1-1-2014

On Ranked Approximate Matching Of Large Attributed Graphs

Mohammad Shafkat Amin
Wayne State University,

ON RANKED APPROXIMATE MATCHING OF LARGE ATTRIBUTED GRAPHS

by

MOHAMMAD SHAFKAT AMIN

DISSERTATION

Submitted to the Graduate School

of Wayne State University,

Detroit, Michigan

in partial fulfillment of the requirements

for the degree of

DOCTOR OF PHILOSOPHY

2014

MAJOR: COMPUTER SCIENCE

Approved by:

Advisor Date

DEDICATION

Dedicated to my wife Fahima Amin Bhuyan and my son Faaris Shadmehr Amin

ACKNOWLEDGMENTS

I would first like to thank the Almighty for giving me this opportunity to pursue my PhD. I also would like to express my deep and sincere gratitude to my advisor, Dr. Hasan Jamil, for his constant encouragement, support, and guidance in every time of need throughout my Ph.D. program. It was his faith in me, that kept me going through challenging times. In addition, I am grateful to my dissertation committee members: Dr. Russ Finley Jr. for instilling in me the value of excellence in research and for numerous brains-storming sessions, Dr. Chandan Reddy for introducing me to the exciting world of data mining and Dr. Zaki Malik for his support and invaluable comments.

I would like to express my gratitude to Anupam Bhattacharjee, who we have lost at a young age, for being an inspiration to us and epitomizing dedication in research. I also would like to thank my academic colleagues Dr. Emdad Hossain, Dr. Shazzad Hossain, Munirul Islam, Aminul Islam, Kazi Zakia Sultana, Saikat Dey and Shahriyar Hossain for their academic cooperation and close friendship.

I am indebted to my wife Fahima Bhuyan and my son Faaris Amin for always being there for me. It was only because of the sacrifices that they have made, encouragement they have offered through out, that I am able to complete my program. My special gratitude goes to my father Aminul Islam and mother Shireen Akhter, brothers Anuj, Ananno and sister Auditi for their unconditional love and good wishes.

TABLE OF CONTENTS

Dedication . . . . **ii**

Acknowledgments . . . . **iii**

List of Tables . . . . **vii**

List of Figures . . . . **viii**

Chapter 1: INTRODUCTION **1**

1.1 Motivation and Application 3

1.2 Goal of this dissertation 5

1.3 Organization of the dissertation 6

Chapter 2: RELATED WORK **7**

Chapter 3: PROBLEM DESCRIPTION **10**

3.1 Basic terminology 10

3.2 Similarity matching and Exact matching 12

3.3 Our contribution 13

Chapter 4: TOP-*K* GRAPH MATCHING USING TraM **15**

4.1 Problem Definition 15

4.1.1 Random Walk as a Global Graph Property 16

4.1.2 Structural Similarity of Nodes 17

4.1.3 Combined Graph Similarity 19

4.2 Overview of TraM 20

4.3 Top-*k* Graph Matching using Neighborhood Signatures 22

4.3.1 TraM Matching Algorithm 23

4.3.2 Caching Local Neighborhood 26
4.3.3 TraM Properties 26
4.4 Edge-labeled Graphs 29

Chapter 5: TOP-*K* GRAPH MATCHING USING ATOM **35**
5.1 Problem Definition and the Semantics of Matching 39
5.1.1 Graph Similarity 41
5.2 Top-*k* Graph Matching in AtoM 42
5.2.1 Step One: Summarization 42
5.2.2 Step Two: Validation 47
5.2.3 Step Three: Extraction 53
5.2.4 The overall algorithm 57
5.3 Accounting for Edge Similarity 59
5.4 Pattern Querying 59
5.5 AtoM in Map-Reduce Framework 60

Chapter 6: EXPERIMENTAL RESULTS **62**
6.1 Experimental Results: TraM 62
6.1.1 Precision and Recall 64
6.1.2 Scalability 65
6.1.3 Effectiveness Analysis 66
6.2 Experimental Results: AtoM 68
6.2.1 Precision and Recall 70
6.2.2 Scalability 70

Chapter 7: CONCLUSIONS AND FUTURE WORK **72**

Appendix A **73**
7.1 Comparison with SAGA 73
7.2 Result on KEGG Data set 73

Appendix B **77**

7.3 EXTRACTING DISEASE SUB-NETWORK USING SEED DISEASE GENES . 77

7.3.1 Method and Implementation 77

7.3.2 Evaluation 83

Appendix C **86**

7.4 TRAM as a graph matching toolkit 86

References **91**

Abstract **105**

Autobiographical Statement **107**

LIST OF TABLES

Table 1.1 Application of Approximate Sub-graph Matching. 5

Table 4.1 Inclusion of edge weight in PPI network 32

Table 5.1 Example Similarity Matrix 57

Table 5.2 Partial Matching result on DBLP data set 60

Table 7.1 Partial Matching result on literature mined data set 75

Table 7.2 Partial matching result on KEGG database 76

LIST OF FIGURES

Figure 3.1 Classification of Graph Matching 11
Figure 3.2 Graph matching versus subgraph isomorphism. 13

Figure 4.1 Example of two different graphs with identical β-signature. 18
Figure 4.2 β-signature, similarity generation for query and localized data graph. . . . 20
Figure 4.3 Extracting matching sub-graphs from the localized data graph. 22
Figure 4.4 Example of matching with weighted edge. 29
Figure 4.5 Sub-graph matching with missing nodes 33
Figure 4.6 Protein-protein interaction query graph. 34

Figure 5.1 Querying a social network database. 36
Figure 5.2 Steps involved in the matching process. 38
Figure 5.3 Example showing matching process of AtoM. 43
Figure 5.4 An alternate summary and matched graph. 46

Figure 6.1 Performance Evaluation on various data graph sizes. 63
Figure 6.2 Precision-Recall Curves for (a) TraM and (b) C-Tree. 65
Figure 6.3 Performance Evaluation on PPI Data. 66
Figure 6.4 Performance Evaluation on KEGG. 67
Figure 6.5 Performance evaluation on various data and query graph sizes. 69
Figure 6.6 Precision-Recall Curves for (a) AtoM and (b) TraM 70
Figure 6.7 Comparing AtoM with TraM on large PPI data set. 71

Figure 7.1 Matching on SAGA dataset 74
Figure 7.2 Execution on a synthetic example 78
Figure 7.3 Performance measurement of the algorithm 82
Figure 7.4 Robustness of the algorithm 83

Figure 7.5 Main page of TRAM Tool 86
Figure 7.6 Example section of the tool 87
Figure 7.7 Result page of the tool 88
Figure 7.8 Data graph: DroiDB 89
Figure 7.9 Query Graph 89
Figure 7.10 Matched Result 89
Figure 7.11 Class diagram for TraM 90

CHAPTER 1: INTRODUCTION

Numerous real-life data, e.g., protein-protein interaction networks, social networks, and the world wide web can conveniently be represented by graphs. The size and complexity of these graphs in terms of the number of entities and their relationships is formidable. The database community has had a long-standing interest in querying graph databases [52, 87, 104, 108, 110, 111] since it can facilitate inferring knowledge from a myriad of data sources. Among the many graph database queries that are useful, graph matching has proven to be a powerful tool for knowledge extraction. Previous approaches have mostly been carried out within the context of exact graph data, and have concentrated on precise graph or subgraph matching queries. However, many actual graph data sets are noisy and inherently incomplete. For example, it is well known that high-throughput protein-protein interaction (PPI) data contain numerous false positives. Moreover, the interactions identified represent only a fraction of the actual network [43, 98]. As a result, exact graph or subgraph matching more often than not fails to produce any useful outcome.

Approximate or inexact graph or subgraph matching methods can play a crucial role in applications where exact graph matching is not useful or possible. Many recent graph applications demand approximate matching results as they are more intuitive and can provide more information such as predicting spurious and missing entities. Additionally, most existing graph matching methods can be applied to databases that contain graphs with small sizes. Moreover, the query graphs allowed in these methods are also small in size. In many new applications, however, the sizes of both the query graph and the data graph can be prohibitively large for existing matching methods. Each graph can contain hundreds to thousands of nodes and edges. For example, PPI networks for individual species are often matched to determine similarities and differences across species. Each PPI network is large, and typically contains hundreds to thousands of nodes and edges in each graph. With the exponential growth of social networks (e.g., LinkedIn, Facebook, etc.) and significantly increased influx of experimentally generated data (e.g., microarray, protein-protein interaction network, etc.), the need for storage, management and querying of large graphs has increased significantly. Graphs naturally help model many complex objects and applications of high importance in social science, engineering and life sciences. Although the computational complexity of graph matching algorithms are generally very high, traditional algorithms such as Ullmann [96] and VFLib [29] remained state of the art in many libraries such as R [77] for databases with relatively very small graphs (e.g., chemical structures, image data, pattern recognition applications, etc.). The emergence of modern, highly connected, complex, large and internet scale graph data generated significant interests toward developing better and faster algorithms to meet the challenges they posed. The sheer volume of social networks, internet scale graphs, inherent inaccuracies in experimentally generated data, and high frequency of queries in public databases in life sciences also make traditional graph processing techniques obsolete.

Among many graph based queries, graph and subgraph matching have been proven to have a significant role on knowledge extraction. However, it is generally believed that approximation algorithms help tame the computational hardness of subgraph isomorphic queries by avoiding the so called "permutation trap" in graphs with highly symmetric labels and edge structures, and high number of candidate matches [27]. Naturally, while a large majority of research concentrated on identifying exact matches, inexact and approximate queries are becoming increasingly more useful since they can inherently cater to inaccuracies in the input graph, and identification of similar rather than exact modules from the source data graph. Since it is well known that protein-protein interaction networks determined by high-throughput methods contain numerous false positives and the identified interactions merely represent a minute fraction of the actual network [93], exact graph or subgraph matching techniques more often then not fail to produce any useful outcome. Instead, approximate or inexact graph matching plays a crucial role in such applications. Many modern applications also prefer approximate matching over precise matching as they are more intuitive and can provide more insight about spurious and missing entities.

From an application standpoint in social networks [33, 34, 49], biological networks [31, 64, 114], image databases [58, 66], and even geographical and remote sensing information systems [51, 94], approximate graph matching as a tool for graph analysis has seen substantial renewed interests. Social network applications that require modeling using large graphs, and involve complex descriptive attributes, approximate matching has been successfully used to identify expert communities [68], drug trafficking network [34], etc. Approximate graph matching has also been instrumental in life sciences to predict drug interactions [64], cancer detection [24], and phylogenetics [47], in software engineering [13, 81], and in patent enforcement and plagiarism detection [67, 79], among many others. In all such applications, scalability becomes a major concern due to the size of the graphs involved, e.g., LinkedIn, currently has more than 250 million users in its social graph, and traditional approaches to query these network are prohibitively expensive.

1.1 Motivation and Application

The motivation for developing an efficient approximate sub-graph matching technique can be attributed to numerous applications in the biological domain. Biological processes are controlled by genetic regulatory networks that consist of interacting groups of genes and gene products (i.e., RNAs and proteins). Protein-protein interaction networks serve as a good example of the importance of biological networks and the challenges associated with analyzing them. Protein-protein interactions play a role in most cellular processes, from synthesizing cellular components and determining cell structure to controlling cell behavior and the response to extracellular signals. Defining the PPIs that take place in a cell, therefore, can provide significant insights into the workings of a cell under normal conditions, and in disease states. Recent developments in experimental biotechnology and computational approaches are resulting in the generation of large datasets of PPIs [30, 95]. Tens of thousands of PPIs have been identified for human proteins and similar numbers have been identified for several model organisms. A useful way to represent and analyze this data is by network analysis approaches, where proteins are represented as the nodes and interactions are represented as the edges of a graph.

Biological networks such as PPI networks have several properties that present challenges for network analysis. First, the networks tend to be large, which necessitates the development of efficient analysis algorithms. One compiled database for humans, for example, has over 37,000 PPIs involving over 10,000 proteins [16]. Another database specific for the model organism Drosophila has over 160,000 interactions among $\sim$10,000 proteins, which includes over 70% of the proteins from that organism [72]. The data in these and similar databases is likely to continue growing as new technologies are developed to identify or predict additional interactions. Second, protein networks are incomplete. Current experimental technologies miss many interactions (i.e., result in false negatives) due to the practical difficulties associated with sampling all pair-wise combinations of large sets of proteins and the technical differences between the way proteins behave in the assays and in normal cells. Third, a related problem is that PPI data contain significant numbers of false positives, interactions that were detected by some assay or prediction algorithm but that do not actually occur in biology.

