the constituent units find a match, i.e., both the pairs are assigned to the category u_0. Similarly, a match is assigned to the category s_1 when one of the two pairs results in a u_1 match while the other in u_0. Mathematically,

$$\begin{gathered} s_0 : (u_0 \wedge u_0) \qquad s_1 : (u_0 \wedge u_1) \qquad s_2 : (u_0 \wedge u_2) \vee (u_1 \wedge u_1) \\ s_3 : (u_1 \wedge u_2) \qquad s_4 : (u_2 \wedge u_2) \end{gathered} \tag{5.3}$$

Here, $\wedge$ represents the AND operation and $\vee$ represents the OR operation among the match events. Observe that the five events (s_0-s_4) are exclusive and exhaustive.

To compute the similarity of a vertex pair $\langle q, v \rangle$, we compute its chi-square value, which calculates the deviation of the observed counts of the symbols s_0-s_4, from their expected counts. The observed counts of these events are computed by comparing the groups of q against the groups of v. A group associated with v once matched is not considered for match with any other group of q. The expected counts of the group-match levels can be computed from the probability distribution of the group-match levels. We first compute the probabilities of the unit-match levels and then using Equation 5.3 compute the probability distribution for the group-match levels.

5.5.2 Probability Distribution of Unit Symbols, $P_q(u_i)$

The probability that a label does not match depends on the number of unique labels L in the target graph. Assuming a uniform distribution of labels, the probability that a label does not match is $(1 - 1/L)$. Then the probability that none of the

1-hop neighbors of a query node q with degree d_q match the label is $p_q = (1-1/L)^{d_q}$. The probability that the second hop label in the query unit does not find a match is $\beta_q = (1-1/L)^{b_q}$, where $b_q = \sum_{i=1}^{d_q}(d_{q_i^1} - 1)$ is the number of 2-hop neighbors of q (d_{q^i} is the degree of q_i^1). The value $d_{q_i^i}$ is reduced by 1 to avoid counting q as a 2-hop neighbor of itself. A unit will be assigned the symbol u_1 when exactly one of the two vertices in the unit match. This can happen in two ways, either the 1-hop label matches and the 2-hop label does not match or the 2-hop label matches but the 1-hop label does not. In the first case, the 2-hop label is absent in the neighborhood of the matching 1-hop label. Therefore, the probability of mismatch of the 2-hop label is conditioned upon the 1-hop label, which is computed as $\delta_q = (1-1/L)^{avg_d(q)}$, where $avg_d(q) = \sum_{i=1}^{d_q}(d_{q_i^1} - 1)/d_q$ is the average number of 2-hop neighbors of q per its 1-hop neighbor. For complete match (u_2) as well, the 2-hop label is conditioned upon the matching 1-hop label. Thus,

$$P_q(u_0) = p_q \cdot \beta_q; \qquad P_q(u_1) = (\bar{p}_q \cdot \delta_q) + (p_q \cdot \bar{\beta}_q); \qquad P_q(u_2) = \bar{p}_q \cdot \bar{\delta}_q \tag{5.4}$$

where, $\bar{p}_q = (1 - p_q)$, $\bar{\beta}_q = (1 - \beta_q)$ and $\bar{\delta}_q = (1 - \delta_q)$.

5.5.3 Probability Distribution of Group Symbols, $P_q(s_i)$

The probabilities of the group-match levels can be computed based on Equation. 5.3 and are given in Equation. 5.5.

The conditions for symbols $s_1 - s_3$, as given in the Equation 5.3, can be achieved in more than one way. For instance, in VeNoM-(2,2) for a group to be assigned the match symbol s_1, the best match found for one of its units has to be u_1 and u_0 for the other. The combination of the match categories u_0 and u_1 can manifest in two

ways. Such scenarios are accounted for when computing the probabilities.

$$P_q(s_0) = P_q(u_0) \cdot P_q(u_0) = (p_q \cdot \beta_q)^2 \tag{5.5a}$$

$$P_q(s_1) = 2 \cdot P_q(u_0) \cdot P_q(u_1) = 2 \cdot (p_q \cdot \beta_q) \cdot (\bar{p}_q \cdot \delta_q) + (p_q \cdot \bar{\beta}_q) \tag{5.5b}$$

$$P_q(s_2) = 2 \cdot (p_q \cdot \beta_q) \cdot (\bar{p}_q \cdot \bar{\delta}_q) + \Big((\bar{p}_q \cdot \delta_q) + (p_q \cdot \bar{\beta}_q)\Big)^2 \tag{5.5c}$$

$$P_q(s_3) = 2 \cdot P_q(u_1) \cdot P_q(u_2) = 2 \cdot \Big((\bar{p}_q \cdot \delta_q) + (p_q \cdot \bar{\beta}_q)\Big) \cdot \bar{p}_q \cdot \bar{\delta}_q \tag{5.5d}$$

$$P_q(s_4) = P_q(u_2) \cdot P_q(u_2) = (\bar{p}_q \cdot \bar{\delta}_q)^2 \tag{5.5e}$$

Observe that both the unit-match categories u_0-u_2 and group-match categories s_0-s_4 are exhaustive. In other words, $\sum_{i=0}^{2} P_q(u_i) = 1$ and $\sum_{i=0}^{4} P_q(s_i) = 1$.

5.5.4 Expected Distribution of Symbols

Multiple units and groups can be generated for a vertex, using different 1-hop and 2-hop neighbors in combination. The total number of units associated with query node q can be computed as $D_q = \sum_{q_j^1 \in \mathcal{N}_Q(q)} (d_{q_j^1} - 1)$. Again, the degree $d_{q_j^1}$ is reduced by 1 to avoid counting q as the 2-hop neighbor of itself. From D_q units, $\binom{D_q}{2}$ groups can be constructed. Of these, $\sum_{q_j^1 \in \mathcal{N}_Q(q)} \binom{d_{q_j^1}-1}{2}$ units correspond to the same 1-hop neighbors of q, and are, therefore, invalid. Assuming that all the groups are independent of each other, the total number of valid groups that q can exhibit, is computed as,

$$\eta_q = \binom{D_q}{2} - \sum_{q_j^1 \in \mathcal{N}_Q(q)} \binom{d_{q_j^1} - 1}{2} \tag{5.6}$$

The probability of symbols s_0-s_4 is defined for a single query group, and is multiplied by the total number of groups of q to compute their expected values for v,

$$E_q(s_i) = P_q(s_i) \times \eta_q \tag{5.7}$$

5.6 VeNoM-(3,1)

In the multi-neighbor extension scheme, instead of matching a deeper neighborhood structure, a broader neighborhood is matched. In other words, units under this scheme consist of 1-hop neighbors only, as in VeNoM-(2,1); but, the number of units in a group is increased to 3. The two extension schemes can be used in conjunction with each other, i.e., an instance of the type VeNoM-(3,2) is also possible. However, for ease of understanding, we discuss the scheme with $h = 1$ and $k = 3$ units in a group.

5.6.1 Unit and Group Matches

For the same graph setting, as in Section 5.5, a group with q as the focus vertex would now look like $\langle \mathcal{L}_{\mathcal{Q}}(q), \mathcal{L}_{\mathcal{Q}}(q_i^1), \mathcal{L}_{\mathcal{Q}}(q_j^1), \mathcal{L}_{\mathcal{Q}}(q_k^1) \rangle$, where $i \neq j \neq k$, with only four vertex labels. For instance, for the graph shown in Figure 5.3d, there is only one group possible for vertex q_1, $\big\langle A, \langle B \rangle, \langle C \rangle, \langle D \rangle \big\rangle$.

Since each unit now has only one vertex, there are only two unit-match levels possible, u_0 and u_1, when none or at least one matching unit is found, respectively. As maximum matching is preferred, $u_1 \succ u_0$. Likewise, there are 4 group-match categories, $s_3 \succ s_2 \succ s_1 \succ s_0$, which are defined using the unit-match levels as,

$$s_0 : (u_0 \wedge u_0 \wedge u_0) \quad s_1 : (u_0 \wedge u_0 \wedge u_1) \quad s_2 : (u_0 \wedge u_1 \wedge u_1) \quad s_3 : (u_1 \wedge u_1 \wedge u_1) \tag{5.8}$$

Note that there are three different ways in which the configurations for the symbols s_1 and s_2 can be obtained. As explained in the previous section for VeNoM-(2,2).

5.6.2 Probability Distribution of Symbols

For q, the probabilities for u_0 and u_1, now, are,

$$P_q(u_0) = p_q; \qquad P_q(u_1) = 1 - p_q = \bar{p}_q \tag{5.9}$$

The probabilities of the group-match categories based on the Equation 5.8 and 5.9, are given by,

$$P_q(s_0) = P_q(u_0) \cdot P_q(u_0) \cdot P_q(u_0) = p_q \cdot p_q \cdot p_q = p_q^3 \tag{5.10a}$$

$$P_q(s_1) = 3 \cdot (P_q(u_0))^2 \cdot P_q(u_1) = 3 \cdot p_q^2 \cdot \bar{p}_q \tag{5.10b}$$

$$P_q(s_2) = 3 \cdot P_q(u_0) \cdot (P_q(u_1))^2 = 3 \cdot p_q \cdot \bar{p}_q^2 \tag{5.10c}$$

$$P_q(s_3) = P_q(u_1) \cdot P_q(u_1) \cdot P_q(u_1) = \bar{p}_q \cdot \bar{p}_q \cdot \bar{p}_q = \bar{p}_q^3 \tag{5.10d}$$

5.6.3 Expected Distribution of Symbols

The expected value for each symbol is obtained by appropriately scaling their probabilities corresponding to the number of groups exhibited by the query node q, as shown in Equation. 5.7. For VeNoM-(3,1), the number of groups exhibited by the query node q is

$$\eta_q = \binom{d_q}{3} \tag{5.11}$$

5.7 VeNoM-(1,1)

The VeNoM framework is flexible and the VeNoM-(2,1) version can also be reduced to VeNoM-(1,1) configuration. In VeNoM-(1,1), the groups consist of a single unit which is made up of only one 1-hop neighbor. For example, in Figure 5.3e, q_1 has three possible groups: $\langle A, \langle B \rangle \rangle, \langle A, \langle C \rangle \rangle$ and $\langle A, \langle D \rangle \rangle$. Once again, only two unit-match levels are defined, u_0 and u_1, with $u_1 \succ u_0$, as in Eqn. 5.9.

Now, since there is only a single unit in a group, unlike previous algorithms, the group-match categories become synonymous with the unit-match levels. To clarify, there are now only two group-match levels: s_0 and s_1, with $s_1 \succ s_0$. The category s_0 is assigned when the unit results in a complete mismatch, i.e., when the single constituent vertex label of the unit does not match. While s_1 denotes the case when the unit is a complete match, i.e., the single constituent vertex label finds a match.

Consequently, the group-match probabilities are the same as the unit-match level.

$$P_q(s_0) = P_q(u_0); \qquad P_q(s_1) = P_q(u_1) \tag{5.12}$$

For computing the expected distribution of the group-match categories, the same process is followed as given in Eqn. 5.7. For the reduced case, the value of $\eta_q = d_q$.

5.8 Complexity Analysis

Assume a target graph G with n_G vertices and L unique labels uniformly distributed and a query graph Q with n_Q nodes and average degree d_Q. The time complexity for matching a single query group is $O(1)$ with efficient target graph indexing. For a query node with m groups and n_G/L possible candidate target vertices, the complexity of finding the best match is $O(m \cdot n_G/L)$. The computation of m can be broken down to the number of units possible with depth h and k units in a group. The number of units possible with depth h is of the order $O(d_Q^h)$. The number of possible groups is then approximately $\binom{d_Q^h}{k}$ which is of the order $O(d_Q^{h \cdot k})$. Therefore, the complexity of matching the query graph Q with VeNoM-(k, h) is roughly $O(n_Q \cdot d_Q^{h \cdot k} \cdot n_G/L)$. Since query graphs are typically quite small, d_Q is a constant. The time complexity of the entire algorithm is, thus, effectively $O(n_Q \cdot n_G/L)$.

5.9 Experimental Results

In this section, we compare and analyze the performance of the extension schemes along with the base algorithm on various graph invariants.

5.9.1 Setup

All the algorithms were implemented in C++. The experiments were performed on an Intel(R) Xeon(R) 2.6GHz CPU E5-2697v3 processor with 504GB RAM running CentOS Linux 7.9.2009.

Dataset	#Vertices	#Edges	#Labels
Human	4.6K	86.2K	44
HPRD	9.4K	37K	307
Flickr	80.5K	5.9M	195
PPI	12K	10.74M	2.4K

Table 5.1: Characteristics of real-world datasets

Benchmarks

(1) VELSET Dutta et al. (2017), since it is the algorithm that we extend, and it also outperformed the then state-of-the-art algorithms NeMa Khan et al. (2013) and SIM-T Kpodjedo et al. (2014b); (2) VerSaChI Agarwal et al. (2021), which is a recent ASM framework that outperformed VELSET.

Real Datasets

Subgraph matching is extensively used in bioinformatics and social network analysis, based on which the datasets were chosen. The Flickr (Rossi and Ahmed, 2015) dataset is a large *social network* based on user interactions, while PPI, Human and HPRD (Bi et al., 2016) are *biological networks* with protein-protein interactions. PPI is a randomly extracted small graph from STRING DB (`version-10-5.string-db.org`). Table 5.1 summarizes the characteristics of the datasets.

Synthetic Datasets

We chose the Barabási-Albert (BA) graphs to evaluate the scalability of instances of VeNoM, as it is known to be scale-free and approximate real-world graphs (Albert and Barabási, 2002). It uses a preferential attachment mechanism. It has two parameters: the number of vertices (n) and the growth factor (m), i.e., the number of edges linking a new node to the existing graph. We introduce a third parameter, the number of unique vertex labels (l). The default values of the parameters were set to be $n = 100,000$, $m = 50$ and $l = 150$.

Query Generation

For our experiments, queries of size 9 were created by randomly extracting subgraphs from real-world graph datasets. A seed vertex was selected at random and gradually expanded. As these queries have an exact match available, we name this query set *exact*. Further, we introduce noises in the *exact* query set to generate different types of noisy queries, (i) addition and deletion of edges (*nEAdd* and *nEDel*, respectively), (ii) alteration of vertex label (*nLabel*) and (iii) addition and deletion of vertices (*nVAdd* and *nVDel*, respectively). The maximum number of perturbations done to any exact query graph was capped at 2 while ensuring the connectivity of the query graph. Each query set consisted of 40 instances.

For the synthetic graph experiments, different query sets of size 9 were generated in a similar fashion. For the default synthetic graph, we also create queries of sizes 5, 7, 11 and 13, (in addition to size 9).