One way that protein network data can be used is to find proteins that are involved in specific biological processes or diseases. A protein with an unknown function, for example, can be predicted to play a role in the same process or disease as the proteins with which it interacts [40, 65, 80, 83]. Many variations of this so-called guilt-by-association (GBA) approach have been used to predict functions for a fraction of the huge number of proteins that have unknown or poorly characterized roles in biology. Several powerful variations of GBA utilize the structure or topology of protein networks to help predict function [99]. For example, it is often useful to consider not only the properties of a protein's direct interactors but also the properties of members of a local subnetwork around the protein [46].

The topological structures of protein networks also correlate with their biological functions. Tightly interconnected clusters of proteins, for example, are often indicative of groups of proteins that function together in a cell to mediate a specific process [12, 19, 20, 44, 88, 97]. The local network topology around a protein can also be used to impute properties of the protein such as its general importance in the cell or its role in disease pathways [25, 50, 55, 57, 62, 75]. Protein pathways that transduce information in cells also have characteristic topological properties [84, 89]. These and other functional implications of network structure have resulted in a need for computational tools that compare structures or that search through networks to find specific structures [85]. One example is the need to compare protein networks from different organisms in order to identify evolutionarily conserved functional modules [18,59,86,102,106]. Comparisons of network structures could also reveal missing nodes or edges and reveal new pathways. Thus, the need to compare and quantify the similarities between the structures of biological networks has brought the subgraph isomorphism problem to biology. Further details of specific applications of similar graph matching may be found in [35, 60, 71, 76, 114]. Table 1.1 lists different application domains and the related research.

Table 1.1: Application of Approximate Sub-graph Matching.

Application Domain	References
Protein-Protein interaction network	[6,7,10]
Social network	[33,34,49]
Image database	[36,58,66]
Geographical and remote sensing information systems	[51,94]
Expert communities identification	[68]
Drug trafficking network	[34]
Life sciences	[24,47]
Software engineering	[13,81]
Plagiarism detection	[67,79]

1.2 Goal of this dissertation

In this dissertation, we present a novel, scalable, efficient and approximate sub-graph matching technique that provides a flexible platform to augment topological information with domain knowledge to reduce search space and obtain context sensitive result. The problem that we address in this dissertation is approximate subgraph matching of a query graphs from a large data graph. Hence, given a graph database containing a single graph with numerous nodes and edges and a query graphs, the task here is to find all similar sub-structures from the data-graph matching the query graph.

In order to facilitate processing of larger graph, we introduce a. database driven processing with in memory cache to support memory resident post-processing b. Summarization of large data graph to cluster semantically meaningful nodes together, which in turn expedites processing. Since, many network data-sets inherently contain annotated information pertaining to each node, the algorithm makes use of this information to unveil the latent semantic information in each sub-network. We have conducted rigorous experiments and have proposed future directions to harness the power of map-reduce framework to make the algorithm scale out even further.

1.3 Organization of the dissertation

The rest of this thesis is organized as following. In chapter 2 we explore the literature and discuss the state of the art. We then identify the areas of improvement in the domain of approximate graph matching and underscore the motivation behind our thesis. In chapter 3, we formally define the problem. We build off of basic graph concept and informally define the problem of approximate graph matching. In chapter 4, we discuss our first approach to solving the problem. We build upon the the concepts presented in chapter 4 and discuss our second approach in chapter 5. In the next two chapters we present the findings of our experimental results comparing and contrasting our approach with the state of the art. Finally in chapter 7, we propose our future work and draw conclusion based on our research.

CHAPTER 2: RELATED WORK

Querying graph databases has been an important research area in the database community for a long time. However, although most of the previous works have concentrated on exact graph or subgraph matching, the applicability of inexact or approximate graph matching is undeniably crucial in numerous application domains such as bioinformatics, social network analysis and so on. Since subgraph isomorphism was proven to be NP complete [39], the application of approximate subgraph matching has attained immense research attention due to its reduced time requirement. The subgraph isomorphism technique proposed by Ullman [96] is proved to be prohibitively expensive due to its exponential time complexity. To reduce search space, indexing methods were developed to expedite the process [87, 108]. Most index-based methods that have been proposed for approximate graph matching, e.g. the methods proposed in [110] and [111] are mostly applicable to small graphs.

A method proposed in [104] is only applicable to graphs with limited sizes since it iteratively enumerates and indexes all the subgraphs in the database. GString [52] takes into account sequence based matching technique to extract answers to graph queries, but it is only useful when the graph contains a small set of basic substructures. C-Tree [45] utilizes an R-tree like index structure and is more generic in terms of answering graph queries. SAGA [92], employs a flexible graph similarity model but is computationally expensive when applied to large graphs. A recent method in [78] proposes a novel technique of embedding graphs in vector space by means of prototype selection. Subsequently, the vector representation of the graphs are used in edit distance calculation to calculate the distance between two graphs. However, selecting representative prototypes still remains a challenging task. The Cytoscape plugin tYNA [112] can carry out a query for extraction of four common motifs, whereas MAVisto [82] and FANMOD [103] identify over-represented and frequent motifs in a network specified by the user. However, in none of these approaches can a user structurally synthesize the sub-structure to be queried. There has been a flurry of research on approximate graph matching and graph pattern matching. A

bulk of research has been done on matching solely based on structures, without considering the attributes, while others have concentrated on exact matching. Many of the existing work focus primarily on matching against a database of small graphs rather than matching within a single large data graph. while the latter setting is more generic and can be tailored to accommodate graph-transactional setting, the converse is not true [61]. A survey of graph pattern matching algorithm can be found in [38] and a discussion on emerging graph queries can be found in [56]. Although a lot of work can be found in the literature on inexact graph matching [42, 70], attributed graph matching [26] and matching in the single graph setting [5,26,70], the combination of these trifecta has been rarely explored [28]. Additionally, a plethora of analogous work is done in the domain of database and data-mining research encompassing the area of frequent pattern identification to inexact query of databases. An efficient method for indexing and mining graph databases, computed based of existence of frequent graph structures has been proposed in [107, 109]. Moreover in [53], the concept of topological minor has been used to discover frequent large scale patterns. In order to process large graphs having thousands or even millions of nodes and edges, the concept of graph summarization has been studied. Most of the work employ simple graph statistics to describe graph characteristics [22, 23, 73]. These statistical measures may include degree distribution, clustering coefficients etc. Methods employed in mining graphs for frequent patterns [48, 100, 101] are also employed to unveil the properties of large graphs. These methods may generate an overwhelming number of results defying the purpose of summarization. Apart from that graph partitioning algorithms [74, 90, 105] have been employed in the literature to identify latent communities in large networks. This approach, however, does not take into account the node attributes and hence may limit the usefulness of the approach.
The methods proposed in this dissertation has the following salient features:

- Seamless integration of node attributes and topology in the matching process
- Database driven efficient and scalable method for processing large graphs to identify approximate matches. The data graph is maintained and the pre-calculations are executed in

the back-end database.This method segments the data graph into local neighborhoods and filters out the segments that are less likely to produce viable matches. Moreover, the global topological attribute associated with the nodes that we take into account in our method, corroborated by the dynamic matching algorithm is much faster in terms of processing and is not limited to graphs with limited sizes. Since a significant amount of processing is pushed to the database side along with memory resident processing of the segmented subsections of the data-graph, the graph database can essentially be as large as the current database technology will allow.

- Graph summarization driven processing, enabling identification of semantically significant matches. This method is extendible to map-reduce frame-work making it scalable and efficient.

CHAPTER 3: PROBLEM DESCRIPTION

3.1 Basic terminology

A primitive graph can be defined by $G = (V,E)$ consisting of vertices and edges. V denotes the set of vertices and $E \subseteq V \times V$ denotes the set of edges of G. The number of edges and number of vertices in a graph are often denoted by $|V|$ and $|E|$ respectively.

Definition 3.1.1 [Graph] A graph G is defined as $G = (V,E)$, where V is the set of vertices, and $E \subseteq V \times V$ is a set of vertex pairs that defines the edges of the graph.

We use graphs to model networks such as protein interaction, social network, world wide web, etc. The nodes of the graph can be attributed i.e. the nodes can have labels associated with them and they may also have attributes delineating their inherent properties.

Definition 3.1.2 [Labeled Graph] A labeled graph G is a four-tuple $G = (V,E,\mu,\tau)$, where

- V is the finite set of nodes,
- $E \subseteq VxV$ is the set of edges,
- $\mu : V \rightarrow L$ is the node labeling function, and
- $\tau : E \rightarrow L$ is the edge labeling function.

This definition allows us to handle arbitrary graphs with unconstrained labeling functions. For example, the labels can be given by the set of integers, the vector space $\mathbb{R}^n$ etc. Moreover, unlabeled graphs are obtained as a special case by assigning the same label to all nodes and edges.

The problem of graph matching can be viewed from two dimensions i.e. exact and inexact matching. To delineate exact graph matching problem we introduce the following definitions.

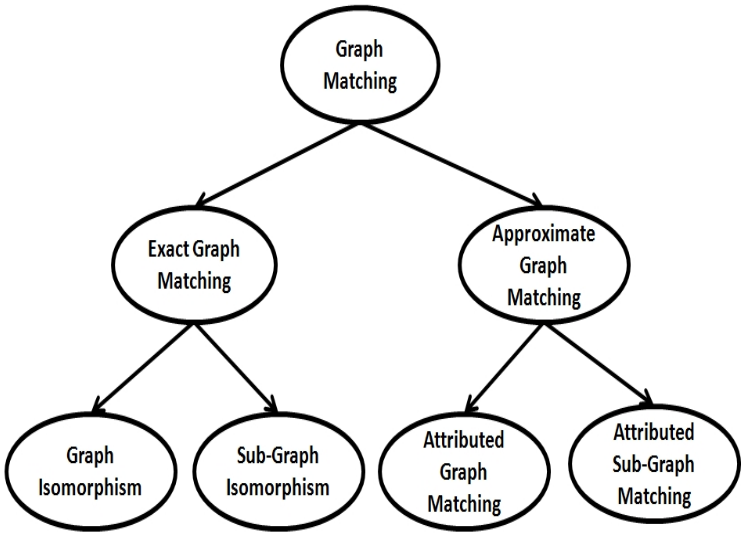

Graph Matching
Exact Graph Matching
Approximate Graph Matching
Graph Isomorphism
Sub-Graph Isomorphism
Attributed Graph Matching
Attributed Sub-Graph Matching

Figure 3.1: Classification of Graph Matching

Definition 3.1.3 [Sub-Graph] A graph $H = (V_H, E_H)$ is called a subgraph of $G = (V, E)$, if $V_H \subseteq V$ and $E_H \subseteq E$.

Definition 3.1.4 [Graph Isomorphism] Assume that we have two graphs $Q = (V_1, E_1)$ and $G = (V_2, E_2)$. Graph Q is isomorphic to graph G, if and only if there exists at least one bijective function $f : V_1 \rightarrow V_2$ such that for any edge $(u, v) \in E_1$, there is an edge $(f(u), f(v)) \in E_2$.

Definition 3.1.5 [Sub-graph Isomorphism] Assume that we have two graphs $Q(V_1, E_1)$ *and* $G(V_2, E_2)$. If there exists at least one sub-graph X in graph G, graph Q is isomorphic to X under the bijective function f, graph Q is sub-graph isomorphic to graph G.

However, there exists cases where it is not possible to find exact matches. This may stem from the fact that the input graphs might be noisy, the query graph might be generated from a template in mind, hence instead of exact graph matches, we adhere to approximate matches or graph similarities. We define the term graph similarity in more detail in the following chapters to elucidate the problem. We can use the term “similarity” loosely at this point to denote a spectrum of graph matches starting from exact matches to inexact matches. The flexibility inherent in approximate graph matching can be attributed to the noise tolerance of the matching process. The matching process allows missing nodes, edges as well as extra nodes and edges in the matched graphs. Since the graphs can have attributes, in other words, the nodes and edges can have labels or feature vectors associated with them, a central part in the domain of approximate graph matching has been in the direction of how to embed this rich set of information into vanilla topological matching system. In this dissertation, Given a large data graph D and a query graph Q, we explore ways to identify and extract top-k approximate matches from D conforming to the node attribute and topology of the query graph.