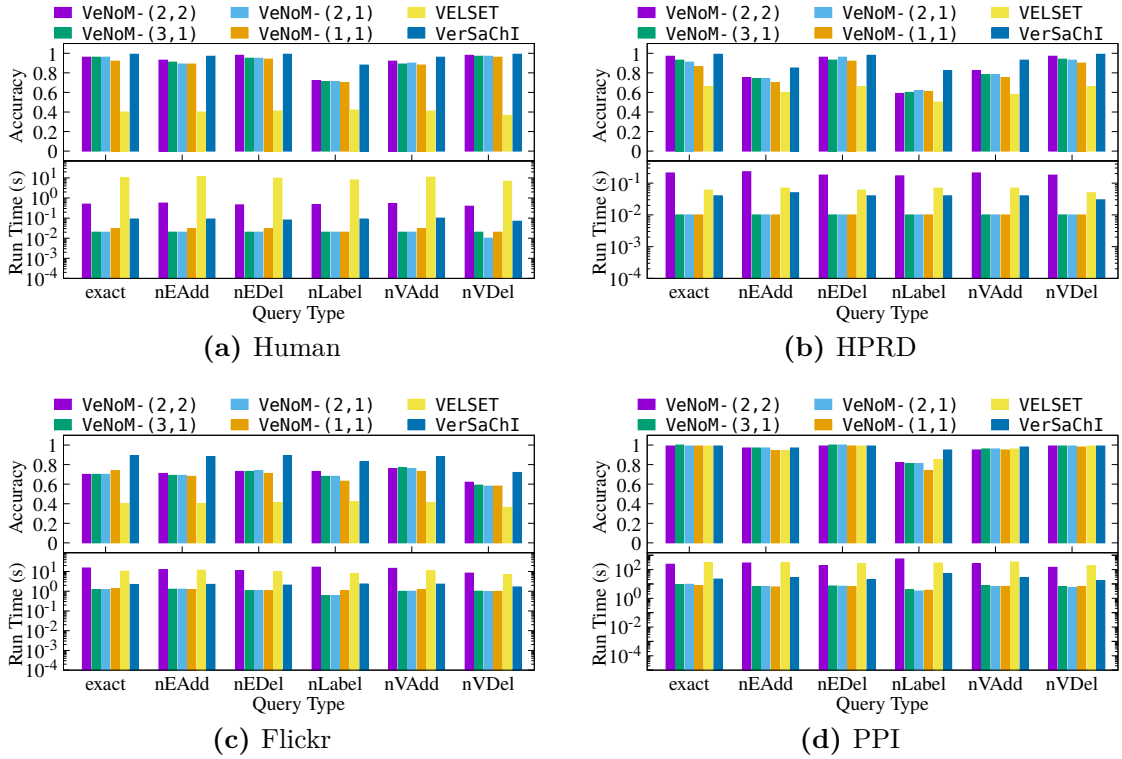
Metrics

We evaluate the performance of the four algorithms on the following metrics:

1. *Maximum mean accuracy*: We define accuracy as the ratio of the number of edges in the answer subgraph that match the query subgraph to the number of edges in the query subgraph. To evaluate the quality of the matching subgraphs returned by the algorithms, we choose the answer subgraph with maximum accuracy among the *top-10* highest χ^2 value subgraphs. The average of the maximum accuracy over each query set is reported. We refer to this as *accuracy* for simplicity.

2. *Average running time*: To evaluate the efficiency of the algorithms, the time taken to return the *top-10* answer subgraphs is recorded for each query. We compare the running times averaged over a query set, referred to as *run time*.

The graph plots in Figure 5.4 and 5.6 depict the run time in the logarithmic scale.



(a) Human **(b)** HPRD

(c) Flickr **(d)** PPI

Figure 5.4: Performance on real-world graphs (run time in log-scale)

5.9.2 Real-World Graphs

Figure 5.4 shows the performance of instances of VeNoM and benchmark algorithms.

The accuracy of VELSET was lowest and can be attributed to the apparent inversion of the chi-square values from the expected trend. The improved performance demonstrated by VeNoM indicates that the proposed remodeling was effective.

Overall, VeNoM-(2,2) performed slightly better than VeNoM-(2,1) or was comparable. This improvement can be attributed to the two-hop neighborhood match done in VeNoM-(2,2) giving it a *look-ahead* advantage over VeNoM-(2,1) and VeNoM-(3,1). At the same time, VeNoM-(1,1) performed slightly worse than VeNoM-(2,1), as it compares only one neighbor at a time, which reduced its capacity to capture the neighborhood topological similarities. However, for PPI it is observed that VeNoM-(3,1) performed marginally better than VeNoM-(2,2). We explore this further in Section 5.9.3. In general, VerSaChI exhibited relatively higher accuracy among all. This can be due to the two-hop neighborhood overlap based similarity. However, the run-time of VerSaChI was five to ten times more than that of VeNoM-(x, 1)

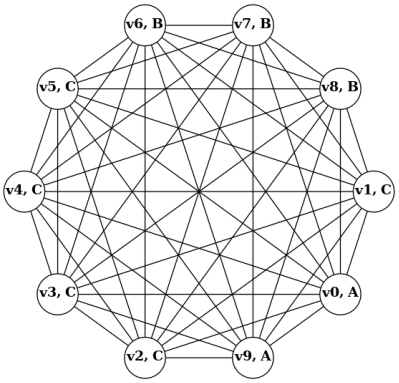
($x \in \{1, 2, 3\}$). Due to the depth of the neighbors considered VeNoM-(2,2) showed a higher run time. As the depth is increased, the number of possible groups increases multi-fold, which added to the complexity of matching. VeNoM-(2,1) and VeNoM-(3,1) were comparable in terms of both accuracy and run time.

In general, a significant drop was observed in the accuracy for the *nLabel* query set for all the algorithms. This can be attributed to the underlying semantic relationship between the node labels. During the noisy label query set generation, the label of some of the vertices was randomly changed to another label from the label vocabulary of the dataset. This did not conform to the underlying label distribution and topology of the real graph, which resulted in partial matches.

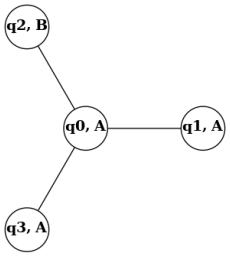
The comparison on real graphs highlights that an edge based neighborhood match, i.e., one vertex at a time, as is the case in VeNoM-(1,1) is insufficient. However, there is no significant improvement when the number of vertices being compared is increased from two (VeNoM-(2,1)) to three (VeNoM-(3,1)). Notably, there is no overhead in runtime in a multi-neighbor extension. Between VerSaChI and VeNoM-(2,2), the former is a more holistic approach and performs better, suggesting that as the depth of the neighborhood being compared increases, a more comprehensive view of the neighborhood may be required, by increasing the neighborhood span, for a better comparison.

5.9.3 VeNoM-(2,2) vs VeNoM-(3,1)

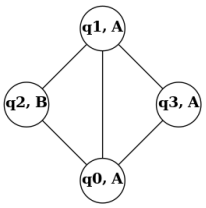
We further analyze the performance difference between VeNoM-(2,2) and VeNoM-(3,1) using toy examples. We choose a complete and labeled graph of size 10 and queries of size 4 (Figure 5.5). For query graph 1, (Figure 5.5 (a)) no performance difference was observed among the algorithms. However, for query graph 2 (Figure 5.5 (b)), VeNoM-(3,1) exhibited a larger accuracy. On further analysis, it was found this is due to the difference in the selection of candidate pair, based on their χ^2 values (as shown in Figure 5.5). Even though the candidate pair $\langle$ q1, v9 $\rangle$ is a better match accuracy-wise, the vertex pair $\langle$ q3, v9 $\rangle$ achieves a higher χ^2 value



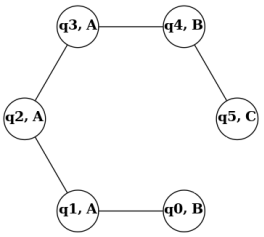
(a) Target Graph



(b) Query Graph 1



(c) Query Graph 2



(d) Query graph 3

Query Graph 2

Algorithms	**Best Match**	**Acc.**	χ^2 **values**	
			⟨ **q1, v9**⟩	⟨ **q3, v9**⟩
VeNoM-(2,2)	⟨ q0, v0 ⟩, ⟨ q2, v7 ⟩, ⟨ q3, v9 ⟩	0.40	1.064	1.074
VeNoM-(3,1)	⟨ q0, v0 ⟩, ⟨ q1, v9 ⟩, ⟨ q2, v7 ⟩	0.60	1.272	1.272
VeNoM-(2,1)	⟨ q0, v0 ⟩, ⟨ q2, v7 ⟩, ⟨ q3, v9 ⟩	0.40	0.871	1.025
VeNoM-(1,1)	⟨ q0, v0 ⟩, ⟨ q2, v7 ⟩, ⟨ q3, v9 ⟩	0.40	1.600	1.999

Query Graph 3

Algorithms	2^{nd} **and;** 3^{rd} **Best Match**	2^{nd} **Best Acc.**	3^{rd} **Best Acc.**
VeNoM-(2,2)	⟨ q4, v7 ⟩, ⟨ q5, v2 ⟩; ⟨ q4, 6 ⟩, ⟨ q5, 3 ⟩	0.2	0.2
VeNoM-(3,1)	⟨ q0, v7 ⟩; ⟨ q5, v2 ⟩	0.0	0.0
VeNoM-(2,1)	⟨ q0, v7 ⟩; ⟨ q5, v2 ⟩	0.0	0.0
VeNoM-(1,1)	⟨ q4, v7 ⟩, ⟨ q5, v5 ⟩; ⟨ q5, v2 ⟩	0.2	0.0

Figure 5.5: Performance comparison of different instances of VeNoM on sample target and query graphs.

for all instances of VeNoM but VeNoM-(3,1). For VeNoM-(3,1), both the above mentioned vertex pairs exhibit same statistical significance which gives it a chance to select ⟨ q1, v9 ⟩ over ⟨ q3, v9 ⟩. For such cases VeNoM-(3,1) may be more desirable than VeNoM-(2,2), as the latter also has a higher runtime. All the algorithms reported a partial match with 0.6 accuracy as the best match for the query graph 3 (Figure 5.5 (c)). However, the overall performance for VeNoM-(2,2) was better with more partial matches in top-3 answers, suggesting that it has the ability to find subgraphs in scenarios where other instances may fail.

5.9.4 Parameter Study

We study the effects of various graph parameters on the performance of the algorithms.

Graph Size

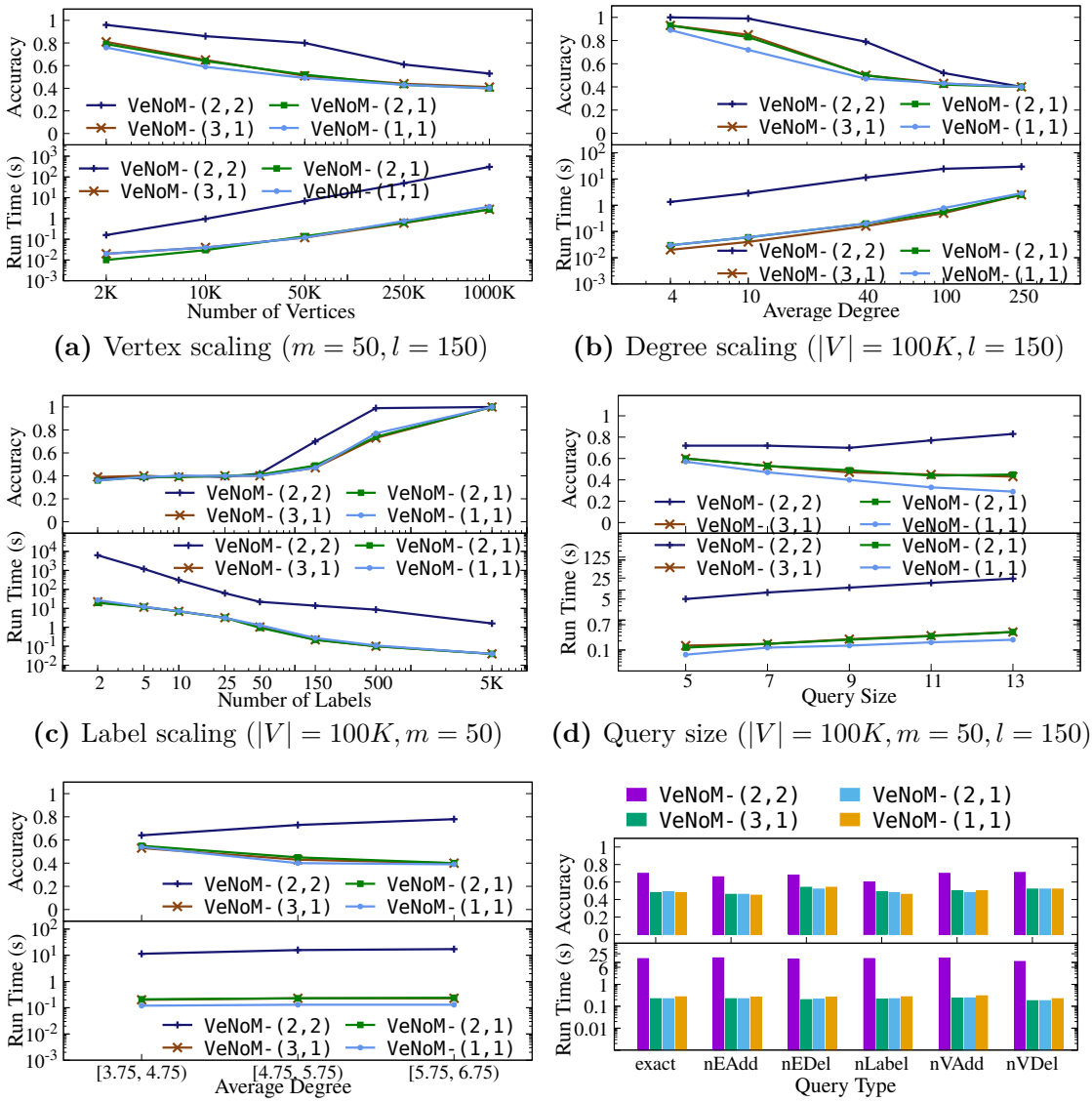
A linear increase in run time was seen for all the algorithms with an increase in the number of vertices (Figure 5.6a). VeNoM-(2,2) is the most accurate as compared to its peers which is due to its ability to match neighbors 2-hops away. The performance of VeNoM-(2,1) and VeNoM-(3,1) remain similar with VeNoM-(1,1) being slightly inferior to them due to its lack of coverage. A higher run time was also reported for VeNoM-(2,2) with an increase in the graph size, as the number of candidate match comparisons increases.

Average Graph Degree

The parameter m (in BA graphs) is directly proportional to the degree of the graph generated. For all the algorithms, a linear increase in runtime was seen with an increase in the average degree of the graph (Figure 5.6b). Noticeably, time taken by VeNoM-(3,1) was observed to be marginally lesser than that of VeNoM-(2,2). This is because the number of groups exhibited by VeNoM-(3,1) is lesser than VeNoM-(2,1), implying a lesser number of groups to match, which results in a lower runtime. The lower runtime of VerSaChI than VeNoM-(2,2), in the previous section (Section 5.9.2) can be attributed to this factor as well. Once again, VeNoM-(2,2) achieved a higher accuracy than others while VeNoM-(1,1) achieved the lowest accuracy for lower graph degrees.