Definition 3.1.6 [Top-k graph matching] Given a data graph D and a query graph Q, the objective here is to identify k sub-graphs S_i from D such that the similarity between Q and each S_i, $1 \leq i \leq k$, are the maximum among all possible subgraphs of D and higher than a threshold λ.

In this dissertation, we concentrate on attributed subgraph matching problem. The attribute pertaining to nodes are captured using a domain dependent similarity function, called the σ-similarity, defined as $\sigma : V_d \times V_q \to [0,1]$, where $V_d \in D$ and $V_q \in Q$ which we assume as given as an input. We identify ways to use this domain value to guide us with matching process to identify candidate matches that are topologically similar as well. The heart of our approach lies in factoring the notion of domain similarity and topological similarity into calculation and seamlessly integrating these two dimensions.

3.2 Similarity matching and Exact matching

Similarity matching, commonly known as graph matching, problems are distinct from traditional graph isomorphism problems where not only do the structure of the graphs play a role, their labels do as well. To illustrate the difference, consider the graphs in figure 3.2 below in which the colors of nodes represent labels and the text next to them represent unique vertex IDs, i.e., query graph q in figure 3.2(a), node v_1 has black, v_2 has salmon and v_3 has cyan labels. Though the three data graphs d_1 through d_3 in figures 3.2(b) through 3.2(d) respectively contain similar shapes, they do not have the same labels. In these figures, the query graph q is subgraph isomorphic to graphs d_1

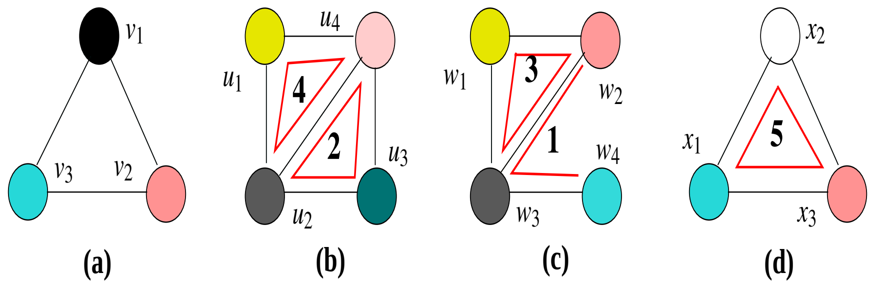


Figure 3.2: Graph matching versus subgraph isomorphism.

and d_2 (as they both contain structurally identical shapes – triangles), while it is isomorphic to graph d_3 (if we ignore the labels). If we consider the labels too, these relationships will not hold as no data graph or its subgraph matches the query graph q.

On the other hand, an approximate match is always possible even if we consider the labels, i.e., given two graphs, they are always similar. The real question is to what degree? For example, if we consider black and white labels to be completely dissimilar, then subgraph 1 with nodes w_2, w_3 and w_4 in figure 3.2(c) may be the most similar graph to query graph q even though it is not a triangle compared to other subgraphs including graph 5 in figure 3.2(d). This is because the labels are almost identical (except a lighter shade of black in node w_3). It is also possible to consider graph 2 with nodes u_2, u_3 and u_4 in figure 3.2(b) to be similar, if the label teal in node u_3 is considered truly similar to cyan. If we prefer to preserve the shape more than label similarity (unlike before, we rank graph 2 higher than graph 1 as it is structurally more similar). From this standpoint, it is perhaps possible to order the matching in decreasing similarity shown as graphs 1 through 5 in figure 3.2.

3.3 Our contribution

In this dissertation we systematically explore and integrate domain and topological information of graphs and propose a quantifiable measure of graph similarity matching. With that end in view, we propose two efficient algorithms to identify top-k sub-graphs that best match a query graph.

In one approach, TraM [9, 10], we identify promising neighborhoods from the data graph and explore it further to deem its candidacy to be a viable match with the query graph. In this approach, we combine global topological properties of promising local neighborhood with domain knowledge to come up with the best match. In our next approach, AtoM [8, 11], we focus our search space into a summary graph that is constructed by using the query graph as a template. Here, instead of exploring the region for viable matches, we extract a narrow search space composed of feasible solutions and then ask “Is this a good-enough match for the query graph?”. Reversing the processing direction improves efficiency and makes the approach a likely candidate for Map-Reduce based processing.

CHAPTER 4: TOP-*K* GRAPH MATCHING USING TRAM

The first approach we explore to solve the approximate graph matching problem is (Top-*k* Graph Matching) or TraM. Let us re-visit the definition of graph in the following section. We will build off of this definition and formally introduce the notion of similairty.

4.1 Problem Definition

Let us first emphasize that we accept the traditional definition of labeled or attributed graphs where graph G is a pair (V_g, E_g), where V_g is a set of vertices, and E_g is a set of edges such that $E_g \subseteq V_g \times V_g$, defined technically as follows.

Definition 4.1.1 [Labeled Graph] A labeled graph G is a quintuple $G = (V_g, E_g, L, \mu, \tau)$, where

- V_g is the finite set of nodes, and L is a set of labels,
- $E_g \subseteq V_g \times V_g$ is the set of edges,
- $\mu : V_g \rightarrow L$ is the node labeling function, and
- $\tau : E_g \rightarrow L$ is the edge labeling function.

This definition allows handling arbitrary graphs with unconstrained labeling functions. For example, the labels can be given by the set of integers, the vector space $\mathbb{R}^n$ etc. Moreover, unlabeled graphs are obtained as a special case by assigning the same label to all nodes and edges. For this approach, we recognize that the nodes may have an arbitrary number of attributes but the edges are unlabeled. The basic problem we are addressing in this thesis is computing similarity between two labeled graphs.

The heart of our approach depends upon encapsulating the similarity of nodes along two axes – (i) the conceptual likeness of nodes, and (ii) their similarity with respect to the topology they are part of, and then finally combining these two similarities into one overall distance value. The conceptual likeness of two sets of nodes V_1 and V_2 is captured using a domain dependent similarity function, called the σ-similarity, defined as $\sigma : V_q \times V_g \rightarrow [0,1]$ which we assume as given prior to the application of our algorithm. The method used to compute this function does not impact our algorithm. For example, in a protein network, the similarity between proteins may be based on their amino acid sequences, their expression pattern, or their function annotations. In fact, the strength of our approach lies in the fact that multiple domain dependent similarity scores (e.g., scores for individual proteins may come from expression data or sequence data) can be incorporated into the overall similarity calculation of nodes. Multiple attribute values can be converted into a vector, which can subsequently be applied to calculate pairwise similarity σ using standard similarity calculation methods, e.g., cosine similarity.

4.1.1 Random Walk as a Global Graph Property

Similarity of nodes with respect to their topology is slightly more complex, but can be explained intuitively using the notion of random walk score [21, 57], ω_v values (formally characterized in definition 4.1.2), of nodes. The random walk on graphs is defined as the repeated transition of a walker from its current node to a randomly selected neighbor starting at a given set of source nodes S. A derivative of the random walk, known as random walk with restart additionally permits the restart of the walk in every time step at nodes in S with probability β to combat spider traps. Formally, the random walk with restart is defined as

$$p^{(t+1)} = (1-\beta)Wp^{(t)} + \beta p^{(0)} \tag{4.1}$$

where W is the column-normalized adjacency matrix of the graph and $p^{(t)}$ is a vector in which the i^{th} element holds the probability of being at node v_i at time step t. In our application, the initial probability vector $p^{(0)}$ is constructed such that equal probabilities are assigned to all the nodes.

This is equivalent to concurrently beginning the random walk process from each of the nodes with equal probability. The nodes are eventually prioritized according to the values in the steady-state probability vector p^{∞}. Algorithm 1 describes the random walk procedure, which essentially computes the random walk scores of all nodes in G.

Algorithm 1: Random Walk

```
  Input: Graph $G = (V_g, E_g)$ and Restart Probability $\beta$.
  Output: Random Walk Score $P_s(V_g)$.
1 Let $r_s(V_g)$ be the restart vector with all entries having value $\frac{1}{|V_g|}$;
2 Let $A$ be the column normalized adjacency matrix defined by $E$;
3 Initialize $P_s(V_g) = r_s(V_g)$;
4 while $P_s(V_g)$ has not converged do
5     $P_s(V_g) = (1-\beta) * A * P_s(V_g) + \beta * r_s(V_g))$;
```

4.1.2 Structural Similarity of Nodes

Since random walk scores encompass the global topological properties of a node, from the standpoint of graph matching, it can be used to compare topological orientation and relative importance of nodes. These scores thus effectively capture structural cues shared among the nodes. Our goal is to find a function f such that when a node v_1 is structurally more similar to v_2, given a choice between v_2 and v_3, the relationship $\forall v_1 \forall v_2 \forall v_3 (f(v_1, v_2) > f(v_1, v_3) \Rightarrow |\omega_{v_1} - \omega_{v_2}| < |\omega_{v_1} - \omega_{v_3}|)$ should hold, where f is computed using standard functions such as cosine similarity or 1-L2 norm. Fortunately, random walk with restart probability or damping factor, called the β values, for every node v in a graph, computed n times captures the topological likeness of nodes to approximate this property. However, it is important to note that the converse is not always true, i.e., two graphs may satisfy this property yet be structurally distinct as shown in figure 4.1 as a counter example.

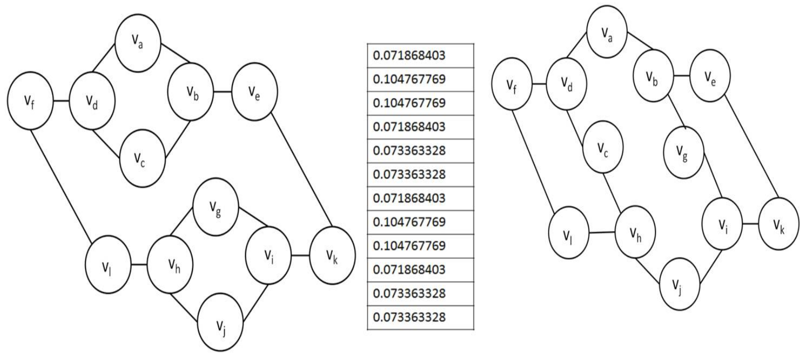


Figure 4.1: Example of two different graphs with identical β-signature.

While an arbitrary choice of n does not impact this property, it can be selected based on the sufficiency conditions discussed in [14,15]. In definition 4.1.2 below, the function ω is computed by extracting the steady state probability vector p^{∞} using equation 4.1.

Definition 4.1.2 [β-signature] Let $G = (V_g, E_g)$ be a graph and $\mathbb{B}$ be a set of β values. Random walk score of a node is a function of the form $\omega : V_g \times \mathbb{B} \rightarrow [0,1]$. An n-dimensional vector $(\omega(v,\beta_1), \omega(v,\beta_2), \ldots, \omega(v,\beta_n))$ is called a β-signature of node $v \in V_g$, denoted $\vec{\beta}(v)$ and the set $\beta(G) = \bigcup_{v \in V} \vec{\beta}(v)$ is called the β-signature of G.

Definition 4.1.3 [Admissible β-signature] For a graph G, a set of β-signatures $\mathbb{A}$, $\mathbb{A} \subseteq \beta(G)$, is called *admissible*, if $\forall v, v'(v, v' \in V_g), \exists \vec{\beta}(v)(\vec{\beta}(v') \in \mathbb{A})$ such that $\vec{\beta}(v) = (\omega(v,\beta_1), \omega(v,\beta_2), \ldots, \omega(v,\beta_n))$, $\vec{\beta}(v') = (\omega(v,\beta_1'), \omega(v,\beta_2'), \ldots, \omega(v,\beta_n'))$ and $\beta_i = \beta_i'$ holds $\beta_i, \beta_i' \in \mathbb{B}$, $1 \leq i \leq n$, i.e., they are defined over identical sequence of β values.

Intuitively, an admissible $\beta(G)$ is a set of n-tuples, one for each node v in G, of random walk scores ω_v. In particular, this definition ensures that the random walk scores in a set for all the nodes are defined over identical β values for a fair comparison. In the remainder of the approach, when we refer to β-signatures of nodes and graphs, we mean admissible β-signatures. Once we have $\beta(G)$, we can now formally define graph similarity as follows.

Definition 4.1.4 [β-similarity] Let $G_1 = (V_{g_1}, E_{g_1})$ and $G_2 = (V_{g_2}, E_{g_2})$ be two graphs and $\beta(G_1)$ and $\beta(G_2)$ be their β-signatures. For any $v_1 \in V_{g_1}$ and $v_2 \in V_{g_2}$, their structural or β-similarity, denoted $\hat{\beta}(v_1, v_2)$, is defined by $\hat{\beta}(v_1, v_2) = 1 - \sqrt{\sum_{i=1}^{k} (a_i - b_i)^2}$, where $\vec{\beta}(v_1) = (a_1, a_2, \ldots, a_k) \in \beta(G_1)$ and $\vec{\beta}(v_2) = (b_1, b_2, \ldots, b_k) \in \beta(G_2)$.

4.1.3 Combined Graph Similarity

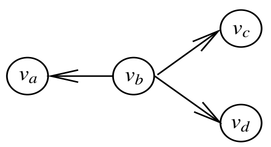
Graph similarity now can be defined using the combination of σ-similarity and β-similarity as follows.

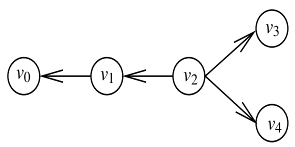
Definition 4.1.5 [Graph Similarity] Let $G_1 = (V_{g_1}, E_{g_1})$ and $G_2 = (V_{g_2}, E_{g_2})$ be two graphs and ϕ be a mapping function. Then, the similarity γ between two graphs G_1 and G_2 under a mapping function ϕ is

$$\gamma(G_1, G_2) = \sum_{\forall v_1, v_2 (v_1 \in V_{g_1}, \phi(v_2) \in V_{g_2})} \sigma(v_1, \phi(v_2)) \times \hat{\beta}(v_1, \phi(v_2)) \tag{4.2}$$

Given a data graph D, a query graph Q, the objective of the top-k inexact graph matching problem is to identify k sub-graphs of D such that the similarity between Q and each subgraph S_i, $1 \leq i \leq k$, is the maximum among all possible subgraphs of D and higher than a threshold λ.

For example, consider the data graph D having five and query graph Q with four nodes in figure 4.2. The β-signatures $\beta(D)$ and $\beta(Q)$ as in definition 4.1.3 are shown in tables 4.2(c) and 4.2(d) respectively for β values 0.8 and 0.6, one for each node as a set of 2-tuples. These signatures were used to calculate pairwise node similarity $\hat{\beta}$ as in definition 4.1.4 shown as the similarity matrix in figure 4.2(e). The overall idea of graph matching is discussed next on intuitive grounds.


(a) Query graph Q


(b) Data graph D

0.440465566	0.366457316
0.290490868	0.285043033
0.089681189	0.11616655
0.089681189	0.11616655
0.089681189	0.11616655

(c) β-signature of $\delta(v_2, 2)$ of D

0.145033072	0.174219384
0.564900785	0.477341849
0.145033072	0.174219384
0.145033072	0.174219384

(d) β-signature of query graph Q

0.647529	0.817134	0.919788	0.919788	0.919788
0.833328	0.664918	0.403107	0.403107	0.403107
0.647529	0.817134	0.919788	0.919788	0.919788
0.647529	0.817134	0.919788	0.919788	0.919788

(e) Similarity matrix

S	**d**	**hop**
v_1	v_0	1
v_2	v_0	2
v_2	v_1	1
v_2	v_3	1
v_2	v_4	1

(f) 2-hop reachability of D

Figure 4.2: β-signature, similarity generation for query and localized data graph.

4.2 Overview of TraM

Intuitively, we follow three distinct steps to compute k similar graphs of a query graph Q from a data graph D. We expect the size of the data graph (number of nodes M) to be very large compared to the number of nodes m in the query graph, i.e., $m \ll M$. In practical applications where protein networks are searched for specific known interaction patterns, or pathways are matched with candidate networks, the query graphs usually contain a relatively small number of nodes, typically in the range of 20-200. Since a subgraph of D that is similar to Q is expected to have a comparable number of nodes to be useful, it is prudent to compare graphs of similar size, and network topology on a need to compare basis. By doing so we reduce the cost substantially and explore only potentially similar graphs. Thus, we introduce the concept of neighborhood biased β-signature for the data graphs to limit the size of candidates as follows.

Definition 4.2.1 [δ-neighborhood of nodes] Let Q be a query graph, and r be its radius[*]. Let $D = (V_d, E_d)$ be any graph, and $v \xrightarrow{j} u$ represent j-hop reachability from node v to u. Then δ-neighborhood of v is the set $\{u | v, u \in V_d \wedge v \xrightarrow{j} u \wedge j \leq r\}$ (including zero hop reachability, i.e., v itself), denoted $\delta(r, v)$.

[*]We use the standard definition of radius of a graph in graph theory which is the minimum eccentricity of any vertex, where eccentricity represents the hop or edge distance between a node and the farthest node in the graph.

Definition 4.2.2 [Induced subgraphs] Let $G = (V_g, E_g)$ be any graph, and $N \subseteq V_g$ be an arbitrary set of nodes. Then, $G' = (N, E')$ is called an induced subgraph, denoted $\chi(N) = G' = (N, E')$, such that whenever $v, v' \in N$ and $e = (v, v') \in E_g$, e is also in E', and nothing else is in E', i.e., G' is a subgraph of G, denoted $G' \sqsubseteq G$.

First, we compute the β-signature of the query graph $Q = (V_q, E_q)$, $\beta(Q)$, for a predetermined set of β values $n = |\mathbb{B}|$. Note that we thus have a set of n-dimensional β-signature vectors, one for each node $v \in V_q$. In the second step, we compute the neighborhood N_u of each node $u \in V_d$ as a selection mechanism for generating D's candidate subgraphs. We proceed to generate the induced subgraph G_u from N_u only if the size of Q and N_u are comparable. The β-similarity $\beta(G_u)$ is then calculated. Using a dynamic programming method, we then match each $v_q \in V_q$ with $u_d \in V_d$ such that the matching is maximized for all nodes, i.e., best possible similarity match. In the final step, we compute the graph similarity $\gamma(Q, G_u)$ of Q and G_u and insert the graph in a priority queue according to its similarity score, and move on to the next node in the data graph and repeat the process. At the end, we select the top k graphs from the queue completing the process.

The matching process can be explained using the examples in figure 4.2 and 4.3 intuitively as follows. Figure 4.2 shows β-signatures of the query graph as well as the 2 hop reachability of data graph nodes as 2-neighborhood set $\bigcup_{\forall u(u \in V_d)} N_u$. It is interesting to note that although the radius r of the query graph is 1, choosing to extract 2-neighborhood from the data graph does not break the algorithm. Although we now extract more nodes from the data graph and increase the number of matching contenders, the algorithm is still well defined and robust. The β-signatures $\beta(Q)$ and $\beta(D)$ for both graphs calculated respectively with damping values 0.8 and 0.6 are shown just below the corresponding graphs. The pair-wise β-similarity of nodes from the query graph and the graph G_2 induced from N_2 is shown. The overall matching process is shown in figure 4.3.

Note that the 2 hop neighbors of node $v_2 \in V_d$ includes all the nodes in V_d, i.e., $\{v_1, v_2, v_3, v_4\}$. So, our goal is to match Q using the β-similarity values we have calculated above with the induced subgraph G_2, which incidentally happens to be the whole data graph in this example. The idea is to find a sequence of matches such that each node in Q is matched with at most one node in G_2, and the overall γ-similarity is the highest. The process of matching using a dynamic programming method is shown in figure 4.3 where, the brown dotted lines show the match that yielded the highest overall match, i.e., $v_a \rightarrow v_1$, $v_b \rightarrow v_2$, $v_d \rightarrow v_3$, and $v_c \rightarrow v_4$. We defer a more detailed and technical discussion on the matching process until later sections.

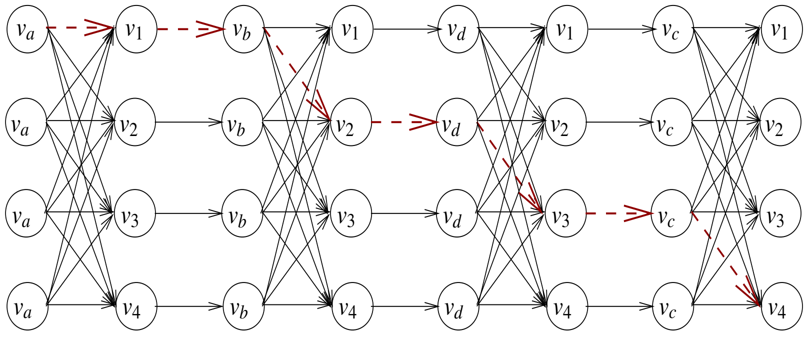


Figure 4.3: Extracting matching sub-graphs from the localized data graph.

4.3 Top-k Graph Matching using Neighborhood Signatures

Technically, the top-k graph matching between a query graph Q and data graph D requires a set $\mathbb{B}$ of n β values, and a set of user supplied thresholds – a conceptual similarity threshold μ_v of nodes, a structural similarity threshold μ_s, and an overall graph similarity threshold λ. We pre-calculate the node reachability information of the data graph D, and store and index it in the database. Since the query graph size is limited to at most m nodes, we retain node reachability information for each individual node in the data graph up to r hops, where r represents the radius of the query graph Q. Biological networks tend to exhibit small-world properties, where most nodes can be reached from any other node by a small number of hops. To be specific, a small-world network is defined to be a network where the typical distance between two randomly chosen nodes grows proportionally to the logarithm of the number of nodes in the network. Thus, since the radius of the query graph would be proportional to the logarithm of the number of nodes in the query graph, the value of m for the reachability calculation is relatively small. Hence, we can accommodate inclusion of an arbitrarily large query graph as well. It is expected that the data graph size can impose significant memory resident processing, and thus the whole reachability calculation process is better done in the database [4].

4.3.1 TraM Matching Algorithm

The algorithm has three main components – (i) generating candidate subgraphs from data graph *D*, (ii) selecting and pruning candidate subgraphs, and (iii) matching candidates with query graph. Unless all matching are of interest regardless of their degree of likeness (conceptual or structural), exploring all possible subgraph is wasteful because the number of potential matches are extremely high. So, it is desirable to state what is an acceptable match, both in terms of conceptual (σ) and structural similarity (β), and then in terms overall graph similarity (λ). We use these thresholds to admit only those candidates that are likely to be of interest, and then rank the top *k* candidates as the answer. Algorithm 2 outlines this process.

```
Algorithm 2: GraphMatch
  Input: Data graph $D = (V_d, E_d)$ and query graph $Q = (V_q, E_q)$. Thresholds $k$, $\mu_v$, $\mu_s$ and $\lambda$.
  Output: Top-$k$ matches of $Q$.
1 Initialize priority queue $PQ$ as empty;
2 Calculate $\beta$-signature $\beta(Q)$ for $Q$;
3 Compute radius $r$ of $Q$;
4 for $\forall v_d (v_d \in V_d)$ do
5     Compute $\delta(r, v_d)$;
6     if $|Filter(\delta(r, v_d), Q, \mu_v, \mu_s)| > |V_q|$ then
7         Top-$k$ Match($Q$, $\beta(Q)$, $D$, $\beta(\chi(\delta(r, v_d)))$, $\lambda$, $PQ$);
8 return All top $k$ graphs $g \in PQ$ ;
```

Candidate Subgraph Generation

We represent the data graph as a set of tuples, each containing source, destination and number of hops information. We then create indices over all these attributes for fast extraction of neighborhood information from the data graph. Once the radius r of the query graph is computed, the δ-neighborhood for each node $v \in V_d$ is computed as needed using this index as follows.

For each node v in the data graph D, we compute the δ-neighborhood $\delta(r,v)$ of v as defined in definition 4.2.1. From $\delta(r,v)$, we compute the candidate subgraph C as the induced graph $\chi(\delta(r,v))$ as defined in definition 4.2.2. This candidate graph is then pruned to eliminate nodes and edges not likely to match with the query graph before processing. We embed this pruned graph in the vector space domain and calculate pair wise node similarity[†] between $v_g \in \chi(\delta(r,v))$ and $v_q \in Q$ as $\hat{\beta}(v_q,v_g)$.