Label Set Size

As the size of the label set was increased, a rapid decrease in the runtime was observed (Figure 5.6c). This is a direct consequence of the decrease in the number of candidate matches caused due to label diversity. Another outcome of this effect is



(a) Vertex scaling ($m = 50, l = 150$) **(b)** Degree scaling ($|V| = 100K, l = 150$)

(c) Label scaling ($|V| = 100K, m = 50$) **(d)** Query size ($|V| = 100K, m = 50, l = 150$)

(e) Query degree ($|V| = 100K, m = 40, l = 150$) **(f)** Noisy Queries ($|V| = 100K, m = 50, l = 150$)

Figure 5.6: Performance evaluation on Barabási-Albert graphs (run time in log-scale)

an increase in the accuracy of the algorithms. All the algorithms showed a significant rise in accuracy from the label set size 50 to 150. This shows that the algorithms are able to distinguish between vertex pairs better after a certain threshold of vertex-per-label ratio is crossed. Another contributing factor towards a lower accuracy, for smaller label sets, is the degree to label ratio, as discussed in Section **??**. Although, VeNoM takes care of inversion in χ^2 values by replacing the target vertex degree with that of the query node, but a graph universe with a very small label set may still lead to value inversion. Once again, VeNoM-(2,2) reached the saturation accuracy

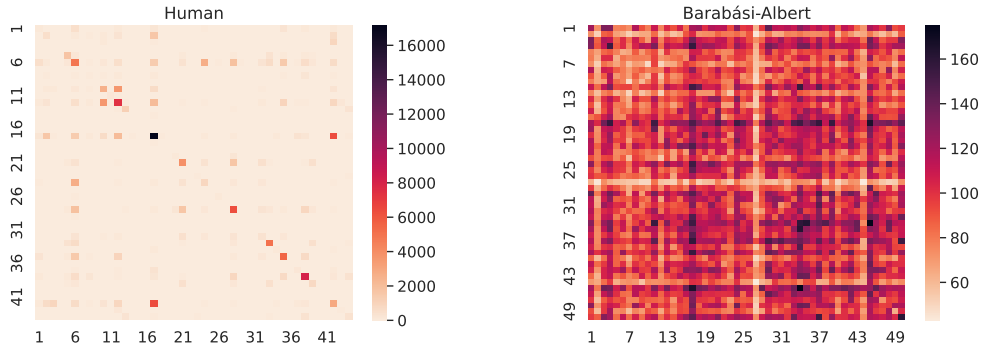
faster by utilizing the *look-ahead* advantage for a better performance.

Query Size

With an increase in the number of vertices in the query graph, a near exponential increase is observed in run time (Figure 5.6d). This is because it increases the number of groups in a query. The effect of this is more pronounced in VeNoM-(2,2) due to the increase in depth of the neighborhood considered. However, VeNoM-(2,2) still outperforms the other VeNoM instances, which show a decrease in accuracy with an increase in the query size. VeNoM-(2,2) shows negligible performance change for small query sizes and a slightly higher accuracy for larger queries. This suggests that with higher number of connections, VeNoM-(2,2) is able to create more meaningful groups, making it more robust toward the increase in query size. On the other hand, VeNoM-(1,1), with a single vertex per group, suffered heavily and achieved the least accuracy.

Query Degree

To better understand the effects of query degree, the queries of size 9 were binned into three buckets based on the average degree of the query graphs: $[3.75, 4.75)$, $[4.75, 5.75)$ and $[5.75, 6.75)$. With an increase in query complexity, no significant change in the runtime was observed in any of the VeNoM instances (Figure 5.6e). However, VeNoM-(2,2) exhibited a steady increase in accuracy and maintained a significant improvement over its counterparts. Meanwhile, the accuracy for VeNoM-(2,1), VeNoM-(3,1) and VeNoM-(1,1) dropped as the average degree of the query graph was increased. The rate of accuracy drop was slightly lower in VeNoM-(2,1) and VeNoM-(3,1), than in VeNoM-(1,1), suggesting slight comparative robustness. The increase in accuracy of VeNoM-(2,2) suggests that with more neighbors, non-trivial groups for a query node could be created, which captured the neighborhood similarity better. Observe that the ratio of the query degree to the label is <0.65 at all times.



(a) Human **(b)** BA graph ($|V| = 5K, m = 25, l = 50$)

Figure 5.7: Heat-map of edge-frequency between two labels for Human and BA graph

Noisy Queries

In experiments with noisy queries, VeNoM-(2,2) performed significantly better than both VeNoM-(2,1) and VeNoM-(3,1), while VeNoM-(1,1) on average reported a relatively lower accuracy. In general, accuracy for *nEDel* and *nVDel* was more than that for *nEAdd* and *nVAdd*, respectively. This is because, in the event of deletion, a perfect match is still guaranteed to exist, while the same cannot be when an edge or a vertex is added to the query. This aligns with the trend seen in the real-world graphs in Figure 5.4. However, in contrast to the performance on real-world graph datasets, a negligible accuracy drop was observed for *nLabel* query set in BA graphs. As hypothesized in Section 5.9.2, this is caused due to the absence of underlying semantic relations of vertex labels in synthetic graphs. In synthetic graphs, the labels were independent of each other and there was no semantic relationship between them. Thus, complete matches could be found for the *nLabel* queries as well. In real graphs, however, labels of vertices that have connections between them may not be random, and, the connections among labels may form a community. We further analyzed the frequency of connections between different labels for the dataset Human and a similar-sized BA graph ($|V| = 5K, m = 25, l = 50$) using heat maps (Figure 5.7). It was observed that for the Human dataset, the connections were concentrated among a handful of labels, while in the BA graph the edges were evenly spread over all label combinations. This reinforced our theory of the existence of

underlying semantic relations among labels in real-world graphs.

5.9.5 Discussion

Overall, our experiments depicted the following trends:

- Incorporating *multi-hop information* offers significant improvements in accuracy, as is shown by VeNoM-(2,2) and VerSaChI. This is potentially due to the extra neighborhood information available during comparison. However, this impacts the run time negatively.

- The strategy of aggregating *multi-neighbor information* has limited improvement in accuracy. At one-hop level neighborhood match, a single edge based comparison is inadequate and increasing it to two neighboring nodes results in an improvement. However, there is no significant improvement when the capacity is increased to three neighboring vertices from two.

- Performance comparison between VeNoM-(2,2) and VerSaChI suggests that comparison of only two units at a time at the 2-hop level is insufficient and a larger view of the neighborhood is required for increased accuracy. Further, the number of groups potentially affects the run time of the algorithm and decreases with larger group size for target graphs with a lower average degree. This causes a slight reduction in the run time.

The above trade-offs can be considered while subgraph matching algorithms for application-specific requirements are designed.

Chapter 6

GraphReach

Inspired by the success of deep neural models on complex data structures, GNN models are being explored for graph matching. However, most GNN architectures rely on neighbors alone for the message-passing mechanism, due to which they fail to distinguish between nodes with similar neighborhoods. Consequently, for predictive tasks that rely on the position of a node with respect to the graph, the performance suffers.

Our goal is to design a GNN that is inductive, captures both position as well as node attribute information in the embeddings, and overcomes the problem of automorphism leading to identical embeddings. We emphasize that we do not claim position information to be more important than the neighborhood structure. Rather, the more appropriate choice depends on the problem being solved and the dataset. For example, if homophily is the main driving factor behind a prediction task, then embeddings based on local neighborhood structure are likely to perform well. On the other hand, to predict whether two nodes belong to the same community, e.g., road networks, gene interaction networks, etc., positional information is more important.

P-GNN (You et al., 2019) is the first work to address the need for an inductive GNN that encodes position information. P-GNN randomly selects a small number of nodes as *anchor* nodes. It then computes the shortest path distances of all remaining nodes to these anchor nodes and embeds this information in a low-dimensional space. Thus, two nodes have similar embeddings if their shortest path distances to the

anchors are similar, as discussed in Section 1.2.4. Although P-GNN has shown impressive improvements over existing GNN architectures in link prediction and pairwise node classification, there is scope to improve.

6.1 Problem Formulation

We represent a graph as $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \boldsymbol{\mathcal{X}}, \mathcal{W})$, where $\mathcal{V}$ is the set of nodes v_i, with $1 \leq i \leq n$ and $\mathcal{E}$ is the set of edges. The attribute set $\boldsymbol{\mathcal{X}} = \{\mathbf{x}_{v_1}, \cdots, \mathbf{x}_{v_n}\}$ has a one-to-one correspondence with the node set $\mathcal{V}$ where $\mathbf{x}_{v_i}$ represents the feature vector of node $v_i \in \mathcal{V}$. Similarly, $\mathcal{W}$ has a one-to-one correspondence with $\mathcal{E}$, where w_{e_i} denotes the edge weights.

A *node embedding model* is a function $f : \mathcal{V} \rightarrow \mathbf{Z}$ that maps the node set $\mathcal{V}$ to a d-dimensional vector space $\mathbf{Z} = \{\mathbf{z}_1, \cdots, \mathbf{z}_n\}$. Our goal is to learn a *position-aware* node embedding model.

Definition 6.1 (Position-aware Embedding)**.** *A node embedding* $\mathbf{z}_v, \forall v \in \mathcal{V}$ *is position-aware if there exists a function* $g(\cdot, \cdot)$ *such that* $d(v_i, v_j) = g(\mathbf{z}_i, \mathbf{z}_j)$*, where* $d(v_i, v_j)$ *is the distance from* v_i *to* v_j *in* $\mathcal{G}$*.*

The distance $d(v_i, v_j)$ should reflect the quality of all paths between v_i and v_j wherein, (1) $d(v_i, v_j)$ is directly proportional to the number of paths between v_i and v_j, and (2) $d(v_i, v_j)$ is inversely proportional to the lengths of the paths between v_i and v_j. We capture these aspects in the form of *reachability estimations* through *random walks*. Note that, $d(\cdot, \cdot)$ is not required to be a metric distance function.

Reachability estimations are similar to PageRank (Brin and Page, 1998) and Random Walk with Restarts (Pan et al., 2004). To the best of our knowledge, GIL (Xu et al., 2020) is the only other GNN framework that uses the concept of reachability estimations. However, GIL uses reachability as an additional feature along with node embeddings for node classification.

6.2 Reachability Estimations

In a fixed-length random walk of length l_w, we start from an initial node v_i, and jump to a neighboring node u through an outgoing edge $e = (v_i, u)$ with *transition probability*,

$$p(e) = \frac{w_e}{\sum_{\forall e' \in N(v_i)} w_{e'}} \tag{6.1}$$

Here, $N(v_i)$ denotes the set of outgoing edges from v_i. From node u, the process continues iteratively in the same manner till l_w jumps. If there are many short paths from v_i to v_j, there is a high likelihood of reaching v_j by a *random walk* from v_i. We utilize this property to define a similarity measure:

$$s(v_i, v_j) = \frac{\sum_{k=1}^{n_w} count_k(v_i, v_j)}{l_w \times n_w} \tag{6.2}$$

Here, n_w denotes the number of random walks started from v_i, and $count_k(v_i, v_j)$ denotes the number of times random walker visited v_j in the k^{th} random walk starting from v_i. The similarity function could also weight the nodes according to the order they appear in a random walk.

Order-weighted Similarity

In Equation 6.2, the order in which nodes occur in the random walk does not affect the similarity function. To incorporate this sequential aspect, we propose the use of *harmonic* weighting. Formally, let $o(v_i, v_j, k)$ denote the step count at which v_j is visited in the k^{th} random walk originating from node v_i; if v_j was not visited, then $o(v_i, v_j, k) = \infty$. The *order-weighted* similarity is defined as:

$$s_o(v_i, v_j) = \sum_{k=1}^{n_w} \frac{1}{o(v_i, v_j, k)} \tag{6.3}$$

Empirically, minimal change in accuracies was observed between the two metrics,

Symbol	Dimensions	Description
$\boldsymbol{\mathcal{X}} = \{\mathbf{x}_v \mid \forall v \in \mathcal{V}\}$	$n \times d$	Original node attributes
$\mathbf{H}^l = \{\mathbf{h}_v^l \mid \forall v \in \mathcal{V}\}$	$n \times d_{hid}$	Feature matrix at layer l
$\boldsymbol{\mathcal{M}}^l = \{\boldsymbol{\mathcal{M}}_v^l \mid \forall v \in \mathcal{V}\}$	$n \times k \times d_{hid}$	Message Tensor at layer l
$\mathbf{W}_{\mathcal{M}}^l$	$2 \cdot d_{hid} \times d_{hid}$	Transform $\boldsymbol{\mathcal{M}}^l$
$\mathbf{a}_{att}^l$	$2 \cdot d_{hid} \times 1$	Attention vector at layer l
$\mathbf{Z} = \{\mathbf{z}_v \mid \forall v \in \mathcal{V}\}$	$n \times k$	Output embeddings
$\mathbf{W}_Z$	$d_{hid} \times 1$	Transforms $\boldsymbol{\mathcal{M}}^L$ to $\mathbf{Z}$

Table 6.1: Matrix notations and descriptions.

and hence we use the similarity function based on frequency counts. From $s(v_i, v_j)$, one may define $d(v_i, v_j) = 1 - s(v_i, v_j)$. However, we directly work with similarity $s(\cdot, \cdot)$ since $d(v_i, v_j)$ is not explicitly required in our formulation.

6.3 GraphReach

The symbols and notations used throughout this chapter are summarized in Table 6.1. All the matrix and vector notations are denoted in bold letters by convention.

6.3.1 The Architecture

Algorithm 1 outlines the pseudocode and Figure 6.1 pictorially depicts the architecture. In addition to the hyper-parameters and message passing functions, the input to Algorithm 1 includes the graph and k anchor nodes. The details of the anchor selection procedure are discussed in Section 6.3.2. In the initial layer, the embedding of a node v is simply its attribute vector $\mathbf{x}_v$ (line 1). In each hidden layer, a set of *messages*, $\boldsymbol{\mathcal{M}}^l$, is computed using the message computing function $\mathcal{F}(v, a, \mathbf{h}_v^l, \mathbf{h}_a^l)$ between each node-anchor pair (details in Section 6.3.3) (lines 2-6). The component $\boldsymbol{\mathcal{M}}_v^l[i]$ in $\boldsymbol{\mathcal{M}}^l$ represents the message received by node v from the i^{th} anchor node. These messages are then aggregated using an aggregation function $\mathcal{S}$ (line 7), which we will discuss in detail in Section 6.3.4. The aggregated messages thus obtained are propagated to the next layer. In the final layer, the set of messages for each node, i.e., $\boldsymbol{\mathcal{M}}_v^l$, is linearly transformed through a trainable weight matrix $\mathbf{W}_Z$ (line

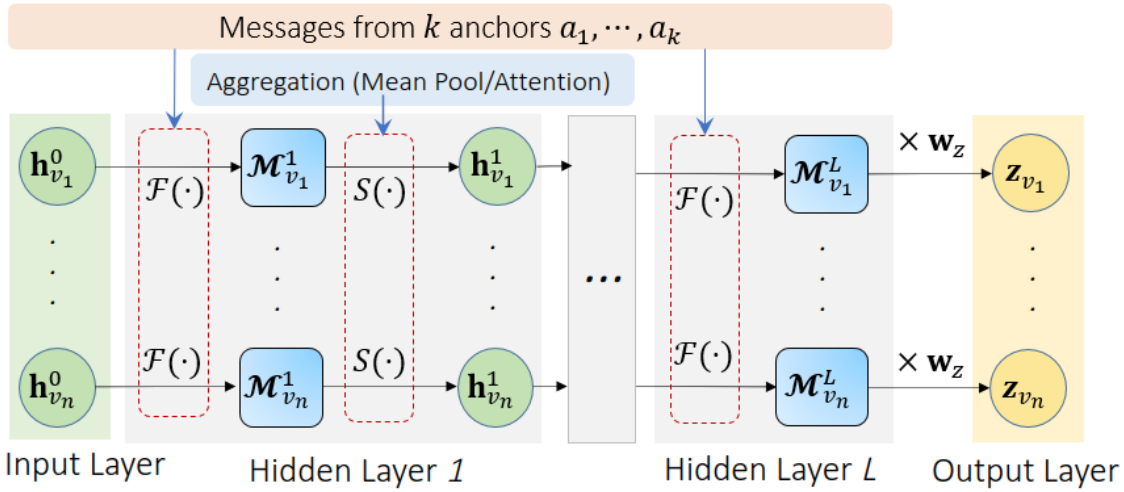


Figure 6.1: The architecture of GraphReach.

8).

Algorithm 1 GraphReach

Input: Graph $G = (\mathcal{V}, \mathcal{E}, \boldsymbol{\mathcal{X}}, \mathcal{W})$; Anchors $\{a_i\}$; Message computation function $\mathcal{F}$; Message aggregation function $\mathcal{S}$; Number of layers L; Non-linear function σ

Output: Node embedding $\mathbf{z}_v, \forall v \in \mathcal{V}$

$\mathbf{h}^0_v \leftarrow \mathbf{x}_v, \forall v \in \mathcal{V}$

for $l = 1, \cdots, L$ **do**

  for $v \in \mathcal{V}$ **do**

    for $i = 1, \cdots, k$ **do**

      $\widehat{\boldsymbol{\mathcal{M}}}^l_v[i] \leftarrow \mathcal{F}(v, a_i, \mathbf{h}^{l-1}_v, \mathbf{h}^{l-1}_{a_i})$ ▷ Message Computation

    $\boldsymbol{\mathcal{M}}^l_v = (\oplus_{a_i \in \mathcal{A}} \widehat{\boldsymbol{\mathcal{M}}}^l_v[i]) \cdot \mathbf{W}^l_{\boldsymbol{\mathcal{M}}}$ ▷ Concatenation: Eq. 6.10

    $\mathbf{h}^l_v \leftarrow \mathcal{S}\left(\boldsymbol{\mathcal{M}}^l_v\right)$ ▷ Message Aggregation: Eq. 6.11 and Eq. 6.13

return $\mathbf{z}_v \in \mathbb{R}^k \leftarrow \sigma(\boldsymbol{\mathcal{M}}^L_v.\mathbf{W}_Z), \forall v \in \mathcal{V}$

6.3.2 Anchor Selection

Anchors act as our reference points while encoding node positions. It is therefore imperative to select them carefully. Selecting two anchors that are close to each other in the graph is not meaningful since the distance to these anchors from the rest of the nodes would be similar. Ideally, the anchors should be diverse as well as *reachable* from a large portion of nodes.

Formally, let $\mathcal{R}$ be the set of all random walks performed across all nodes in $\mathcal{V}$. We represent the reachability information in the form of a *bipartite* graph $\mathcal{B} = (\mathcal{V}_1, \mathcal{V}_2, \mathcal{E}_B)$. Here, $\mathcal{V}_1 = \mathcal{V}_2 = \mathcal{V}$. There is an edge $e = (u, v) \in \mathcal{E}_B$, $u \in \mathcal{V}_1$, $v \in \mathcal{V}_2$

if there exists a walk in $\mathcal{R}$ that starts from v and *reaches* u. The *reachability set* of a subset of nodes $\mathcal{A}$ is:

$$\rho(\mathcal{A}) = \{v \mid (u, v) \in \mathcal{E}_B,\ u \in \mathcal{A}, v \in \mathcal{V}_2\} \tag{6.4}$$

Our objective is to find the set of k anchors $\mathcal{A}^* \subseteq \mathcal{V}_1$, $|\mathcal{A}^*| = k$, that *maximizes* reachability. Specifically,

$$\mathcal{A}^* = \underset{\mathcal{A} \subseteq \mathcal{V}_1, |\mathcal{A}|=k}{\arg\max} \{|\rho(\mathcal{A})|\} \tag{6.5}$$

Lemma 6.2. *The maximization problem in Eq. 6.5, performed on the bipartite graph formed on the reachability set, is NP-hard.*

Proof. We reduce the *Maximum Coverage problem (MCP)* to the problem of anchor selection.

Definition 6.3 (Maximum Coverage)**.** *Given a collection of subsets* $\mathbb{S} = \{S_1, \cdots, S_m\}$ *from a universal set of items* $U = \{t_1, \cdots, t_n\}$ *and budget* k*, choose at most* k *subsets* $\mathcal{T}^* \subseteq \mathbb{S}$ *such that the coverage of items* $\cup_{\forall S_i \in \mathcal{T}^*} S_i$ *is maximized.*

MCP is known to be NP-hard (Cormen et al., 2009).

Given an arbitrary instance of MCP, we construct a bipartite graph $\mathcal{B} = (\mathcal{V}_1, \mathcal{V}_2, \mathcal{E}_B)$, where we have a node $u_i \in \mathcal{V}_1$ corresponding to each subset $S_i \in \mathbb{S}$, a node $v_j \in \mathcal{V}_2$ corresponding to each item $t_j \in U$ and an edge $e = (u_i, v_j) \in \mathcal{E}_B$ if S_i contains item t_j. With this construction, it is easy to see that $\mathcal{A}^* \subseteq \mathcal{V}_1,\ |\mathcal{A}^*| = k$ maximizes reachability, i.e., $|\ \rho(\mathcal{A}^*)\ |$, if and only if selecting the subsets corresponding to the nodes in $\mathcal{A}^*$ maximizes coverage of items from U. □

Lemma 6.4. *For any given set of nodes* $\mathcal{A}$*,* $f(\mathcal{A}) = |\rho(\mathcal{A})|$ *is monotone and submodular.*

Monotonicity: The function $f(\mathcal{A}) = |\rho(\mathcal{A})|$ is monotone since adding any node from $u \in \mathcal{V}_1$ to $\mathcal{A}$ can only bring in new neighbors from $\mathcal{V}_2$. Hence, $\rho(\mathcal{A} \cup \{u\}) \supseteq \rho(\mathcal{A})$

.

Submodularity: A function $f(S)$ is *submodular* if the *marginal gain* from adding an element to a set S is at least as high as the marginal gain from adding it to a superset of S. Mathematically, it satisfies:

$$f(S \cup \{o\}) - f(S) \geq f(T \cup \{o\}) - f(T) \tag{6.6}$$

for all elements o and all pairs of sets $S \subseteq T$.

We prove reachability maximization is submodular by contradiction.

Proof by contradiction:

Assume,

$$f(T \cup \{v\}) - f(T) > f(\mathcal{A} \cup \{v\}) - f(\mathcal{A}) \tag{6.7}$$

where $\mathcal{A}$ and T are subsets of $\mathcal{V}_1$, such that $\mathcal{A} \subseteq T$, and $v \in \mathcal{V}_1$ of bipartite graph $\mathcal{B}$. Given that $f(\mathcal{A})$ is monotone, Equation 6.7 is feasible only if:

$$\begin{gathered} \rho(\{v\}) \setminus \rho(T) \supseteq \rho(\{v\}) \setminus \rho(\mathcal{A}) \\ \text{or,} \qquad \mathcal{A} \not\subseteq T \end{gathered} \tag{6.8}$$

which contradicts the assumption that $\mathcal{A} \subseteq T$. □

For *monotone* and *submodular* optimization functions, the greedy-hill climbing algorithm provides a $(1 - 1/e)$ approximation guarantee (Nemhauser et al., 1978). We, thus, follow the same strategy and iteratively add the node that provides highest *marginal* reachability (Algorithm 2).

Algorithm 2 outlines the pseudocode for the greedy anchor selection procedure. We start from an empty anchor set $\mathcal{A}$ (line 1), and in each iteration, add the node $u \in \mathcal{V}_1$ that has the highest (*marginal*) degree in $\mathcal{B}$ to $\mathcal{A}$ (lines 3-4). Following this operation, we remove u from $\mathcal{V}_1$ and all neighbors of u from $\mathcal{V}_2$ (lines 5-6). This process repeats for k iterations (line 2).

Algorithm 2 Greedy Anchor Selection

Input: Graph $\mathcal{B} = (\mathcal{V}_1, \mathcal{V}_2, \mathcal{E}_B)$; number of anchors: k
Output: $\mathcal{A}$: a set of k anchors (nodes)

```
A ← ∅
while |A| < k do
    u* ← arg max_{u∈V1} {|ρ(A ∪ {u})| − |ρ(A)|}
    A ← A ∪ {u*}
    V2 ← V2 \ ρ(u*)
    V1 ← V1 \ {u*}
return A
```

Corollary 6.5. *If set $\mathcal{A}$ is the output of Algorithm 2, then $\mid \rho(\mathcal{A}) \mid \geq (1 - 1/e) \mid \mathcal{A}^* \mid$, where $\mathcal{A}^*$ is the anchor set of size k that maximizes reachability coverage.*

Proof. Follows from Lemma 6.4.

Modeling Reachability Frequency.

The above approach models reachability as a binary occurrence; even if there exists just one walk where $v \in \mathcal{V}_2$ reaches $u \in \mathcal{V}_1$, an edge (u, v) is present in $\mathcal{B}$. It does not incorporate the frequency with which u is visited from v. To capture this aspect, we randomly sample $X\%$ of the walks from $\mathcal{R}$ and form the bipartite graph only on this sampled set. Note that the down-sampling does not require the bipartite graph to be fully connected. Algorithm 2 is then run to select anchors on this bipartite graph. This process is repeated multiple times by drawing multiple samples of the same size from $\mathcal{R}$ and the final anchor set consists of nodes that are selected in the answer sets the highest number of times. In our experiments, we sample 5 subsets with $X = 30\%$.

6.3.3 Message Computation

The message computation function $\mathcal{F}\left(v, a, \mathbf{h}_v^l, \mathbf{h}_a^l\right)$ should incorporate both the position information of node v with respect to the anchor set $\mathcal{A}$ as well as the node attributes. While node attributes may provide important information that may be useful in the eventual prediction task, position information, in the form of reachability estimations, captures the location of the node in the global context of the

graph. To encode these dual needs, $\mathcal{F}\left(v, a, \mathbf{h}_v^l, \mathbf{h}_a^l\right)$ is defined as follows.

$$\mathcal{F}\left(v, a, \mathbf{h}_v^l, \mathbf{h}_a^l\right) = \Big(\left(s(v,a) \times \mathbf{h}_v^l\right) \parallel \left(s(a,v) \times \mathbf{h}_a^l\right)\Big) \tag{6.9}$$

where $\parallel$ denotes the concatenation of vectors. The function $\mathcal{F}$ takes as input a node v, an anchor a and their respective layer attributes, $\mathbf{h}_v^l$ and $\mathbf{h}_a^l$. It returns a message vector, which is a weighted aggregation of their individual attributes in proportion to their reachability estimations (Equation 6.2). Observe that, the reachability estimations are used in both directions to account for an asymmetric distance function.

Due to concatenation, the output message vector has dimensions $2 \cdot d_{hid}$, where d_{hid} is the dimension of the node embeddings in the hidden layers. To linearly transform the message vector back into $\mathbb{R}^{d_{hid}}$, we multiply it with a weight vector $\mathbf{W}_{\mathcal{M}}^l$. The complete global structure information for node v is encompassed in the message matrix $\boldsymbol{\mathcal{M}}_v^l$ ($\oplus$ denotes row-wise stacking of message vectors).

$$\boldsymbol{\mathcal{M}}_v^l = \Big(\bigoplus_{a \in \mathcal{A}} \mathcal{F}\left(v, a, \mathbf{h}_v^l, \mathbf{h}_a^l\right)\Big) \cdot \mathbf{W}_{\mathcal{M}}^l \tag{6.10}$$

6.3.4 Message Aggregation

To compute the hidden representation of nodes, messages corresponding to anchors are aggregated for each node. We propose two aggregation schemes.

1. **Mean Pool (M)**: In this, a simple mean of the message vectors are taken across anchors.

$$\mathcal{S}^M(\boldsymbol{\mathcal{M}}_v^l) = \frac{1}{k} \sum_{i=1}^{k} \boldsymbol{\mathcal{M}}_v^l[i] \tag{6.11}$$

2. **Attention Aggregator (A)**: In mean pooling, all anchors are given equal weight. Theorizing that the information being preserved can be enhanced by capturing the significance of an anchor with respect to a node, we propose to

calculate the significance distribution among anchors for each node. Following the Graph Attention Network (GAT) architecture (Velickovic et al., 2018), we compute attention coefficients of anchors for an anchor-based aggregation. The attention coefficient of the i^{th} anchor a_i is computed with trainable weight vector $\mathbf{a}^l_{att}$ and weight matrix $\mathbf{W}^l_{att}$. For node v, the attention weight with respect to anchor i is

$$\alpha_v[i] = \text{sm}\Big(\sigma_{att}\Big((\mathbf{h}^l_v \cdot \mathbf{W}^l_{att} \parallel \boldsymbol{\mathcal{M}}^l_v[i] \cdot \mathbf{W}^l_{att}) \cdot \mathbf{a}^l_{att}\Big)\Big) \tag{6.12}$$

Here, *sm* denotes the softmax function. As followed in GAT architecture, we use *LeakyReLU* as the non-linear function σ_{att} (with negative input slope 0.2).

Finally, the messages are aggregated across anchors using these coefficients.

$$\mathcal{S}^A(\boldsymbol{\mathcal{M}}^l_v) = \sum_{i=1}^{k} \alpha_v[i] \times \boldsymbol{\mathcal{M}}^l_v[i] \cdot \mathbf{W}^l_{att} + \mathbf{h}^l_v \cdot \mathbf{W}^l_{att} \tag{6.13}$$

6.3.5 Hyper-parameters for Reachability Estimation

Reachability information relies on two key random walk parameters: the length l_w of each walk, and the total number of walks n_w. If l_w is too short, then we do not gather information with respect to anchors more than l_w-hops away. With n_w, we allow the walker to sample enough number of paths so that our reachability estimations are accurate and holistic. We borrow an important theorem from (Wadhwa et al., 2019) to guide our choice of l_w and n_w.

Theorem 6.6. *If there exists a path between two nodes u and v in a graph, with $1 - 1/n$ probability the random walker will find the path if the number of random walks conducted, n_w, is set to $\Theta(\sqrt[3]{n^2 \ln n})$ with the length of each random walk, l_w, being set to the diameter of the graph.*

6.3.6 Complexity Analysis

We conduct n_w random walks of length l_w for all the n nodes of the graph; this requires $O(n_w \cdot l_w \cdot n)$ time. For anchor set selection, we sample random walks multiple times and select k anchors using the resulting bipartite graphs formed. The complexity of sampling walks is $O(n \cdot n_w)$ while selecting the anchors takes $O(k + k \cdot \log k) = O(k \cdot \log k)$ operations. Considering that each node communicates with k anchors, there are $O(n \cdot k)$ message computations. The aggregation of messages also requires $O(n \cdot k)$ operations owing to k messages being aggregated for each of the n nodes. The attention aggregator has an additional step devoted to the computation of attention coefficients which takes $O(n \cdot k)$ time as well.

6.4 Experiments

In this section, we benchmark GraphReach and establish that GraphReach provides up to 40% relative improvement over state-of-the-art GNN architectures.

6.4.1 Setup

All the experiments have been performed on an Intel(R) Xeon(R) Silver 4114 CPU with a clock speed of 2.20GHz. The GPU used was NVIDIA GeForce RTX 2080 Ti (12GB of FB memory). We use PyTorch 1.4.0 and NetworkX 2.3 on CUDA 10.0. GraphReach is implemented in Python 3.7.6. The codebase of all other benchmarked models is obtained from the respective authors.

Datasets

Below we explain the semantics of each of these datasets.[1] A summary of the characteristics of the datasets is given in Table 6.2

- *Grid* is a synthetic 2D grid graph with $20 \times 20 = 400$ nodes and no features.

[1] All the datasets have been taken from `https://github.com/JiaxuanYou/P-GNN`.