Pruning Search Space

The induced subgraph $\chi(\delta(r,v))$ may be quite large depending on the density of the data graph demanding too many comparisons with nodes that will not be part of a possible candidate. In order to prune the search space in the data graph, we consider two specific criteria. The first is that if the size of the δ-neighborhood is less than the number of nodes in the query graph, we discard this subgraph because it is not considered a candidate "subgraph" of D[‡]. The other criteria used to prune a candidate subgraph is to remove nodes in $\delta(r,v_d)$ that will not contribute to the matching as they are significantly below the matching thresholds μ_v or μ_s. To determine admissibility, we check to see if there exists at least one node in the data graph for every node in $\delta(r,v_d)$ for which the σ and β similarities are above the thresholds μ_v or μ_s respectively. The function *Filter()* in algorithm 4 captures this idea. Once $\delta(r,v_d)$ is pruned, the induced subgraph $\chi(\delta(r,v_d))$ serves as a likely candidate for matching in the next step.

[†]As discussed in section 5.1, the vertices of the graphs may have several attributes. To compute conceptual similarity of nodes as σ-similarity, we assume that we are given a domain and application dependent computable function σ, that can compute the similarity between all pairs of vertices in the data graph and the query graph respectively which is left as an orthogonal research.

[‡]However, should we like to consider such graphs as candidates, this condition may be eliminated or adjusted to admit certain size of subgraphs of D smaller than Q.

Matching Candidate Subgraphs

Since the computation of maximum graph similarity is an optimization problem, the overall matching process is implemented using a dynamic programming algorithm to avoid redundant computation. In this formulation, we maximize the function *sim* defined in terms of pairwise σ and β-similarity of query and data graph nodes. Our goal is to find a sequence of matches between nodes in query graph Q and candidate data graph $\chi(\delta(r,v)), v \in V_d$ such that the matching score is maximum. We then accept the matching score at the final step as the graph similarity $\gamma(\chi(\delta(r,v)),Q)$ and include it as one of the possible solutions. The final decision is made once all the candidates are matched for maximum matching scores, and we then select only the top k solutions based on these matching scores. In the expressions below, i and j represent nodes in the graphs, $sim(i,S^{(i,t)})^{(t)}$ denotes the max score at time stamp t ending at node i for the set of nodes $S^{(i,t)}$. Algorithm 3 essentially implements equation 4.3, which is used by algorithm 2 to match graphs.

$$\begin{aligned} sim(i,S^{(i,t)})^{(t)} &= max\{sim(j,S^{(j,t-1)})^{(t-1)} + \sigma(j,i) \times \hat{\beta}(j,i)\} \text{ when } t \text{ even} \\ sim(i,S^{(i,t)})^{(t)} &= sim(i,S^{(i,t-1)})^{(t-1)} \text{ when } t \text{ odd} \end{aligned} \tag{4.3}$$

At each odd time stamp, we have all the nodes representing a single node from the query graph; at each even time stamp, we have all the nodes from $\delta(r,v_d)$. Figure 4.3 depicts the process. Nodes in the odd time stamps have the same labels. These nodes in the same column represent one single node from the query graph. It is noticeable from the figure that at time stamp 1 we have node v_a, time stamp 3 node v_b, time stamp 5 node v_d and so on. At each odd time stamp, the assignment of a specific node label is done by calculating the breadth first search (BFS) node order starting from an arbitrary node of the query graph. This order guarantees that the next node to be matched is at most, one hop away from at least one node of the already visited node set. $S^{(j,t-1)}$ represents all the nodes that have already been visited along this path. For all nodes j in level $t-1$, we calculate this value for all the nodes i in level t such that $i \notin S^{(j,t-1)}$ and distance between i and at least one node in $S^{(j,t-1)}$ is 1 (if t is even). The notion of best match is identified by the maximum value that function γ can assume at any given time-stamp which in turn depends on σ and $\hat{\beta}$ values. Each column in the figure represents either a single node or a set of nodes. In the former case, the single node comes from the set of nodes in the query graph and in the latter case are all the nodes from the extracted neighborhood of the data graph. Each column represents a time-stamp (t) i.e. the first column represents time-stamp 1 and so on.

4.3.2 Caching Local Neighborhood

Since the data graph is stored in the database, accessing each node and generating its δ-neighborhood requires multiple database accesses. Moreover, while calculating the Top-k Match, we query the database to assert whether the node in question has a path of length one from any node from the previous time-stamp. This is a repetitive task that takes place for all pairs of nodes from the query graph and the δ-neighborhood of the data graph $2 \times |Q|$ times, where Q is the query graph. Hence, at each iteration of the algorithm, caching the relevant δ-neighborhood would expedite the process by $2 \times |Q|$ times.

However, we can minimize the number of cache invalidate operations by imposing an order on the nodes of the data graph and executing the GraphMatch algorithm according to that order. If the next candidate node in the iteration (Line 2 GraphMatch algorithm) has the highest number of shared neighbors with current node, the extent of cache invalidate operations would be minimal. The structure of the cache is a two dimensional matrix containing information regarding whether there is an edge between node i and node j, where $i, j \in \delta(r, c)$, where c is the current node. The first node to be expanded is one with the highest clustering coefficient in the network. The order is pre-computed and stored for convenience. Algorithm 5 delineates the whole process where the function $Shared\_Neighbor(i, j)$ returns the number of shared neighbors between node i and j.

4.3.3 TraM Properties

Several interesting properties of TraM can be stated formally to highlight its strengths in a definitive manner. While an orthogonal research to determine how well TraM identifies biologically significant networks is interesting, these properties and run time performance help TraM stand out in sharp contrast to major contemporary matching tools.

Theorem 4.3.1 *The algorithm Top-k Match() returns connected sub-graph in response to a connected query graph.*

Proof: Let us assume that the algorithm extracts disconnected sub-graph from the data graph as a response to a connected query graph. Thus as shown in equation 4.3, at some time stamp t, the algorithm chooses a new node i that is at least two hops away from all the nodes in $S^{(j,t-1)}$. However, i is chosen such that $i \notin S^{(j,t-1)}$ and $\exists k(k \in S^{(j,t-1)} \wedge distance(i,k) = 1)$, where the function $distance(i,k)$ returns the number of hops needed to reach from i to k. Thus at time stamp t, the chosen i cannot be at least two hops away from all the members of $S^{(j,t-1)}$. Thus the algorithm cannot return disconnected subgraph in response to a connected query graph. $\square$

Theorem 4.3.2 *Top-k Match() algorithm returns the maximum scoring connected components in the final time-stamp.*

Proof: The fact that Top-k Match() returns connected subgraph in response to a connected query graph follows from the previous theorem. Let us assume that the algorithm Top-k Match() extracts set of nodes with sub-optimal matching scores. We can assume that the set of nodes that Top-k Match() returns at the final time stamp is $\{n_1, n_2, \ldots, n_i, n_{i+1}, \ldots, n_t\}$. We can safely assume the n_i is the first node that was a suboptimal choice. And $n_{i'}$ is the optimal choice at time stamp i. It is evident that while calculating $sim(n^{i+1}, S^{(n_{i+1},i+1)})^{(i+1)}$, we chose $sim(n_i, S^{(n_i,i)})^{(i)} + \sigma(n_i, n_{i+1}) \times \hat{\beta}(n_i, n_{i+1})$ if $i+1$ is even, $sim(n_i, S^{(n_i,i)})^{(i)}$ otherwise. However, from equation 4.3 at every even time step, we choose

$$max\{sim(n_i, S^{(n_{i+1},i)})^{(i)} + \sigma(i+1,i) \times \hat{\beta}(i+1,i)\}\text{when } t \text{ is even}$$

Hence, if $n_{i'}$ maximized the sim score, the equation would have chosen $n_{i'}$ instead of n_i. Since $sim(n_i, S^{(n_i,i)})^{(i)} + \sigma(n_i, n_{i+1}) \times \hat{\beta}(n_i, n_{i+1}) < sim(n_{i'}, S^{(n_{i'},i')})^{(i')} + \sigma(n_{i'}, n_{i+1}) \times \hat{\beta}(n_{i'}, n_{i+1})$ if t is even and since the odd time stamp is used only to select a new query node to match, it does not affect the quality of the match. Thus, the process could not have chosen n_i, since it would have resulted in a suboptimal score and would not have maximized the *sim* score at time stamp *i*. Thus it is evident that top-*k* Match() procedure returns the maximum scoring connected components in the final time-stamp. □

Theorem 4.3.3 *Identical random walk scores for multiple subgraphs do not imply subgraph isomorphism.*

Proof: Let $G = (V_g, E_g)$ be an undirected graph where $V_g = \{n_1, n_2, \ldots, n_k\}$ are the vertices of the graph. Let $S = \{n_i, n_j, n_m, n_p\}$ be the set of vertices that induces the subgraph of interest. Moreover, $\{(n_i, n_j), (n_j, n_m), (n_m, n_p), (n_p, n_i)\} \in E$. Let us assume that the random walk scores of the members of S are $\{r_1, r_2, r_1, r_2\}$ respectively. Let n_x and n_y be two nodes such that $n_x \notin S$ and $n_y \notin S$ and $(n_x, n_y) \in E$, and none of the nodes in S has edges with n_x or n_y. Further we assume that the random walk score of n_x and n_y is r_1 and r_2 respectively. Now we can reconstruct G by redistributing the edges in a way such that the random walk scores of the members of S do not change. This can be done by removing edge (n_m, n_p) and (n_x, n_y) and adding edges (n_m, n_y) and (n_p, n_x). Since the structure of the graph from S has changed yet the random walk scores of its members have not, it is evident that the same set of random walk scores for multiple subgraphs does not necessarily infer subgraph isomorphism. □

In figure 4.5, we see an example matching scenario where one of the nodes from the query graph does not appear in the data graph. Here the similarity value among the nodes is a function of the node labels i.e. $\forall i, j(\sigma(i, j) = 1)$, if label of i and label of j is same; 0 otherwise. Since we take into account both topological and entity similarity, when we compute the approximate matching subgraphs from the data graph, the entire neighborhood of the candidate subgraph is explored for a potential match. In the example, although node y is missing from the data graph, a similar node x, with $\sigma(x, y) = 0$ and $\beta(x, y) \approx 1$ is present. Thus, the approach systematically explores the neighborhood of the data graph to come up with the best pairwise match for the query graph nodes. Figure 4.5(e) depicts the matching subgraphs, where the first subgraph has

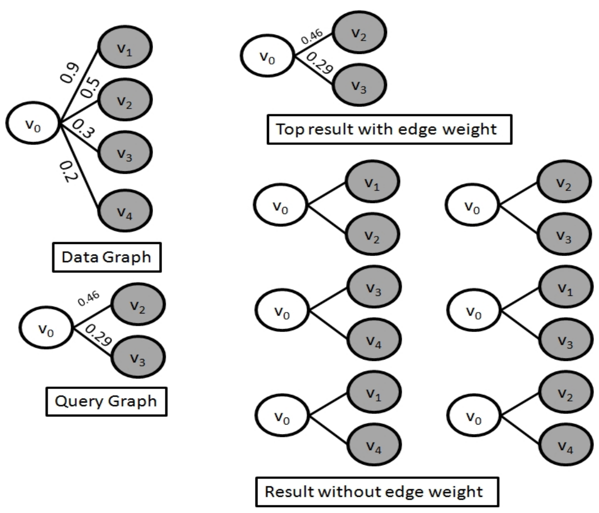


Figure 4.4: Example of matching with weighted edge.

three matching nodes and the highest score and the subsequent matches have a lower score as they have two matching nodes augmented by topological matches.

4.4 Edge-labeled Graphs

The applicability of the method can be extended for labeled edges as well. Protein-protein interaction networks, for example, can have numeric values associated with edges delineating the confidence score of the interactions. To account for such cases, we extended the first dynamic equation of our method as follows:

$$sim(i,S^{(i,t)})^{(t)} = max\{sim(j,S^{(j,t-1)})^{(t-1)} + \sigma(j,i) \times \hat{\beta}(j,i) + \theta_c \times \frac{1}{((E_{q_{sel}}) - (E_{d_{sel}}))^2 + \varepsilon}\}$$

The terms $E_{q_{sel}}$ and $E_{d_{sel}}$ refer to the attribute values associated with selected edges from the query and the data graph respectively. Since the set $S^{(j,t-1)}$ contains all the mapping between nodes in the data and the query graph and since, the expansion in the data graph guarantees connected subgraph according to theorem 1, we can always compute the equation above. The term θ_c is a weight factor that assigns importance to the overall edge attribute portion of the equation. The ε parameter is a very small value that ensures that the value of the denominator does not assume zero.