- *Communities* is the connected caveman graph (Watts, 1999). It has 20 communities of 20 nodes each.
- *PPI* is a protein-protein interaction network containing 24 graphs (Zitnik and Leskovec, 2017). Each graph on average has 3000 nodes and 33000 edges. Each node is characterized with a 50-dimensional feature vector.
- *Email-Complete* is a real-world communication graph from SNAP (Leskovec et al., 2007).
- *Email* dataset is a set of 7 graphs obtained by dividing Email-Complete and has 6 communities. The label of each node denotes which community it belongs to.
- *Protein* is a real graph from (Borgwardt et al., 2005). It contains 1113 components. Each component on average contains 39 nodes and 73 edges. Each node has 29 features and is labeled with the functional role of the protein.
- *CoRA* is a standard citation network of machine-learning papers with 2.7K documents, 5.4K links and 1433 distinct word vector attributes, divided into seven classes (McCallum et al., 2000).
- *CiteSeer* is another benchmark citation graph with 3.3K documents, 4.7K links and 3703 distinct word vector attributes, divided into six classes (Giles et al., 1998).

Baselines

We measure and compare the performance of GraphReach with five baselines:
1. P-GNN (You et al., 2019) 2. GCN (Kipf and Welling, 2017) 3. GraphSAGE (Hamilton et al., 2017) 4. GAT (Velickovic et al., 2018) 5. GIN (Xu et al., 2019)

Predictive Tasks

GraphReach is evaluated on three different tasks.

Datasets	#Nodes	#Edges	#Labels	Diameter	#Attributes
Grid	400	760	-	38	-
Communities	400	3.8K	20	9	-
PPI	56.6K	818K	-	8	50
Email-Complete	986	16.6K	42	7	-
Email	920	7.8K	6	7	-
Protein	43.4K	81K	3	64	29
CoRA	2.7K	5.4K	7	19	1433
CiteSeer	3.3K	4.7K	6	28	3703

Table 6.2: Characteristics of graph datasets used.

- **Link Prediction (LP):** Given a pair of nodes in a graph, the task is to predict whether there exists a link (edge) between them.
- **Pairwise Node Classification (PNC):** Given two nodes, the task is to predict whether these nodes belong to the same class label or come from different labels.
- **Node Classification (NC):** For each node, the task is to predict the class/label of the node.

Loss

We evaluate GraphReach on the prediction tasks of *Link Prediction (LP)* and *Pairwise node classification (PNC)* using the *Binary Cross Entropy* (BCE) loss with *logistic activation*, and on *Node Classification (NC)* using the *Negative Log Likelihood* (NLL) loss.

Setting

For LP, we evaluate in both inductive and transductive settings, whereas for PNC, only inductive setting is used.

- **Transductive learning:** The nodes are assigned a fixed ordering. Consequently, the model needs to be re-trained if the ordering changes. Since the ordering is fixed, *one-hot* vectors can be used as unique identifiers of nodes. We use these one-hot vectors to augment the node attributes.

- **Inductive learning:** Only the node attributes are used since under this scenario, the model must generalize to unseen nodes.

Train-Validation-Test Setup

For all prediction tasks, the datasets are individually split in the ratio of 80:10:10 for training, validation and testing, respectively. In LP, the positive set contains actual links present in the graph. The negative set is constructed by sampling an equal number of node pairs that are not linked. A similar strategy is also applied for PNC. In NC, we randomly sample train, validation and test nodes with their corresponding labels. We always ensure that the test data is unseen. When a graph dataset contains multiple graphs, we divide each component into train, validation and test sets.

Each experiment (train and test) is repeated 10 times following which we report the mean *ROC AUC* and the standard deviation.

Default Parameters and Design Choices.

Unless specifically mentioned, we set the number of anchors (k) as $\log^2 n$. While our experiments reveal that a smaller number of anchors is sufficient, since P-GNN uses $\log^2 n$ anchors, we keep it the same. The length of each random walk, l_w, is set to *graph diameter*, and the number of walks n_w as 50. We also conducted experiments to analyze how these parameters influence the prediction accuracy of GraphReach.

We use *Attention* aggregation (Equation 6.13) and simple random walk counts as the similarity function (Equation 6.2) to compare with the baselines.

Common Parameters

The number of *hidden layers* is set to 2. The *hidden embedding dimension* is set to 32. All models are trained for 2000 epochs. The *learning rate* is set to 0.01 for the first 200 epochs and 0.001 thereafter. The *drop-out* parameter is set to 0.5. *Batch size* is kept to 8 for Protein and PPI, and 1 for all other datasets. The final

Models	Communities	Email	Email-Complete	Protein
GCN	0.520 ± 0.025	0.515 ± 0.019	0.536 ± 0.006	0.515 ± 0.002
GraphSAGE	0.514 ± 0.028	0.511 ± 0.016	0.508 ± 0.004	0.520 ± 0.003
GAT	0.620 ± 0.022	0.502 ± 0.015	0.511 ± 0.008	0.528 ± 0.011
GIN	0.620 ± 0.102	0.545 ± 0.012	0.544 ± 0.010	0.523 ± 0.002
P-GNN -F	0.997 ± 0.006	0.640 ± 0.037	0.630 ± 0.031	0.729 ± 0.176
P-GNN -E	$\mathbf{1.000 \pm 0.001}$	0.640 ± 0.029	0.637 ± 0.037	0.631 ± 0.175
GraphReach	$\mathbf{1.000 \pm 0.000}$	$\mathbf{0.949 \pm 0.009}$	$\mathbf{0.935 \pm 0.006}$	$\mathbf{0.904 \pm 0.003}$

(a) Pairwise Node Classification (PNC)

Models	Grid-T	Communities-T	Grid	Communities	PPI
GCN	0.698 ± 0.051	0.981 ± 0.004	0.456 ± 0.037	0.512 ± 0.008	0.769 ± 0.002
GraphSAGE	0.682 ± 0.050	0.978 ± 0.003	0.532 ± 0.050	0.516 ± 0.010	0.803 ± 0.005
GAT	0.704 ± 0.050	0.980 ± 0.005	0.566 ± 0.052	0.618 ± 0.025	0.783 ± 0.004
GIN	0.732 ± 0.050	0.984 ± 0.005	0.499 ± 0.054	0.692 ± 0.049	0.782 ± 0.010
P-GNN -F	0.637 ± 0.078	0.989 ± 0.003	0.694 ± 0.066	$\mathbf{0.991 \pm 0.003}$	0.805 ± 0.003
P-GNN -E	0.834 ± 0.099	0.988 ± 0.003	0.940 ± 0.027	0.985 ± 0.008	0.808 ± 0.003
GraphReach	$\mathbf{0.945 \pm 0.021}$	$\mathbf{0.990 \pm 0.005}$	$\mathbf{0.956 \pm 0.014}$	$\mathbf{0.991 \pm 0.003}$	$\mathbf{0.810 \pm 0.002}$

(b) Link Prediction (LP)

Table 6.3: ROC AUC evaluation of GraphReach with benchmarks. (Grid-T and Communities-T indicates performance in transductive settings. P-GNN -E uses the exact shortest path distance to all anchors while P-GNN -F is a fast variant of P-GNN that uses truncated 2-hop shortest path distance.)

embeddings for PNC and LP tasks are passed through 1-layer MLPs characterized by $label(v, u) = \sigma\left(\mathbf{z_v}^T \mathbf{z_u}\right)$ where σ is the *sigmoid* activation function. The final embeddings for NC are passed through *LogSoftmax* layer to get the log-probabilities of each class. In both LP and PNC, the input is a pair of nodes, while in NC it is a single node. The neural network parameters are tuned using the Adam optimizer (Kingma and Ba, 2015).

6.4.2 Overall Results

Pairwise Node Classification (PNC)

Table 6.3a summarizes the performances in PNC. We observe a dramatic performance improvement by GraphReach over all existing GNN models. While P-GNN clearly established that encoding global positions of nodes helps in PNC, GraphReach further highlights the need to go beyond the shortest paths. Except in Communities, the highest accuracy achieved by any of the baselines is 0.73. In sharp contrast, GraphReach pushes the ROC AUC above 0.90, which is a significant

Dataset	Task	Training Time (in sec)		Inference Time (in sec)	
		P-GNN -E	GraphReach	P-GNN -E	GraphReach
CoRA	LP	326	111	0.01	0.02
CiteSeer		537	125	0.01	0.01
PPI		5, 901	2,980	0.20	0.20
CoRA	PNC	405	265	0.05	0.05
CiteSeer		645	381	0.07	0.06
Protein		13,552	11,254	0.90	0.90

Table 6.4: Training and Inference time comparison for GraphReach

$\approx 40\%$ relative improvement, on average, over the state-of-the-art.

Link Prediction (LP)

Table 6.3b presents the results for LP. Consistent with the trend observed in PNC, GraphReach outperforms other GNNs across inductive and transductive settings. P-GNN and GraphReach are significantly better than the rest of the architectures. This clearly indicates that position-aware node embeddings help. GraphReach being better than P-GNN suggests that a holistic approach of encoding position with respect to all paths is necessary.

The second observation from Table 6.3b is that the performance of position-*unaware* architectures are noticeably better in a transductive setting. Since the transductive setting allows unique identification of nodes through one-hot encodings, traditional GNN architectures are able to extract some amount of position information, which helps in the prediction. In contrast, for both P-GNN and GraphReach, one-hot encodings do not impart any noticeable advantage as position information is captured through distances to anchors.

We also compare the time taken by the two best performing models in Table 6.4. We observe that, on average, GraphReach is 2.5 times faster than P-GNN . P-GNN is slower since it samples a new set of anchors in every layer and epoch, which necessitates the need to recompute distances to all anchors. In contrast, GraphReach uses the same set of strategically chosen anchors through all layers. The inference times of both techniques are less than a second and, hence, are not a

Task	Dataset	GCN		GraphSAGE		GAT		GIN		P-GNN		GraphReach	
		S+T	S	S+T	S	S+T	S	S+T	S	S+T	S	S+T	S
LP	CoRA	**0.86**	0.59	0.85	0.53	**0.86**	0.51	**0.86**	0.59	0.81	0.77	0.83	**0.84**
LP	CiteSeer	**0.87**	0.61	0.85	0.56	**0.87**	0.56	0.86	0.68	0.77	**0.76**	0.77	0.74
PNC	CoRA	**0.98**	0.50	0.96	0.50	**0.98**	0.51	**0.98**	0.52	0.86	0.59	0.96	**0.77**
PNC	CiteSeer	**0.96**	0.51	0.95	0.51	**0.96**	0.50	**0.96**	0.53	0.77	0.57	0.91	**0.61**
NC	CoRA	**0.92**	0.52	**0.92**	0.50	0.90	0.50	0.90	0.54	0.73	0.50	0.84	**0.86**
NC	CiteSeer	**0.82**	0.52	**0.82**	0.50	0.81	0.50	**0.82**	0.53	0.73	0.55	0.75	**0.71**

Table 6.5: ROC AUC comparison of position-aware and traditional GNNs ('S' denotes the version containing only the graph structure and 'S+T' denotes structure with node attributes.)

computational concern.

6.4.3 Difference from Neighborhood-based GNNs

We conducted experiments on attributed graph datasets, with and without attributes for prediction tasks. Table 6.5 presents the results for CoRA and CiteSeer on LP, PNC and NC.

In addition to the network structures, both CoRA and CiteSeer, are also accompanied by binary word vectors characterizing each node. When the word vectors are ignored, the performance of neighborhood-aggregation based GNNs are significantly inferior ($\approx 25\%$) to GraphReach. When supplemented with word vectors, they outperform GraphReach ($\approx 10\%$ better). This leads to the following conclusions.

1. Position-aware GNNs are better at utilizing the structural information.

2. Neighborhood-aggregation based GNNs may be better in exploiting the feature distribution in the neighborhood. (This is not always though, e.g., in PPI and Protein, GraphReach is better than the baselines even when attribute information is used as seen in Tables 6.3a and 6.3b).

3. The two approaches are *complementary* in nature and, therefore, good candidates for ensemble learning.

To elaborate, neighborhood aggregation based architectures rely on *homophily* (Zhu et al., 2020). Consequently, if the node property is a function of neighborhood attribute distribution, then neighborhood aggregation performs well. On the other

Dataset	Task	P-GNN			GraphReach		
		Bf	Af		Bf	Af	
Communities	PNC	0.92	0.82	−0.10	1.00	0.98	**−0.02**
Communities	LP	1.00	0.89	−0.11	1.00	1.00	**−0.00**

Table 6.6: Robustness to adversarial attacks. (*Bf* and *Af* indicate ROC AUC before and after collusion respectively, while Δ denotes the change in accuracy due to collusion.)

hand, in LP, even if the local neighborhoods of two distant nodes are isomorphic, this may not enhance their chance of having a link. Rather, the likelihood increases if the two nodes have many neighbors in common. When two nodes have common neighbors, their distances to the anchors are also similar, and this positional information leads to improved performance.

6.4.4 Adversarial Attacks

We assume the standard *black-box* adversarial setup where the attacker has knowledge of only the graph and can modify it through the addition or deletion of edges (Li et al., 2020; Chang et al., 2020). Let $G = (\mathcal{V}, \mathcal{E})$ be the test graph which has a *colluding* subset of nodes $\mathcal{C} \subseteq \mathcal{V}$. The nodes in $\mathcal{C}$ can add as many edges as needed among them so that predictions on $\mathcal{C}$ are inaccurate.

For PNC, we randomly sample 10% nodes from the unseen test graph and form a clique among these colluding nodes. For LP, we randomly sample 10% of the node pairs from the unseen test graph such that they do not have an edge between them. From this set, we select the top-2% of the highest degree nodes and connect them to the remaining colluding nodes through an edge. This makes the diameter of the colluding group at most 2.

We perform prediction using pre-trained models of both P-GNN and GraphReach on the test graph and measure the ROC AUC on the colluding group. This process is repeated with 5 different samples of colluding groups and the mean accuracy is reported. If the model is robust, despite the collusion, its prediction accuracy should not suffer.

As visible in Table 6.6, the impact on the accuracy of GraphReach is minimal.

Models	Communities	Email	Email-Complete	Protein
GR-A	$\mathbf{1.000 \pm 0.000}$	$\mathbf{0.949 \pm 0.009}$	0.935 ± 0.006	0.904 ± 0.003
GR-M	$\mathbf{1.000 \pm 0.000}$	0.938 ± 0.017	$\mathbf{0.945 \pm 0.004}$	$\mathbf{0.916 \pm 0.008}$
GR-M$^-$	0.500 ± 0.000	0.500 ± 0.000	0.500 ± 0.000	0.559 ± 0.007

(a) Pairwise Node Classification (PNC)

Models	Grid-T	Communities-T	Grid	Communities	PPI
GR-A	$\mathbf{0.945 \pm 0.021}$	0.990 ± 0.005	$\mathbf{0.956 \pm 0.014}$	0.991 ± 0.003	0.810 ± 0.002
GR-M	0.940 ± 0.018	$\mathbf{0.994 \pm 0.003}$	0.931 ± 0.020	$\mathbf{0.993 \pm 0.003}$	$\mathbf{0.830 \pm 0.004}$
GR-M$^-$	0.542 ± 0.071	0.888 ± 0.046	0.500 ± 0.000	0.500 ± 0.000	0.519 ± 0.026

(b) Link Prediction (LP)

Table 6.7: ROC AUC in PNC and LP for ablation study over aggregation functions. (GR stands for GraphReach.)

On the other hand, P-GNN receives more than 10% relative drop in the performance. This result highlights another advantage of reachability estimations. Since GraphReach incorporates all paths in its position-aware embeddings, causing a significant perturbation in position with respect to all paths in the network is difficult. In the case of P-GNN , perturbing shortest paths is relatively easier and hence there is a higher impact on the performance.

6.4.5 Ablation Study

Mean Pool versus Attention

In the first two rows of Table 6.7, we compare the performance of GraphReach with Attention (GraphReach-A) and Mean Pool (GraphReach-M) aggregation functions. As clearly evident, the performance is comparable. In the mean pool, the message from each anchor is weighted equally, while when an attention layer is used, GraphReach learns an additional importance weight for each anchor to aggregate messages. The comparable performance of the Mean Pool and Attention shows that the similarity encoded in the message from each anchor is enough to learn meaningful embeddings; the marginal gain from an additional attention layer is minimal.