Figure 4.4, for example, shows a scenario where matching with weighted edge may alter the ranking of the obtained result. On the left hand side of the image, a data and a query graph with edge weight are delineated. For the sake of simplicity, we consider that similarity among any pair of nodes is equal. It can be seen that there are theoretically six matches with the same matching score if the edge weight is not considered in the process. If the edge-weight is taken into consideration, as shown in the top-right part of the image, the top ranked match from the data-graph contains nodes v_0, v_2 and v_3. To show how incorporating edge-weight into the calculation can significantly alter the outcome, we chose a subset of 400 nodes and 3326 edges from PPI network data set [113] containing confidence scores for the edges.

We chose a query graph with five distinct nodes as shown in figure 4.6. We run the query using both methods and observe that since there exists one isomorphic subgraph in the interaction network, it appears as the top result for both the original and the extended method. For the other approximate matches, however, the ranking changes due to inclusion of edge weight. The result is shown in Table 4.1.

Algorithm 3: Top-*k* Match

```
Input: Query graph $Q = (V_q, E_q)$, $\beta(Q)$, candidate graph $C = (V_c, E_c)$, $\beta(C)$, threshold $\lambda$.
       Queue $PQ$
Output: Updated priority queue $PQ$.
1  for $i = 1$ to Number of Nodes in time-stamp 1 do
2      initialize $S^{(i,1)} = \emptyset$ and $sim(i, S^{(i,1)})^{(1)} = 0$;
3  for $t = 2$ to $2 \times |V_q|$ do
4      for $r = 1$ to Number of Nodes in time-stamp $t$ do
5          if $t$ is Even then
6              for $p = 1$ to Number of Nodes in time-stamp $t-1$ do
7                  if $(r \notin S^{(p,t-1)})$ and $((\exists i(i \in S^{(p,t-1)}$ and $r \xrightarrow{1} i$ and $i \in V_c)))$ and
                   $(sim(p, S^{(p,t-1)})^{(t-1)} + \sigma(p,r) \times \hat{\beta}(p,r)) \geq MAX)$ then
8                      $MAX = sim(p, S^{(p,t-1)})^{(t-1)} + \sigma(p,r) \times \hat{\beta}(p,r)$;
9                      $k = p$;
10                     $newMAX = true$;
11             if $newMAX = true$ then
12                 $S^{(r,t)} = S^{(k,t-1)} \cup \{r\}$;
13                 $sim(r, S^{(r,t)})^{(t)} = MAX$;
14         if $t$ is Odd or $newMAX = false$ then
15             $S^{(r,t)} = S^{(r,t-1)}$;
16             $sim(r, S^{(r,t)})^{(t)} = sim(r, S^{(r,t-1)})^{(t-1)}$;
17         if $t = 2 \times |V_q|$ and last $r$ then
18             $\gamma(Q,C) = sim(r, S^{(r,t)})^{(t)}$;
19 if $\gamma(Q,C) \geq \lambda$ then
20     Add $\langle C, \gamma(Q,C) \rangle$ to queue $PQ$;
21 return $PQ$
```

Algorithm 4: Filter

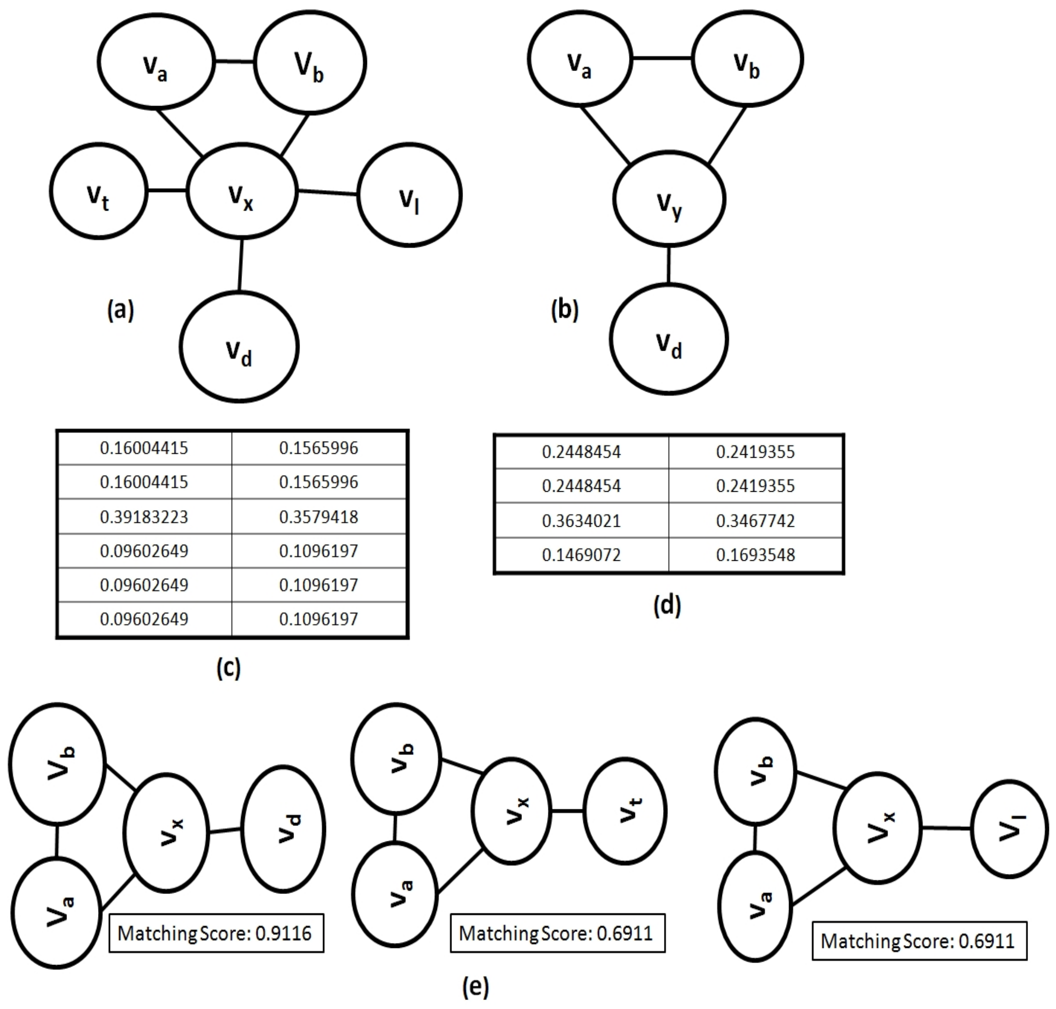
```
Input: Set of nodes $\delta(r, v_d)$ and Graph $Q = (V_q, E_q)$. Thresholds $\mu_v$ and $\mu_s$.
Output: Pruned $\delta(r, v_d)$.
1 if $|\delta(r, v_d)| > |V_q|$ then
2     for every $v_n \in \delta(r, v_d)$ do
3         Compute candidate subgraph as $\chi(\delta(r, v_d))$;
4         Compute $\beta$-signature $\beta(\chi(\delta(r, v_d)))$ for $\chi(\delta(r, v_d))$;
5         if $\forall v_q(v_q \in V_q(\max\{\sigma(v_n, v_q)\} < \mu_v$ or
6         $\max\{\hat{\beta}(v_n, v_q)\} < \mu_s))$ then
7             remove $v_N$ from $\delta(r, v_d)$;
8 return $\delta(r, v_d)$;
```

Algorithm 5: Node Order

```
1 $i = argmax_{\forall j(j \in D)}\{ClusteringCoefficient(j)\}$;
2 $visited(i) = true$;
3 $counter = 1$;
4 while $counter <= |D|$ do
5     Generate $\delta(d, i)$;
6     $AssignOrder(i) = counter$;
7     $i = argmax_{\forall j \in D}\{Shared\_Neighbor(i, j)\}$ such that $visited(i) = false$;
8     Increment $counter$;
```

Table 4.1: Inclusion of edge weight in PPI network

Rank(Extended Method)	Rank(Original Method)	Matching Nodes
1	1	{ENSG00000119203, ENSG00000181222, ENSG00000186298, ENSG00000172531, ENSG00000161654}
2	9	{ENSG00000184900, ENSG00000100167, ENSG00000149617, ENSG00000168385}
3	5	{ENSG00000184900, ENSG00000100167, ENSG00000149617, ENSG00000070669}
4	8	{ENSG00000184900, ENSG00000100167, ENSG00000149617, ENSG00000160271}
5	10	{ENSG00000184900, ENSG00000100167, ENSG00000149617, ENSG00000160271}
6	11	{ENSG00000184900, ENSG00000100167, ENSG00000149617, ENSG00000183878}
7	7	{ENSG00000184900, ENSG00000100167, ENSG00000149617, ENSG00000143466}
8	2	{ENSG00000184900, ENSG00000100167, ENSG00000149617, ENSG00000147439}
9	4	{ENSG00000184900, ENSG00000100167, ENSG00000149617}
10	6	{ENSG00000184900, ENSG00000100167, ENSG00000149617, ENSG00000164022}
11	3	{ENSG00000184900, ENSG00000100167, ENSG00000149617, ENSG00000149617}



0.16004415	0.1565996
0.16004415	0.1565996
0.39183223	0.3579418
0.09602649	0.1096197
0.09602649	0.1096197
0.09602649	0.1096197

0.2448454	0.2419355
0.2448454	0.2419355
0.3634021	0.3467742
0.1469072	0.1693548

Figure 4.5: Sub-graph matching with missing nodes: (a) Data graph, (b) Query graph, (c) β-signature of data graph, (d) β-signature of query graph, and (e) Matching.

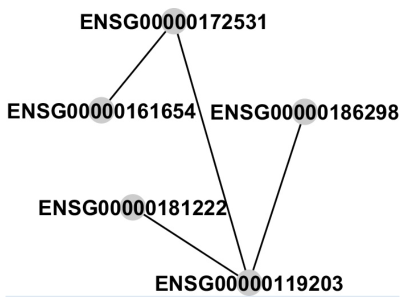


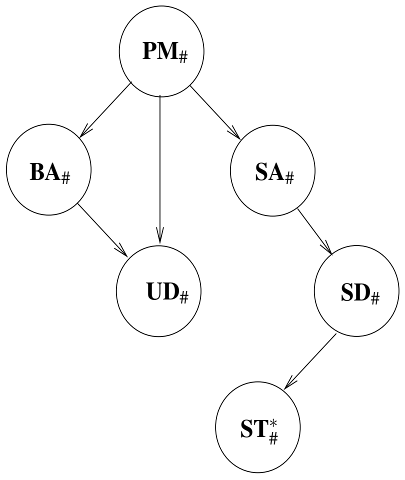
Figure 4.6: Protein-protein interaction query graph.

CHAPTER 5: TOP-*K* GRAPH MATCHING USING ATOM

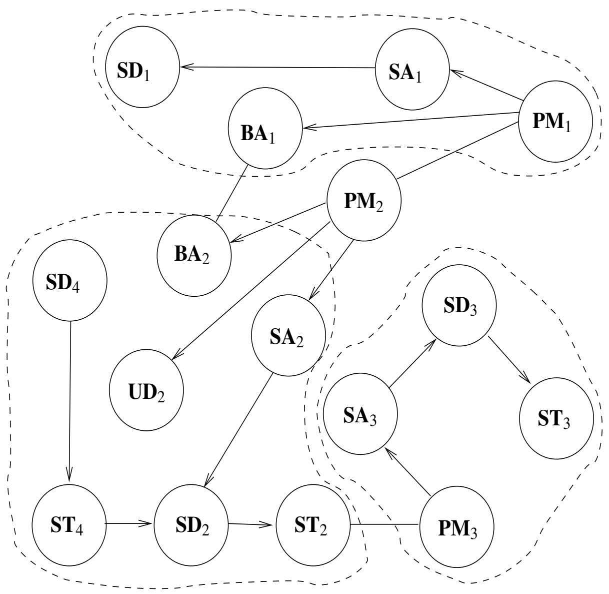
We adapt the network query example from [68] to illustrate the basic idea behind AtoM on intuitive grounds. In this example, consider an application in which an administrator is trying to identify a team of people related to each other in a social network such as LinkedIn or Facebook each of which possesses some skills set. The relationship among them could be that they know each other as friends, or they had already worked in a team, and the objective is for the administrator to find the best possible team for a cluster hire to shore up an ambitious software development project*. Her requirement for the best possible team she has in mind includes a Project Manager (PM), a Business Analyst (BA), a Software Architect (SA), a User Interface Designer (UD), a Software Developer (SD), and a number of Software Testers (ST). The organizational relationship among the team members she has in mind is shown as the query *Q* in figure 5.1(a) where a direct arrow (serves as an edge annotation) from a node *A* to *B* functionally means *B* reports to *A*, i.e., the Business Analyst reports directly to the Project Manager, and the requirement of the set of Software Testers is shown as the node ST*. The required skills for each of these positions are shown in the table in figure 5.1(c), whereas the skills of each member in the social network is shown in the table in figure 5.1(d).