Models	Communities	Email	Email-Complete	Protein
GraphReach-OA	$\mathbf{1.000 \pm 0.000}$	0.937 ± 0.009	0.936 ± 0.004	0.906 ± 0.004
GraphReach-OM	$\mathbf{1.000 \pm 0.000}$	$\mathbf{0.949 \pm 0.014}$	0.934 ± 0.004	0.909 ± 0.006

(a) Pairwise Node Classification

Models	Grid-T	Communities-T	Grid	Communities	PPI
GraphReach-OA	0.935 ± 0.025	0.990 ± 0.005	$\mathbf{0.956 \pm 0.014}$	0.992 ± 0.004	0.822 ± 0.007
GraphReach-OM	0.940 ± 0.025	0.993 ± 0.003	0.951 ± 0.017	$\mathbf{0.993 \pm 0.004}$	0.825 ± 0.003

(b) Link Prediction

Table 6.8: ROC AUC in PNC and LP for ablation study over reachability estimation function. GraphReach-OA and GraphReach-OM denote Mean Pool and Attention aggregation, respectively, with order-weighted reachability estimation (Equation 6.3).

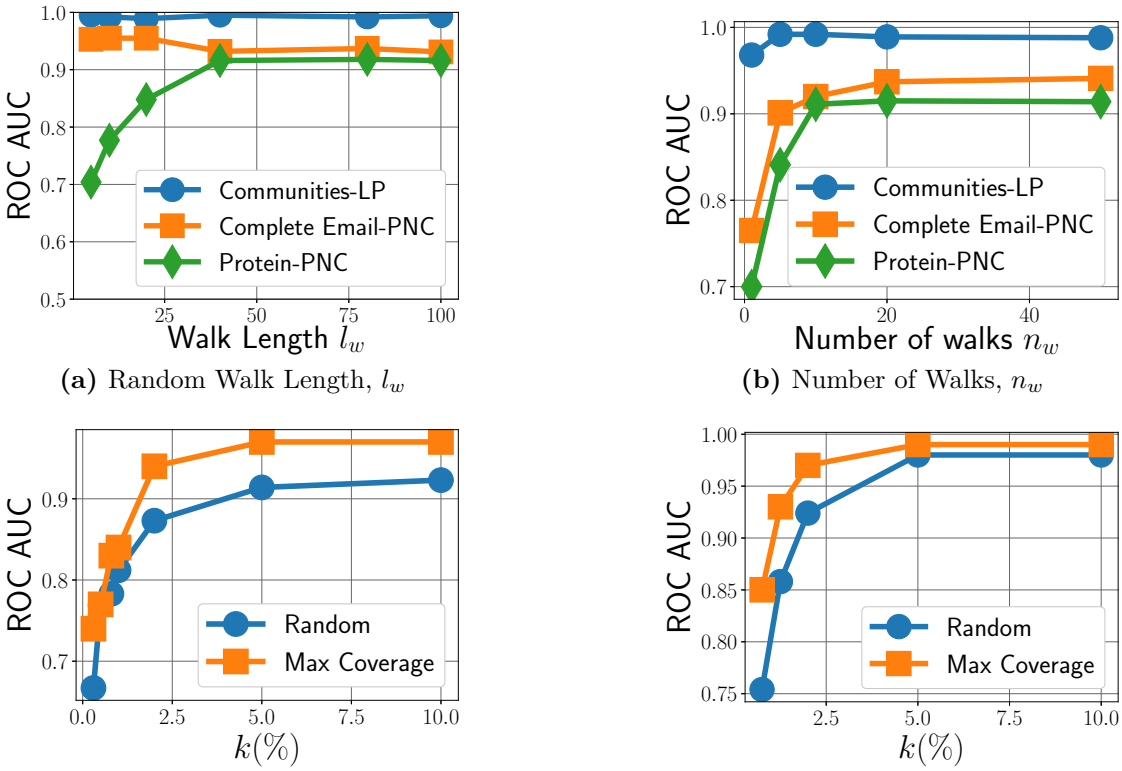
To further substantiate this claim, we alter the Equation 6.9 to

$$\mathcal{F}\left(v, a, \mathbf{h}_v^l, \mathbf{h}_a^l\right) = \mathbf{h}_v^l \parallel \mathbf{h}_a^l$$

i.e., contributions of all anchors are made equal. The variant GraphReach-M$^-$ presents the performance with this modification; as evident, there is a massive drop in accuracy.

Order-weighted Similarity

Table 6.8 presents the performance achieved with the order-weighted similarity function. We observe a minimal change in accuracies when compared to using frequency counts (Equation 6.2). A random walker is more likely to visit nearby nodes from the source than those located far away. Consequently, the *early* nodes that receive higher weights in order-weighted similarity are often the same ones that are visited repeatedly. Hence, order-weighting is correlated to count frequency.



(a) Random Walk Length, l_w

(b) Number of Walks, n_w

(c) PNC (Email-Complete)

(d) LP (Communities)

Figure 6.2: Impact of (a) walk length (b) number of walks and (c-d) number of anchors for PNC and LP, on the accuracy of GraphReach.

6.4.6 Impact of Parameters

Random Walk Length, l_w

Figure 6.2a presents the result on three datasets covering both PNC and LP. Results on the remaining datasets are provided in Table 6.9. On Communities and Email-Complete, the impact of l_w is minimal. For small datasets such as Communities, Email, Email-Complete and PPI, the ROC AUC scores start saturating when l_w is around 10, which is approximately the diameter of the datasets. The accuracy on the Grid dataset saturates for $l_w = 20$ (diameter is 38)while in Protein, the accuracy saturates at around $l_w = 40$. This is a direct consequence of the property that Protein has a significantly larger diameter of 64 (see Table 6.2). Recall from our discussion in Section 6.3.5 that setting l_w to the graph diameter is recommended for ensuring accurate reachability estimations. The trends in Figure 6.2a substantiate this theoretical result.

l_w	Communities	Email	Email-Complete	Protein
5	1.000 ± 0.000	0.953 ± 0.011	0.922 ± 0.003	0.704 ± 0.167
10	1.000 ± 0.000	0.955 ± 0.015	0.940 ± 0.007	0.777 ± 0.168
20	1.000 ± 0.000	0.955 ± 0.010	0.934 ± 0.007	0.848 ± 0.140
40	1.000 ± 0.000	0.922 ± 0.020	0.937 ± 0.008	0.916 ± 0.004
80	1.000 ± 0.000	0.927 ± 0.011	0.936 ± 0.005	0.918 ± 0.007
100	1.000 ± 0.000	0.921 ± 0.022	0.943 ± 0.003	0.916 ± 0.004

(a) Pairwise Node Classification

l_w	Grid-T	Communities-T	Grid	Communities	PPI
5	0.921 ± 0.033	0.989 ± 0.005	0.938 ± 0.021	0.995 ± 0.002	0.815 ± 0.006
10	0.922 ± 0.015	0.991 ± 0.003	0.939 ± 0.011	0.992 ± 0.002	0.827 ± 0.005
20	0.951 ± 0.014	0.993 ± 0.002	0.938 ± 0.017	0.989 ± 0.002	0.833 ± 0.001
40	0.952 ± 0.013	0.991 ± 0.003	0.948 ± 0.020	0.995 ± 0.001	0.830 ± 0.004
80	0.920 ± 0.033	0.993 ± 0.004	0.940 ± 0.015	0.992 ± 0.004	0.823 ± 0.004
100	0.938 ± 0.026	0.992 ± 0.003	0.935 ± 0.029	0.994 ± 0.002	0.821 ± 0.005

(b) Link Prediction

Table 6.9: Effect of the length of random walk on the accuracy of GraphReach.

Number of Random Walks, n_w

Figure 6.2b presents the results. As expected, with a higher number of walks, the accuracy initially improves and then saturates. As discussed earlier in Section 6.3.5, theoretically, the number of random walks conducted is proportional to the number of nodes in the graph. The trend in Figure 6.2b is consistent with this result. The saturation point for all the datasets was achieved for a small number of random walks as shown in Table 6.10.

Number of Anchors and Anchor Selection Strategy

Figures 6.2c and 6.2d present the results with varying number of anchors as a percentage of the dataset and different anchor selection strategies. In addition to the default anchor selection strategy, we evaluate the accuracy obtained when an equal number of anchors are selected randomly. Two key properties emerge from this experiment. First, as the number of anchors increases, the accuracy improves till a saturation point. This is expected since with more anchors we have more reference

n_w	Communities	Email	Email-Complete	Protein
1	0.999 ± 0.002	0.882 ± 0.018	0.765 ± 0.014	0.700 ± 0.165
5	1.000 ± 0.000	0.946 ± 0.007	0.901 ± 0.008	0.841 ± 0.140
10	1.000 ± 0.000	0.909 ± 0.030	0.920 ± 0.009	0.911 ± 0.004
20	1.000 ± 0.000	0.950 ± 0.014	0.937 ± 0.003	0.915 ± 0.007
50	1.000 ± 0.000	0.949 ± 0.016	0.941 ± 0.003	0.914 ± 0.002
75	1.000 ± 0.000	0.936 ± 0.008	0.941 ± 0.005	0.916 ± 0.003
100	1.000 ± 0.000	0.932 ± 0.021	0.939 ± 0.004	0.920 ± 0.004
200	1.000 ± 0.000	0.955 ± 0.007	0.940 ± 0.004	0.916 ± 0.003
500	1.000 ± 0.000	0.947 ± 0.006	0.948 ± 0.004	0.914 ± 0.006

(a) Pairwise Node Classification

n_w	Grid-T	Communities-T	Grid Communities	PPI	
1	0.739 ± 0.048	0.988 ± 0.006	0.805 ± 0.036	0.968 ± 0.013	0.624 ± 0.028
5	0.920 ± 0.021	0.991 ± 0.004	0.898 ± 0.009	0.992 ± 0.002	0.790 ± 0.006
10	0.926 ± 0.023	0.992 ± 0.002	0.941 ± 0.023	0.992 ± 0.002	0.812 ± 0.003
20	0.935 ± 0.012	0.993 ± 0.006	0.932 ± 0.019	0.989 ± 0.002	0.825 ± 0.005
50	0.924 ± 0.026	0.990 ± 0.003	0.946 ± 0.018	0.988 ± 0.002	0.834 ± 0.002
75	0.932 ± 0.013	0.990 ± 0.004	0.961 ± 0.017	0.996 ± 0.001	0.832 ± 0.005
100	0.949 ± 0.010	0.992 ± 0.002	0.955 ± 0.016	0.992 ± 0.004	0.829 ± 0.004
200	0.926 ± 0.018	0.993 ± 0.002	0.925 ± 0.018	0.990 ± 0.002	0.824 ± 0.003
500	0.938 ± 0.005	0.994 ± 0.001	0.938 ± 0.018	0.992 ± 0.004	0.818 ± 0.006

(b) Link Prediction

Table 6.10: Effect of the number of walks on the accuracy of GraphReach.

points to accurately encode node positions. Second, the proposed anchor selection strategy is clearly better than random anchor selection. More importantly, the proposed anchor selection strategy saturates at around 2.5% compared to 5% in random. Recall that the dimension of the final embeddings, i.e., the final layer, is equal to the number of anchors. Consequently, this experiment highlights that high-quality embeddings can be obtained within a low-dimensional space. A low dimension is preferred in various tasks such as indexing and querying of multi-dimensional points, due to the adverse impacts of *curse-of-dimensionality* (Samet, 2006).

Chapter 7

Conclusions and Future Work

In this thesis, we identify the need for a *parameter-free* approximate subgraph matching framework for both deterministic and probabilistic graphs. To this end, we propose two ASM frameworks based on *statistical significance test*: VerSaChI for deterministic graphs and ChiSeL for probabilistic input graphs. Additionally, we propose a novel position-aware node embedding model, GraphReach, which is particularly useful for subgraph matching neural models that rely on node embedding aggregation to achieve the final graph embedding. As a part of our research, we conducted a study on VeNoM, a proposed variant of an existing ASM approach, VELSET. Our experimental study involved varying the depth and breadth of the neighborhood considered during a match, and we observed that an integrated approach with both breadth and depth of the neighborhood performs better.

The VerSaChI ASM framework for deterministic graphs performs a two-hop neighborhood similarity computation between the query node and the input vertex during the search process. To capture the underlying distribution of the input graph, the one-hop similarity of input vertices is computed and characterized into fine buckets using *Chebyshev's Inequality*. For the two-hop similarity computation, the one-hop similarity of neighbors of candidate vertices with query nodes is also considered, and their deviation from the expected value is computed. The candidate matches are expanded to a solution through a greedy approach. The experiments show significant improvements in terms of accuracy compared to state-of-the-art

methods, as well as robustness to noise.

ASM in probabilistic input graphs requires an enumeration of each possible world to search for a feasible subgraph match, making it a challenge. Our proposed framework, ChiSeL, avoids expensive possible world enumeration based on insights. The *chi-squared statistic* is used to quantify the label and structural overlap of the candidate input vertex with the query node and a greedy neighborhood search is used to arrive at the best possible solution. Empirical results show that ChiSeL was highly accurate and efficient on large-scale graphs, reporting an accuracy of 84% with an average runtime of less than a second on billion-edge protein-protein interaction networks.

During our investigation into the factors affecting the performance of an ASM approach, we discovered a limitation in VELSET, where in some cases, input vertices were penalized for having a higher degree. To address this limitation, we proposed VeNoM, which is an enhancement over VELSET, and created various instances of VeNoM by adjusting the depth and breadth of the neighborhood considered during a match. Our extensive experiments, along with comparisons with other state-of-the-art algorithms, revealed that increasing the depth of the neighborhood comparison improves accuracy at the expense of longer runtime, while breadth has little effect as long as it is greater than a single vertex, meaning that a single edge is the unit of comparison.

As part of our exploration of parameter-free ASM paradigms, we investigated neural models for subgraph retrieval. Such models often obtain a graph embedding from its constituent node embeddings. In this context, we introduce GraphReach, an inductive *graph neural network* that generates *position-aware* node embeddings. These embeddings encode the location of the nodes within the global context of the graph by using a set of selected *anchor nodes*. The model employs a random walk based reachability distance metric and a diversified anchor selection strategy, making it more holistic, closely aligned with real-world semantics, and robust against adversarial attacks. Experimental results show that compared to state-of-the-art

GNN architectures, GraphReach achieved up to a 40% relative improvement in accuracy.

The development and application of the approximate subgraph matching algorithms presented in this thesis exhibit the potential of statistical significance matching. Moreover, their versatility and effectiveness, as shown by experiments done on *protein-protein interaction* dataset, open up new avenues for applications in biological research for drug discovery and drug repurposing, among other applications in the medical domain. Potential applications also exist in fields such as data analysis, information retrieval using knowledge graphs, network security, chemoinformatics (e.g., compound analysis) ultimately benefiting relevant communities or industries. The node embeddings of GraphReach can be augmented to graph nodes for similarity computation in VerSaChI. However, this is non-trivial and could be used in addition to the label matching. Further, GraphReach node embeddings encode the global positioning of the nodes, which may be more helpful for graph similarity. Approaches that use GraphReach embeddings obtained with random walks limited to a 2-hop or 3-hop radius can be used to match for smaller subgraph patterns. However, the target graph may need to be decomposed into multiple subgraphs for effective matching.

7.1 Scope for Further Work

During our research, we assumed that the label distribution of graphs was uniform, which may not hold for real-world graphs where labels are often interdependent and do not follow a uniform distribution. For instance, in social networks, a person is more likely to be connected to people he shares nationality with. Further, the inter-dependence of labels is also evident in the datasets we have used, as shown for Human in Figure 5.7; Protein-Protein Interaction networks also exhibit this behavior. Future work could focus on developing ASM frameworks that cater to such needs, and it would be interesting to compare their performance with existing frameworks.

Additionally, in the case of probabilistic graphs, ChiSeL assumed that edge probabilities were independent of each other, however, in general, such is not the case with real-world graphs. (Yuan et al., 2012) is one such work in this direction. A potential direction for future research is to explore statistical significance-based ASM approaches for graphs with a joint probability distribution. It would also be insightful to study the performance of ASM methods for various types of probabilistic graphs discussed in this study.

Another potential avenue for future research is the exploration of how position-aware embeddings behave/ perform in GNN architectures designed for graph-level tasks such as subgraph matching, subgraph retrieval and graph classification. Numerous Graph Convolutional Network (GCN) based architectures have been developed for predicting graph similarity, primarily relying on metrics like *graph edit distance (GED)* and *maximum common subgraph (MCS)*. These models stand to gain significant advantages by incorporating or enhancing their existing frameworks with position-aware embeddings. Some notable such models are SIMGNN (Bai et al., 2019) that predicts similarity using GCN embedding based pairwise node comparison and graph-graph interaction; GMN (Li et al., 2019b) uses a cross-graph attention-based matching mechanism that quantifies the degree to which a node in one graph can be matched with nodes in the other graph; in another GNN model, GENN (Wang et al., 2021), an A* search tree is used to mask nodes and predicts similarity on the graph embeddings derived from the cached GCN node embeddings of the remaining nodes; GraphSim (Bai et al., 2020), HGMN (Xiu et al., 2020) and H^2MN (Zhang et al., 2021) each generates similarity matrix for outputs of different layers of a GCN to capture node similarity at different depths of the neighborhood, and differ in initial inputs; the similarity matrices are fed to a CNN or fully connected MLP for graph similarity prediction. GOTSim (Doan et al., 2021) is another GNN which jointly learns over a pair of graphs by computing optimal graph transformation cost from the similarity matrices to approximate graph similarity score. Investigating the performance of embeddings generated by position-aware

and other embedding models can provide valuable understanding and insights into the effectiveness of the methods and their applicability to different graph-related tasks. Notably, GraphReach employs fixed anchors generated via random walks. This can potentially yield dissimilar anchor sets for similar graph pairs. This, in turn, can result in non-alignable node embeddings. Nonetheless, the anchor selection strategy implemented by GraphReach, grounded in the 'most visited' and 'maximal coverage' principles, is engineered to mitigate disparities in node embeddings. Further, it is worth emphasizing that a substantial portion of graph similarity GNN models predominantly serve as GED and MCS solvers. It would be intriguing to explore the potential of employing statistical significance measures as an alternative metric for evaluating similarity. In such a situation, the model can be trained on the statistical significance of a pair of graphs computed with methodology similar to VerSaChI, ChiSeL or VELSET-NAGA (Dutta et al., 2017). Other avenues that can be explored are the use of statistical significance of a pair of nodes and transforming the observed vector obtained for a pair of nodes in VerSaChI into vector representation for cross-graph matching. It holds significant promise to investigate the potential of integrating statistical significance measures, into the neural architectures discussed earlier. Exploring these possibilities could open new avenues for enhancing the performance and capabilities of these neural architectures.

Publications

1. “ChiSeL: Graph Similarity Search using Chi-Squared Statistics in Large Probabilistic Graphs”, Shubhangi Agarwal, Sourav Dutta and Arnab Bhattacharya, Proceedings of the International Conference on Very Large Data Bases (VLDB), 2020, pages 1654-1668, Japan.

2. “GraphReach: Position-Aware Graph Neural Network using Reachability Estimations”, Sunil Nishad, Shubhangi Agarwal, Arnab Bhattacharya and Sayan Ranu, Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), 2021, pages 1527-1533, Canada.

3. “VerSaChI: Finding Statistically Significant Subgraph Matches using Chebyshev’s Inequality”, Shubhangi Agarwal, Sourav Dutta and Arnab Bhattacharya, Proceedings of the International Conference on Information and Knowledge Management (CIKM), 2021, pages 2812-2816, Australia.

4. “VeNoM: Approximate Subgraph Matching with Enhanced Neighbourhood Structural Information”, Shubhangi Agarwal, Sourav Dutta and Arnab Bhattacharya, 7th Joint International Conference on Data Science and Management of Data (CODS-COMAD), 2024.

References

Agarwal, S., Dutta, S., and Bhattacharya, A. (2020). ChiSeL: Graph Similarity Search using Chi-Squared Statistics in Large Probabilistic Graphs. *PVLDB*, 13(10):1654–1668.

Agarwal, S., Dutta, S., and Bhattacharya, A. (2021). VerSaChI: Finding Statistically Significant Subgraph Matches using Chebyshev's Inequality. In *CIKM*, pages 2812–2816. ACM.

Aggarwal, C. C. and Wang, H. (2010). Graph Data Management and Mining: A Survey of Algorithms and Applications. *Advances in Database Systems*, 40:13–68.

Albert, R. and Barabási, A.-L. (2002). Statistical mechanics of complex networks. *Rev. Mod. Phys.*, 74:47–97.

Anwar, M., Hassanien, A. E., Snášel, V., and Basha, S. H. (2022). Subgraph query matching in multi-graphs based on node embedding. *Mathematics*, 10(24):4830.

Armiti, A. and Gertz, M. (2014). Geometric Graph Matching and Similarity: A Probabilistic Approach. In *International Conference on Scientific and Statistical Database Management (SSDBM)*, pages 1–12.

Arora, A., Sachan, M., and Bhattacharya, A. (2014). Mining Statistically Significant Connected Subgraphs in Vertex Labeled Graphs. In *International Conference on Management of Data (SIGMOD)*, pages 1003–1014.

Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., and Ives, Z. (2007). Dbpedia: A nucleus for a web of open data. In *Proceedings of the 6th International The Semantic Web and 2Nd Asian Conference on Asian Semantic Web Conference*, ISWC'07/ASWC'07, pages 722–735, Berlin, Heidelberg. Springer-Verlag.

Babai, L. (2016). Graph Isomorphism in Quasipolynomial Time. In *Symposium on Theory of Computing (STOC)*, pages 684–697.

Bai, Y., Ding, H., Bian, S., Chen, T., Sun, Y., and Wang, W. (2019). Simgnn: A neural network approach to fast graph similarity computation. In *Proceedings of the twelfth ACM international conference on web search and data mining*, pages 384–392.

Bai, Y., Ding, H., Gu, K., Sun, Y., and Wang, W. (2020). Learning-based efficient graph similarity computation via multi-scale convolutional set matching. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 3219–3226.

Barcelo, P., Libkin, L., and Reutter, J. L. (2011). Querying Graph Patterns. In *Symposium on Principles of Database Systems (PODS)*, pages 199–210.

Baskararaja, G. R., Manickavasagam, M. S., et al. (2012). Subgraph matching using graph neural network. *Journal of Intelligent Learning Systems and Applications*, 4(04):274.

Bejerano, G., Friedman, N., and Tishby, N. (2004). Efficient Exact p-value Computation for Small Sample, Sparse and Surprisingly Categorical Data. *Journal of Computational Biology*, 11(5):867–886.

Bi, F., Chang, L., Lin, X., Qin, L., and Zhang, W. (2016). Efficient Subgraph Matching by Postponing Cartesian Products. In *SIGMOD*, pages 1199–1214.

Boldi, P., Bonchi, F., Gionis, A., and Tassa, T. (2012). Injecting Uncertainty in Graphs for Identity Obfuscation. *PVLDB*, 5(11):1376–1387.

Borgwardt, K. M., Ong, C. S., Schönauer, S., Vishwanathan, S., Smola, A. J., and Kriegel, H.-P. (2005). Protein function prediction via graph kernels. *Bioinformatics*, 21:i47–i56.

Bourgain, J. (1985). On lipschitz embedding of finite metric spaces in hilbert space. *Israel Journal of Mathematics*, 52(1-2):46–52.

Brin, S. and Page, L. (1998). The anatomy of a large-scale hypertextual web search engine. *Computer Networks*, 30:107–117.

Caetano, T. S., McAuley, J. J., Cheng, L., Le, Q. V., and Smola, A. J. (2009). Learning graph matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(6):1048–1058.

Chandrasekar, R. (2014). Elementary? Question Answering, IBM's Watson, and the Jeopardy! Challenge. *Resonance*, 19:222–241.

Chang, H., Rong, Y., Xu, T., Huang, W., Zhang, H., Cui, P., Zhu, W., and Huang, J. (2020). A restricted black-box adversarial framework towards attacking graph embedding models. In *AAAI*, pages 3389–3396.

Chen, C., Yan, X., Philip, S. Y., Han, J., Zhang, D. Q., and Gu, X. (2007). Towards Graph Containment Search and Indexing. In *PVLDB*, pages 926–937.

Chen, W., Liu, J., Chen, Z., Tang, X., and Li, K. (2018). Pbsm: An efficient top-k subgraph matching algorithm. *International Journal of Pattern Recognition and Artificial Intelligence*, 32(06):1850020.

Chen, Y., Zhao, X., Lin, X., Wang, Y., and Guo, D. (2019). Efficient Mining of Frequent Patterns on Uncertain Graphs. *Transactions on Knowledge and Data Engineering*, 31(2):287–300.

Cheng, J., Ke, Y., Ng, W., and Lu, A. (2007). FG-Index: Towards Verification-free Query Processing on Graph Databases. In *SIGMOD*, pages 857–872.

Conte, D., Foggia, P., Sansone, C., and Vento, M. (2004). Thirty Years of Graph Matching in Pattern Recognition. *International Journal of Pattern Recognition and Artificial Intelligence*, 18(3):265–298.

Cook, S. A. (1971). The Complexity of Theorem-proving Procedures. In *STOC*, pages 151–158.

Cordella, L. P., Foggia, P., Sansone, C., and Vento, M. (2004). A (sub)graph Isomorphism Algorithm for Matching Large Graphs. *Transactions on Pattern Analysis and Machine Intelligence*, 26(10):1367–1372.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2009). *Introduction to Algorithms,*. The MIT Press, 3rd edition.

de Tchébychef, P. (1867). Des valeurs moyennes. *Journal de Mathématiques Pures et Appliquées*, 12:177–184.

de Virgilio, R., Maccioni, A., and Torlone, R. (2015). Approximate Querying of RDF Graphs via Path Alignment. *Distributed Parallel Databases*, 33:555–581.

Dinari, H. (2017). A survey on graph queries processing: techniques and methods. *International Journal of Computer Network and Information Security*, 9(4):48.

Doan, K. D., Manchanda, S., Mahapatra, S., and Reddy, C. K. (2021). Interpretable graph similarity computation via differentiable optimal alignment of node embeddings. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 665–674.

Dutta, S. (2015). MIST: Top-k Approximate Sub-string Mining Using Triplet Statistical Significance. In *European Conference on Information Retrieval (ECIR)*, pages 284–290.

Dutta, S. and Bhattacharya, A. (2010). Most Significant Substring Mining based on Chi-square Measure. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD)*, pages 319–327.

Dutta, S. and Bhattacharya, A. (2012). Mining Statistically Significant Substrings Based on the Chi-Square Measure. In *Pattern Discovery Using Sequence Data Mining: Applications and Studies*, pages 73–82. IGI Global.

Dutta, S. and Lauri, J. (2019). Finding a Maximum Clique in Dense Graphs via Chi-Square Statistics. In *International Conference on Information and Knowledge Management (CIKM)*, pages 2421–2424.

Dutta, S., Nayek, P., and Bhattacharya, A. (2017). Neighbor-Aware Search for Approximate Labeled Graph Matching using the Chi-Square Statistics. In *International Conference on World Wide Web (WWW)*, pages 1281–1290.

Gallagher, B. (2006). Matching Structure and Semantics: A Survey on Graph-based Pattern Matching. In *AAAI*, pages 45–53.

Gao, Y., Miao, X., Chen, G., Zheng, B., Cai, D., and Cui, H. (2017). On Efficiently Finding Reverse k-nearest Neighbors over Uncertain Graphs. *The VLDB Journal*, 26:467–492.

Giles, C. L., Bollacker, K. D., and Lawrence, S. (1998). Citeseer: An automatic citation indexing system. In *Proceedings of the third ACM conference on Digital libraries*, pages 89–98.

Giugno, R. and Shasha, D. (2002). GraphGrep: A Fast and Universal Method for Querying Graphs. In *ICPR*, pages 112–115.

Grover, A. and Leskovec, J. (2016). node2vec: Scalable feature learning for networks. In *ACM SIGKDD*, pages 855–864.

Gu, Y., Gao, C., Wang, L., and Yu, G. (2016). Subgraph Similarity Maximal All-matching over a Large Uncertain Graph. In *International Conference on World Wide Web (WWW)*, pages 755–782.

Gulyás, A., Heszberger, Z., Bíró, J., Tapolcai, J., Csoma, A., Pelle, I., Kőrösi, A., Klajbár, D., Halasi, V., Rétvári, G., and Novák, M. (2018). A dataset on human navigation strategies in foreign networked systems. *Scientific Data*, 5.

Hall, P. (1983). Chi-Squared Approximations to the Distribution of a Sum of Independent Random Variables. *The Annals of Probability*, 11(4):1028–1036.

Hamilton, W., Ying, Z., and Leskovec, J. (2017). Inductive representation learning on large graphs. In *NIPS 2017*, pages 1024–1034.

Han, M., Kim, H., Gu, G., Park, K., and Han, W. (2019). Efficient Subgraph Matching: Harmonizing Dynamic Programming, Adaptive Matching Order, and Failing Set Together. In *SIGMOD*, pages 1429–1446.

Han, S., Peng, Z., and Wang, S. (2014). The Maximum Flow Problem of Uncertain Network. *Information Sciences*, 265:167–175.

Han, W. S., Lee, J., Pham, M. D., and Yu, J. X. (2010). iGraph: A Framework for Comparisons of Disk-based Graph Indexing Techniques. *PVLDB*, 3(1-2):449–459.

Hintsanen, P. and Toivonen, H. (2008). Finding Reliable Subgraphs from Large Probabilistic Graphs. In *International Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD)*, pages 3–23.

Hua, M. and Pei, J. (2010). Probabilistic Path Queries in Road Networks: Traffic Uncertainty Aware Path Selection. In *International Conference on Extending Database Technology (EDBT)*, pages 347–358.

Jia, Y., Zhang, J., and Huan, J. (2011). An Efficient Graph-mining Method for Complicated and Noisy Data with Real-world Applications. *Knowledge and Information Systems*, 28(2):423–447.

Jiang, H., Wang, H., Yu, P. S., and Zhou, S. (2007). GString: A Novel Approach for Efficient Search in Graph Databases. In *ICDE*, pages 566–575.

Jiang, R., Tu, Z., Chen, T., and Sun, F. (2006). Network Motif Identification in Stochastic Networks. *Proceedings of the National Academy of Sciences*, 103(25):9404–9409.

Jin, R., Liu, L., Ding, B., and Wang, H. (2011). Distance-constraint Reachability Computation in Uncertain Graphs. *PVLDB*, 4(9):551–562.

Jüttner, A. and Madarasi, P. (2018). VF2++: An Improved Subgraph Isomorphism Algorithm. *Discrete Applied Mathematics*, 242:69–81.

Kassiano, V., Gounaris, A., Papadopoulos, A. N., and Tsichlas, K. (2016). Mining Uncertain Graphs: An Overview. In *International Symposium on Algorithmic Aspects of Cloud Computing (ALGOCLOUD)*, pages 87–116.