To find structures similar to *Q* in *D*, we will have to consider the similarity of the nodes in terms of their skills set, and their structural relationships. Figure 5.2(a) shows the node and possible structural similarity by grouping node types together according to similarity of skills where darker shades of data graph nodes depict stronger similarity with the corresponding query

*An equally interesting social query may involve an Editor's attempt to decide whether or not to send a paper for review to an expert in an area of research. To find such an expert, he might want to find an expert who has the needed expertise and does not have a conflict of interest. A conflict of interest is assumed to exist if a member of a social group involving co-authors, advisors, and colleagues has authored the paper under consideration and the expert also belongs to the same social group, and thus cannot review the paper. The structure of the social group may be expressed a graph query with a pattern similar to the query in figure 5.1(a).



(a) Query pattern Q.



(b) Data graph D.

Position	Skills
PM$_\#$	Project development, business strategy, mobile
BA$_\#$	Project management, business analysis, requirement gathering, sharepoint
SA$_\#$	Data modeling, ETL, software design
SD$_\#$	MapReduce, Pig, Java
ST$_\#$	Test automation

(c) Required skills set for Q.

Position	Skills
PM$_1$	Project development
PM$_2$, PM$_3$	Project development, business strategy, mobile
BA$_1$	Business analysis
BA$_2$	Project management, business analysis, requirement gathering, sharepoint
SA$_1$, SA$_3$	Software design
SA$_2$	Data modeling, ETL, software design
SD$_1$, SD$_2$, SD$_3$, SD$_4$	MapReduce, Pig, Java
ST$_2$, ST$_3$, ST$_4$	Test automation

(d) Available skills set in D.

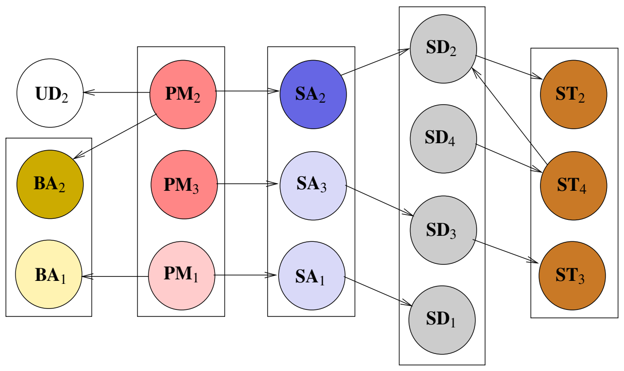
Figure 5.1: Querying a social network database.

node. For example, the lighter colored node in PM category, i.e., PM$_1$, has a low similarity with the PM$_\#$ node in the query graph (due to the low degree of overlapping skills set). Our goal is to filter out such low quality matches and generate the candidate structure in figure 5.2(b) as a possible match. Note that even though the node SA_3 has low similarity with $SA_\#$, the overall structure is similar enough to be retained as a possible match. From figure 5.2(b), we extract the three matches shown in figure 5.2(c) as our query response in which we account for the query constraint $ST^*_\#$ by selecting two alternatives.

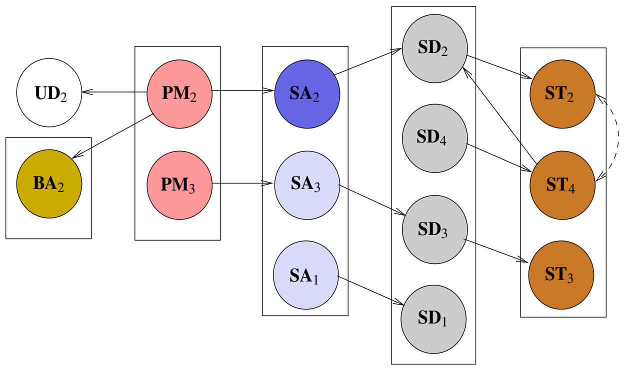
Intuitively, we initially group data graph nodes together using one of the nodes from the query graph as the grouping guide. Thus, for each node in the query graph[†], multiple nodes in the data graph are grouped together. For example, in order to group together all the nodes that are similar to $PM_\#$, PM_2 and PM_3 are grouped together. Hence, once the grouping process is completed, we are guaranteed to have a set of nodes with high similarity values with those of the query graph. Subsequently, the topology of this reduced search space is explored in form of a "look-up operation" to identify similar sub-graphs as discussed above in example 5.1, and help us significantly reduce computational overhead in AtoM.

To appreciate this advantage, let us compare AtoM's matching process with TraM [10] that has been shown to outperform SAGA [92] and C-Tree [45]. To match *Q* with *D*, TraM would have systematically explored the neighborhood of each node to apply filtration criteria and assess their potential of being a valid match. This could, however, generate highly overlapping neighborhoods multiple times. A neighborhood once selected undergoes a matching process in TraM to identify top-*k* matches. AtoM, on the other hand, saves time by significantly pruning search space followed by efficient validation of the the sub-graphs in the reduced search space to identify top-*k* matches. The summarization step by grouping similar nodes in AtoM as shown in figure 5.2(a) is critical to this drastically reduced search time. To illustrate this, let us assume that we had one more node in the data graph named AR (stands for artist with painting expertise). This node would not have been picked by AtoM and placed in any of the groups since AR is not similar to any of the query graph nodes. Therefore, the topology that AR assumes with its neighbors would not have been explored by AtoM, but TraM and other contemporary algorithms such as GADDI [115], TALE [93], VFLib [29] and C-Tree [45] would.

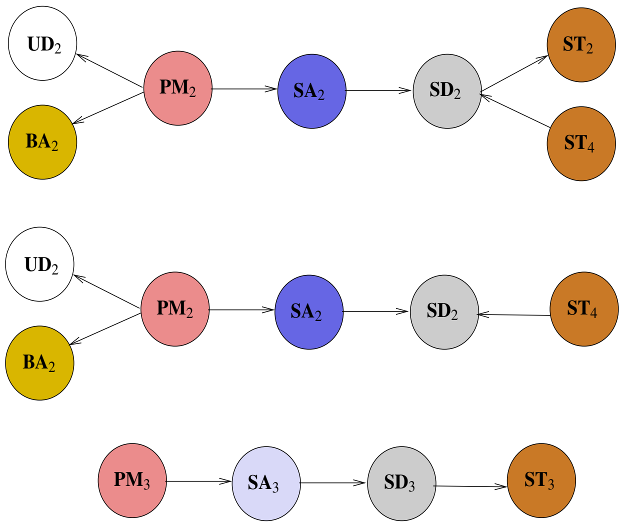
[†]In general, multiple nodes from the query graph can be used to group nodes from the data graph, as we will discuss shortly when we formally explain the matching process using example 5.3.



(a) Similarity grouping.



(b) Filtering low similarity nodes.



(c) Extraction of results.

Figure 5.2: Steps involved in the matching process.

5.1 Problem Definition and the Semantics of Matching

Let us revisit the definition of graph to reiterate the importance of attributed graphs. A graph G is a mathematical pair (V,E), where V is a set of vertices, and E is a set of edges such that $E \subseteq V \times V$. The nodes and edges of graphs can have labels and other attributes to describe them further. Therefore, a graph is formally defined as follows:

Definition 5.1.1 A labeled graph G is a quintuple $G = (V,E,L,\mu,\tau)$, where

- V is a finite set of nodes, and L is a set of labels,
- $E \subseteq V \times V$ is a set of edges,
- $\mu : V \to L$ is a node labeling function, and
- $\tau : E \to L$ is an edge labeling function.

This definition allows us to handle arbitrary graphs with unconstrained labeling functions. For example, the labels can be given by the set of integers, the vector space $\mathbb{R}^n$ etc. Moreover, unlabeled graphs are obtained as a special case by assigning the same label to all nodes and edges. For this dissertation, we consider that the nodes may have an arbitrary number of attributes but the edges are unlabeled. In the discussion to follow, when we refer to a graph $G = (V,E)$, we actually mean it as a shorthand for a labeled graph $G = (V,E,L,\mu,\tau)$ unless otherwise stated where the remaining components are assumed.

The ability to cast overall similarity in terms of the nodes and edges of graphs has generally been the main vehicle for graph matching. However, the way these similarities are computed and composed to construct the overall similarity contributes largely to the high computational costs of graph matching. A notable distinction was the approach inspired by random walk with restart [21] where node similarity is made dependant on the structural likeness of the neighborhoods they are part of [10, 41, 63] and thus embed the edge similarity directly into node similarity in a quantifiable manner. Although it was shown that such approaches improved precision, recall, efficiency and overall scalability of graph matching compared to leading algorithms such as C-Tree, and SAGA, the recent random walk based algorithm TraM was still spending too much time to compute similarity scores. Depending on the user supplied similarity threshold, TraM is likely destined to spend a significant amount of time without impacting the precision in an appreciable way because to ensure structural similarity component of a node is high, it will need to explore an increasingly larger space of the networks to compute the random walk scores.

In this dissertation, we explore an alternative to random walk based similarity score computation for graph matching without sacrificing efficiency, and precision and recall. In this new approach, we use a simple pairwise node similarity computation along the lines of traditional approaches in C-Tree, SAGA and TALE, but boost performance by a novel method in which we avoid computing a large number of candidate graphs and rejecting the ones not meeting the expected threshold. Generally, as the number of top matched graphs k and the similarity threshold ε are made smaller to respectively select only a handful of quality matches from a possible exhaustive set of graph pairs, the degree of wasted computation increases[‡]. In our current approach, we are able to allow arbitrarily lower threshold yet select the most promising matches using a low k at a cost lower than TraM, and thus by extension at a cost lower than most leading algorithms we are able to compare with.

The conceptual likeness of two graphs in AtoM is captured as a function of the similarity of two sets of nodes V_1 and V_2 using a domain dependent similarity function over the set of associated labels L_1 and L_2 respectively, called the σ-similarity and defined as $\sigma : (V_1 \to L_1 \times V_2 \to L_2) \to [0,1]$, and an expected structural dissimilarity threshold λ, both of which we assume as given prior to the application of our algorithm. The k selected graphs will also have a σ-similarity above a threshold $(1-\varepsilon)$ among the nodes with the query graph. The intent is to control the match degradation through the two thresholds ε and λ, and admit only a select few that meet the product of these two thresholds through the parameter k. Higher ε admits more dissimilar nodes, while higher λ will admit graphs that are potentially less structurally dissimilar. But lower dissimilarity thresholds will entail higher computational costs yielding more accurate

[‡]Higher threshold with lower k potentially miss valid matches of interest. So, there is a need to allow lower similarity thresholds but only select the best possible matches.

match. Therefore, user has control over the cost and the graphs admitted by making judicious choices of these parameters based on application needs.

5.1.1 Graph Similarity

Similarity between two graph can be defined in terms of σ-similarity and the structural properties of the input graphs. The structural property may also be defined in terms of topological similarity and then both can be woven into a higher level similarity function to compute overall graph similarity. In this dissertation, we quantify the quality of the overall similarity in terms of the similarity of the matched nodes and the matched edges as follows.

Definition 5.1.2 [Graph Similarity] Let $G = (V_g, E_g)$ and $H = (V_h, E_h)$ be two graphs, ε is minimum similarity threshold, σ is a domain dependent node similarity function of the form $\sigma : (V \to L \times V \to L) \to [0,1]$, ε and λ respectively are node and structural dissimilarity tolerances, and ϕ is a vertex selection function of the form $\phi : V_g \to V_h$ such that $\forall v_g(v_g \in V_g(\phi(v_g) = v_h \Rightarrow \sigma(\mu(v_g), \mu(\phi(v_g))) \geq (1-\varepsilon)))$[§]. Then, the conceptual similarity γ between two graphs G and H is defined as the product of their aggregate node similarity γ_σ and structural closeness γ_e as follows

$$\gamma(G,H) = n \times \gamma_\sigma(G,H) \times \gamma_e(G,H)$$

where

$$\gamma_\sigma(G,H) = \sum_{\forall v_g(v_g \in V_g),\ \sigma(\mu(v_g),\mu(\phi(v_g))) \geq (1-\varepsilon)} \frac{\sigma(\mu(v_g), \mu(\phi(v_g)))}{|V_g|},$$

$$\gamma_e(G,H) = \sum_{\forall v_1,v_2(v_1,v_2 \in V_g,\ edge(v_1,v_2) \in E_g)} \frac{I(edge(\phi(v_1), \phi(v_2)))}{|E_g|}$$

and $\gamma_e(G,H) \geq (1-\lambda)$ holds.