Kelley, B. P., Yuan, B., Lewitter, F., Sharan, R., Stockwell, B., and Ideker, T. (2004). PathBLAST: A Tool for Alignment of Protein Interaction Networks. *Nucleic Acids Research*, 32:83–88.

Khan, A. and Chen, L. (2015). On Uncertain Graphs Modeling and Queries. *PVLDB*, 8(12):2042–2043.

Khan, A., Wu, Y., Aggarwal, C. C., and Yan, X. (2013). Nema: Fast graph search with label similarity. *Proc. VLDB Endow.*, 6(3).

Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. In *3rd ICLR*.

Kipf, T. N. and Welling, M. (2017). Semi-supervised classification with graph convolutional networks. In *5th ICLR*.

Kollios, G., Potamias, M., and Terzi, E. (2011). Clustering Large Probabilistic Graphs. *Transactions of Knowledge and Data Engineering*, 25(2):325–336.

Kpodjedo, S., Galinier, P., and Antoniol, G. (2014a). Using Local Similarity Measures to Efficiently Address Approx. Graph Matching. *Discrete Appl. Maths.*, 164:161–177.

Kpodjedo, S., Galinier, P., and Antoniol, G. (2014b). Using local similarity measures to efficiently address approximate graph matching. *Discrete Applied Mathematics*, 164:161–177.

Lan, Z., Yu, L., Yuan, L., Wu, Z., Niu, Q., and Ma, F. (2021). Sub-gmn: The subgraph matching network model. *arXiv preprint arXiv:2104.00186*.

Larrosa, J. and Valiente, G. (2002). Constraint Satisfaction Algorithms for Graph Pattern Matching. *Mathematical Structures in Computer Science*, 12(4):403–422.

Leskovec, J. and Faloutsos, C. (2006). Sampling from Large Graphs. In *Conference on Knowledge Discovery and Data Mining (KDD)*, pages 631–636.

Leskovec, J., Kleinberg, J., and Faloutsos, C. (2007). Graph evolution: Densification and shrinking diameters. *ACM TKDD*.

Li, G., Yan, L., and Ma, Z. (2019a). An Approach for Approximate Subgraph Matching in Fuzzy RDF Graph. *Fuzzy Sets and Systems*, 376:106–126.

Li, J., Zhang, H., Han, Z., Rong, Y., Cheng, H., and Huang, J. (2020). Adversarial attack on community detection by hiding individuals. In *WWW 2020*, page 917–927.

Li, J., Zou, Z., and Gao, H. (2012). Mining Frequent Subgraphs over Uncertain Graph Databases under Probabilistic Semantics. *The VLDB Journal*, 21(6):753–777.

Li, X., Cheng, R., Fang, Y., Hu, J., and Maniu, S. (2018). Scalable Evaluation of k-NN Queries on Large Uncertain Graphs. In *International Conference on Extending Database Technology (EDBT)*, pages 181–192.

Li, Y., Gu, C., Dullien, T., Vinyals, O., and Kohli, P. (2019b). Graph matching networks for learning the similarity of graph structured objects. In *International conference on machine learning*, pages 3835–3845. PMLR.

Lian, X., Chen, L., and Wang, G. (2016). Quality-Aware Subgraph Matching Over Inconsistent Probabilistic Graph Databases. *Transactions on Knowledge and Data Engineering*, 28(6):1560–1574.

Liang, Z., Xu, M., Teng, M., and Niu, L. (2006). NetAlign: A Web-based Tool for Comparison of Protein Interaction Networks. *Bioinfomatics*, 22(17):2175–2177.

Liu, C., Li, X., Zhao, D., Guo, S., Kang, X., Dong, L., and Yao, H. (2019a). A-gnn: Anchors-aware graph neural networks for node embedding. In *International Conference on Heterogeneous Networking for Quality, Reliability, Security and Robustness*, pages 141–153. Springer.

Liu, L., Du, B., Xu, J., and Tong, H. (2019b). G-Finder: Approximate Attributed Subgraph Matching. In *International Conference on Big Data*, pages 513–522.

Liu, L., Jin, R., Aggrawal, C., and Shen, Y. (2012). Reliable Clustering on Uncertain Graphs. In *International Conference on Data Mining (ICDM)*, pages 459–468.

Liu, X., Pan, H., He, M., Song, Y., Jiang, X., and Shang, L. (2020). Neural subgraph isomorphism counting. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1959–1969.

London, T. (2017). Rolling Origin and Destination Survey.

Lou, Z., You, J., Wen, C., Canedo, A., Leskovec, J., et al. (2020). Neural subgraph matching. *arXiv preprint arXiv:2007.03092*.

Mahmood, A., Farooq, H., and Ferzund, J. (2017). Large Scale Graph Matching (LSGM): Techniques, Tools, Applications and Challenges. *International Journal of Advanced Computer Science and Applications*, 8(4):494–499.

Maniu, S., Cheng, R., and Senellart, P. (2017). An Indexing Framework for Queries on Probabilistic Graphs. *Transactions on Database Systems*, 42(2):1–34.

McCallum, A. K., Nigam, K., Rennie, J., and Seymore, K. (2000). Automating the construction of internet portals with machine learning. *Information Retrieval*, 3(2):127–163.

Mongiovi, M., Di Natale, R., Guigno, R., Pulvirenti, A., Ferro, A., and Sharan, R. (2010). SIGMA: A Set-Cover-Based Inexact Graph Matching Algo. *Journal of Bioinformatics and Computational Biology*, 8(2):199–218.

Moustafa, W. E., Kimmig, A., Deshpande, A., and Getoor, L. (2014). Subgraph Pattern Matching over Uncertain Graphs with Identity Linkage Uncertainty. In *International Conference on Data Engineering (ICDE)*, pages 904–915.

Nemhauser, G. L., Wolsey, L. A., and Fisher, M. L. (1978). An analysis of approximations for maximizing submodular set functions-I. *Mathematical Programming*, 14(1):265–294.

Pan, J.-Y., Yang, H.-J., Faloutsos, C., and Duygulu, P. (2004). Automatic multimedia cross-modal correlation discovery. In *ACM SIGKDD*, pages 653–658.

Papapetrou, O., Ioannou, E., and Skoutas, D. (2011). Efficient Discovery of Frequent Subgraph Patterns in Uncertain Graph Databases. In *International Conference on Extending Database Technology (EDBT)*, pages 355–366.

Perozzi, B., Al-Rfou, R., and Skiena, S. (2014). DeepWalk: Online learning of social representations. In *ACM SIGKDD*, pages 701–710.

Potamias, M., Bonchi, F., Gionis, A., and Kollios, G. (2010). K-nearest Neighbors in Uncertain Graphs. *PVLDB*, 3(1-2):997–1008.

Read, T. and Cressie, N. (1989). Pearson's χ^2 and the Likelihood Ratio Statistic G^2: A Comparative Review. *International Statistical Review*, 57(1):19–43.

Read, T. R. C. and Cressie, N. A. C. (1988). *Goodness-of-fit Statistics for Discrete Multivariate Data*. Springer.

Rossi, R. A. and Ahmed, N. K. (2015). The Network Data Repository with Interactive Graph Analytics and Visualization. In *AAAI*, pages 4292–4293.

Roy, I., Velugoti, V. S. B. R., Chakrabarti, S., and De, A. (2022). Interpretable neural subgraph matching for graph retrieval. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 8115–8123.

Sachan, M. and Bhattacharya, A. (2012). Mining Statistically Significant Substrings using the Chi-Square Statistic. *PVLDB*, 5(10):1052–1063.

Samet, H. (2006). *Foundations of multidimensional and metric data structures*. Academic Press.

Shang, H., Zhang, Y., Lin, X., and Yu, J. (2008). Taming Verification Hardness: An Efficient Algorithm for Testing Subgraph Isomorphism. *PVLDB*, 1(1):364–375.

Shen, R. and Guda, C. (2014). Applied Graph-Mining Algorithms to Study Biomolecular Interaction Networks. *BioMed Research International: Big Data and Network Biology*, 2014.

Singh, R., Xu, J., and Berger, B. (2008). Global Alignment of Multiple Protein Interaction Networks with Application to Functional Orthology Detection. *PNAS*, 105(35):12763–12768.

Suchanek, F. M., Kasneci, G., and Weikum, G. (2007). YAGO: A Core of Semantic Knowledge. In *International Conference on World Wide Web (WWW)*, pages 697–706.

Sun, H., Zhou, W., and Fei, M. (2020). A survey on graph matching in computer vision. In *13th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI)*, pages 225–230.

Sun, S. and Luo, Q. (2019). Scaling up subgraph query processing with efficient subgraph matching. In *ICDE*, pages 220–231.

Tian, Y., McEachin, R., Santos, C., States, D., and Patel, J. (2006). SAGA: A Subgraph Matching Tool for Biological Graphs. *Bioinformatics*, 23(2):232–239.

Tian, Y. and Patel, J. (2008). TALE: A Tool for Approximate Large Graph Matching. In *ICDE*, pages 963–972.

Tong, H. and Faloutsos, C. (2006). Center-piece Subgraphs: Prob. Defn. and Fast Solutions. In *Conference on Knowledge Discovery and Data Mining (KDD)*, pages 404–413.

Tong, H., Faloutsos, C., Gallagher, B., and Eliassi-Rad, T. (2007). Fast Best-effort Pattern Matching in Large Attributed Graphs. In *Conference on Knowledge Discovery and Data Mining (KDD)*, pages 737–746.

TransStat, R. (2016). Origin and Destination Survey database (DB1B).

Tsai, W. H. and Fu, K. (1979). Error-correcting Isomorphisms of Attributed Relational Graphs for Pattern Recognition. *Transactions on Systems, Man, and Cybernetics*, 9(12):757–768.

Tversky, A. (1977). Features of Similarity. *Psychological Review*, 84(4):327–352.

Ullmann, J. R. (1976). An Algorithm for Subgraph Isomorphism. *JACM*, 23(1):31–42.

Valiant, L. G. (1979). The Complexity of Enumeration and Reliability Prob. *J. of Computing*, 8(3):410–421.

Vandin, F., Upfal, E., and Raphael, B. J. (2011). Algorithms for Detecting Significantly Mutated Pathways in Cancer. *JCB*, 18(3):507–522.

Velickovic, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., and Bengio, Y. (2018). Graph attention networks. In *6th ICLR*.

Wadhwa, S., Prasad, A., Ranu, S., Bagchi, A., and Bedathur, S. (2019). Efficiently answering regular simple path queries on large labeled networks. In *ICMD*, pages 1463–1480.

Waikhom, L., Singh, Y., and Patgiri, R. (2023). Po-gnn: Position-observant inductive graph neural networks for position-based prediction. *Information Processing & Management*, 60(3):103333.

Wang, R., Zhang, T., Yu, T., Yan, J., and Yang, X. (2021). Combinatorial learning of graph edit distance via dynamic embedding. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5241–5250.

Watts, D. J. (1999). Networks, dynamics, and the small-world phenomenon. *American Journal of Sociology*, 105(2):493–527.

West, R. and Leskovec, J. (2012). Human wayfinding in information networks. In *WWW*, page 619–628.

Xiu, H., Yan, X., Wang, X., Cheng, J., and Cao, L. (2020). Hierarchical graph matching network for graph similarity computation. *arXiv preprint arXiv:2006.16551*.

Xu, C., Cui, Z., Hong, X., Zhang, T., Yang, J., and Liu, W. (2020). Graph inference learning for semi-supervised classification. In *8th ICLR*.

Xu, K., Hu, W., Leskovec, J., and Jegelka, S. (2019). How powerful are graph neural networks? In *7th ICLR*.

Yan, J., Yin, X.-C., Lin, W., Deng, C., Zha, H., and Yang, X. (2016). A short survey of recent advances in graph matching. In *ICMR*, page 167–174.

Yan, X., Yu, P. S., and Han, J. (2005a). Graph Indexing Based on Discriminative Frequent Structure Analysis. *Transactions on Database Systems*, 30(4):960–993.

Yan, X., Yu, P. S., and Han, J. (2005b). Substructure Similarity Search in Graph Databases. In *SIGMOD*, pages 766–777.

You, J., Ying, R., and Leskovec, J. (2019). Position-aware graph neural networks. In *ICML*, pages 7134–7143.

Yu, X., Sun, Y., Zhao, P., and Han, J. (2012). Query-driven Discovery of Semantically Similar Substructures in Heterogeneous Networks. In *Conference on Knowledge Discovery and Data Mining (KDD)*, pages 1500–1503.

Yuan, Y., Wang, G., Chen, L., and Ning, B. (2016). Efficient Pattern Matching on Big Uncertain Graphs. *Information Sciences*, 339:369–394.

Yuan, Y., Wang, G., Chen, L., and Wang, H. (2012). Efficient Subgraph Similarity Search on Large Probabilistic Graph Databases. *PVLDB*, 5(9):800–811.

Yuan, Y., Wang, G., Chen, L., and Wang, H. (2015). Graph Similarity Search on Large Uncertain Graph Databases. *The VLDB Journal*, 24(2):271–296.

Yuan, Y., Wang, G., Wang, H., and Chen, L. (2011). Efficient Subgraph Search over Large Uncertain Graphs. *PVLDB*, 4(11):876–886.

Zeng, Z., Tung, A. K. H., Wang, J., Feng, J., and Zhou, L. (2009). Comparing Stars: On Approximating Graph Edit Distance. *PVLDB*, 2(1):25–36.

Zhang, S., Li, J., Gao, H., and Zou, Z. (2009a). A Novel Approach for Efficient Supergraph Query Processing on Graph Databases. In *International Conference on Extending Database Technology (EDBT)*, pages 204–215.

Zhang, S., Li, S., and Yang, J. (2009b). GADDI: Distance Index Based Subgraph Matching in Biological Networks. In *International Conference on Extending Database Technology (EDBT)*, pages 192–203.

Zhang, S., Yang, J., and Jin, W. (2010). SAPPER: Subgraph Indexing and Approximate Matching in Large Graphs. *PVLDB*, 3(1-2):1185–1194.

Zhang, Z., Bu, J., Ester, M., Li, Z., Yao, C., Yu, Z., and Wang, C. (2021). H2mn: Graph similarity learning with hierarchical hypergraph matching networks. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 2274–2284.

Zhao, P. and Han, J. (2010). On Graph Query Optimization in Large Networks. *PVLDB*, 3(1-2):340–351.

Zhu, J., Yan, Y., Zhao, L., Heimann, M., Akoglu, L., and Koutra, D. (2020). Beyond homophily in graph neural networks: Current limitations and effective designs. In *NeurIPS*.

Zitnik, M. and Leskovec, J. (2017). Predicting multicellular function through multi-layer tissue networks. *Bioinformatics*, 33(14):i190–i198.

Zou, L., Chen, L., and Lu, Y. (2007). Top-k Subgraph Matching Query in a Large Graph. In *PhD Workshop: International Conference on Information and Knowledge Management (CIKM)*, pages 139–146.

Zou, L., Chen, L., Yu, J. X., and Lu, Y. (2008). A Novel Spectral Coding in a Large Graph Database. In *International Conference on Extending Database Technology (EDBT)*, pages 181–192.

Zou, Q., Liu, S., and Chu, W. W. (2004). CTree: A Compact Tree for Indexing XML Data. In *WIDM*, pages 39–46.

Zou, Z., Gao, H., and Li, J. (2010). Discovering Frequent Subgraphs over Uncertain Graph Databases under Probabilistic Semantics. In *Conference on Knowledge Discovery and Data Mining (KDD)*, pages 633–642.