In the above definition, $I : E_h \to \{0,1\}$ is an indicator function which assumes 1 if the parameter $edge(\phi(v_1), \phi(v_2)) \in E_h$, assumes 0 otherwise. Functionally, I verifies if an edge in H

[§]Also let $\phi^{-1} : V_h \to V_g$ be the dual of the function ϕ such that $\phi^{-1}(\phi(v)) = v$. Furthermore, $\phi(v_h) = v_h$ whenever $V_h = V_g$.

corresponds to an edge in G under ϕ. Furthermore, n is the number of matched nodes in H. Incorporating n in the definition enables extending the notion of similarity for pattern queries, where the number of matched nodes (e.g., Kleene closure) has a significant role in the overall matching process. In the remainder of the presentation, we follow the convention of labeling nodes with identical alphabetical prefixes if they have high σ-similarity, data graph nodes are labeled with lower case letters and query graph nodes are labeled with capital letter prefixes.

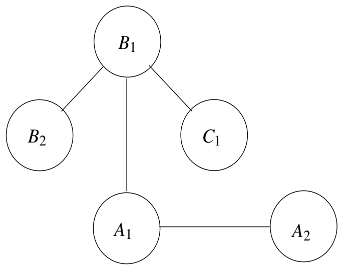
Intuitively, if we ignore the influence of the selection function ϕ in the above definition for the moment, the graph similarity amounts to the product of the similarity of randomly selected nodes and the fraction of the edges they form in the target graph. The lack of a guidance on the node selection forces an algorithm to search a large space of possible networks for subgraphs similar enough in the target graph, and then select the highest k similar subgraphs at a huge expense. However, the introduction of the functions ϕ, and I, and the structural dissimilarity tolerance λ in definition 5.1.2 acts as a smart node selection guide and help us prune the search space significantly with the help of the indexing function χ (in definition 5.2.1) and the randomization function ρ (in definition 5.2.2). The tolerance λ controls the degree of missing edges we allow based on the notion of *gate nodes* (in definition 5.2.4) that ensures connectivity.

5.2 Top-*k* Graph Matching in AtoM

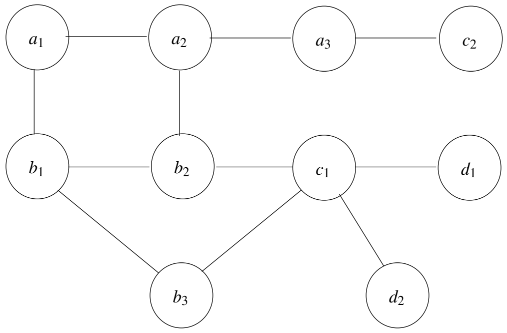
In this section, we introduce our algorithm AtoM consisting of three distinct steps using the example in figure 5.3, for expository purposes. We develop the required concepts formally as we build up to discuss the algorithm.

5.2.1 Step One: Summarization

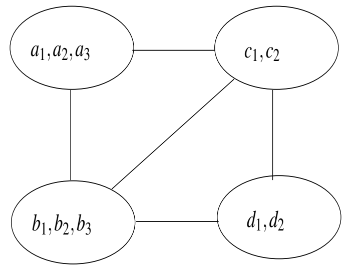
The first step in the matching process involves summarizing the data graph in the context of the query graph. The implication is that for every query graph, the summarized data graph will be different. The summarization process involves two steps. In the first step, we collapse identical types of nodes into a single node, called a bucket, and redraw the graph ignoring their underlying



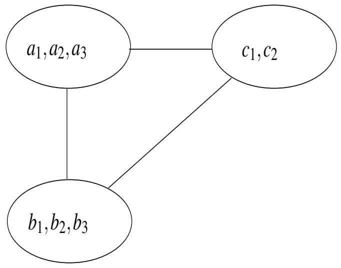
(a) Query graph.



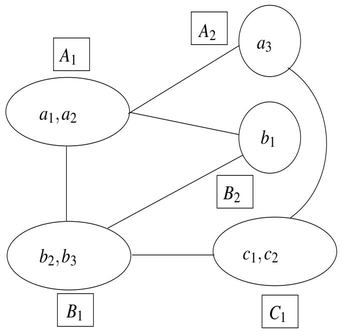
(b) Data graph.



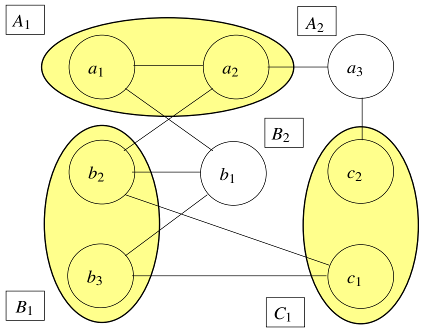
(c) Collapsed data graph.



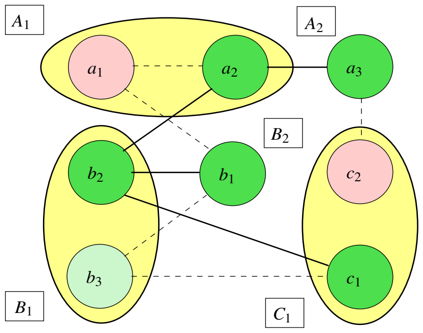
(d) Filtered collapse graph.



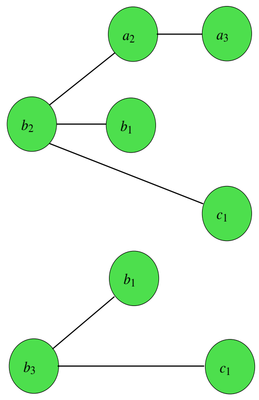
(e) Randomized summary graph.



(f) Structure under the summary graph.



(g) Matched graph: shown in green nodes and solid lines, and randomized splitting shown in yellow.



(h) Projection graphs: all possible matches of Q over the randomization in figure 5.3(e).

Figure 5.3: Example showing matching process of AtoM.

edge relationships. The collapsed graph corresponding to the query graph in figure 5.3(a) and data graph in figure 5.3(b) is shown in figure 5.3(c). Formally, we define the summarization as follows.

Definition 5.2.1 [Buckets] Let σ be a similarity function, and $G = (V,E)$ be any graph. Then χ is an *indexing function* of the form $\chi : L \rightarrow \mathbb{N}$ such that $\forall l_1, l_2 \in L$, $\chi(l_1) = \chi(l_2)$, whenever $\mu(v_1) = l_1$ and $\mu(v_2) = l_2$ and $\sigma(\mu(v_1), \mu(v_2)) \geq (1-\varepsilon)$, and χ^{-1} is its inverse, called the *clustering function*, i.e., $\chi^{-1}(n) \in 2^L$ such that $\forall n_1, n_2 \in \mathbb{N}, \chi^{-1}(n_1) \cap \chi^{-1}(n) = \emptyset$, and $\forall l, l' \in \chi^{-1}(n)$, $\mu(v) = l, \mu(v') = l'$, $\sigma(\mu(v), \mu(v')) \geq (1-\varepsilon)$. When disambiguation is needed, we subscript functions χ and χ^{-1} with V to indicate that they are over the labels corresponding to the vertices V, i.e., χ_V and χ_V^{-1} respectively.

The mapping $\chi(l)$ is called a *bucket, index* or *type* of node v with which it is associated, i.e., $\mu(v) = l$. Since similar nodes map to identical buckets, and the buckets do not overlap, χ acts as a partitioning function, and each such partition is given by χ^{-1}, i.e., for every $l \in \chi^{-1}(n)$, $\chi(l) = n$ holds. In figure 5.3(c), the node $\{a_1, a_2, a_3\}$ is one such bucket, say bucket 1. Without loss of any generality, we can represent all the nodes in bucket 1 as a_i, and its type or bucket index as A for ease of reference.

Once the summary graph is created, we draw an edge between two buckets in figure 5.3(c) if two nodes in the underlying data graph have an edge between them as well which are members of these buckets. For example, since there is an edge between c_1 and b_3 in the data graph in figure 5.3(b), we have and edge between buckets labeled $\{c_1, c_2\}$ and $\{b_1, b_2, b_3\}$. We then remove the nodes from each of the buckets that does not meet a given similarity threshold. In other words, the nodes in the filtered collapsed graph will have high domain similarity, i.e., σ-similarity. Since the bucket $\{d_1, d_2\}$ is not similar to any of the query graph nodes (i.e., A_i, B_j or C_k), we remove this node and all incident edges from the collapsed graph, as shown in figure 5.3(d).

In the next step of summarization process, we try to recreate the query graph from the filtered collapsed graph as follows. In this step, our goal is to create exactly the same number of nodes in the query graph[¶] by picking nodes from each of the buckets in figure 5.3(d). One of the candidate summary graphs is shown in figure 5.3(e). In this figure, we have five nodes labeled $\{a_1, a_2\}$, a_3, $\{b_2, b_3\}$, b_1, and $\{c_1, c_2\}$ corresponding to query graph nodes A_1, A_2, B_1, B_2 and C_1 respectively.

[¶]If the query graph had a set of nodes of type E, for example, that did not have corresponding e nodes in data graph, we will not have a node corresponding to E in the collapsed graph. In that case, we will proceed with the summarization with less than identical number of query graph nodes.

We create this graph by random partitioning of the buckets and by maintaining the same edge retention convention we have adopted before, i.e., an edge between two nodes is included in the graph if two members in the underlying data graph have an edge. In this figure, we partitioned $\{a_1,a_2,a_3\}$ into $\{a_1,a_2\}$, and a_3, and $\{b_1,b_2,b_3\}$ into $\{b_2,b_3\}$ and b_1 since we needed two A and two B type nodes. We do not partition $\{c_1,c_2\}$ as we need only one C type node. Formally,

Definition 5.2.2 [Randomized Summary Graphs] Let σ be a node similarity function, and $Q = (U,E_q)$ and $D = (W,E_d)$ respectively be the query and data graphs. Then ρ is a randomization function of the form $\rho : U \to 2^W$ such that for every $u \in U$, (i) $\forall w \in \rho(u), \chi(\mu(u)) = \chi(\mu(w))$, (ii) $\forall u' \in U, u \neq u', \rho(u) \cap \rho(u') = \emptyset$, (iii) $\rho(u)$ possibly empty, and (iv) $\rho(u) \subseteq \chi_W^{-1}(\chi_U(\mu(u)))$, hold. A *randomized graph R* with respect to Q and D is a graph of the form $(\mathbf{X},E_r)$ such that for every $X \in \mathbf{X}, X = \rho(u)$, and for every $e = \langle X_1,X_2\rangle \in E_r \Leftrightarrow \exists w_1,w_2(w_1 \in X_1 \wedge w_2 \in X_2 \wedge \langle w_1,w_2\rangle \in E_d)$.

Intuitively, ρ associates with each node in the query graph Q a subset of the nodes in a data graph bucket of the same type and nothing else, i.e., A_is will be matched with only subsets of a_js. Notice that once ρ partitions the buckets and generates a possible skeleton of the candidate subgraph, we have reduced the search space to be explored to a very small set of data nodes corresponding to each query node that are already known to be similar enough. At this stage the choice of data nodes for the function ϕ in definition 5.1.2 is localized to each partitioned buckets and help us ignore matching all other nodes that are not in the bucket $\rho(u)$ resulting in a significant saving, i.e., $\phi(u) \in \rho(u)$. In figure 5.3(e), $\rho(A_1) = \{a_1,a_2\} \subset \chi_W^{-1}(\chi_U(\mu(A_1)))$, whereas in figure 5.4, $\rho(A_1) = \{a_1\} \subset \chi_W^{-1}(\chi_U(\mu(A_1)))$, giving us the possible node selections $\phi(A_1) = a_1$ and $\phi(A_1) = a_2$ for the former, and only $\phi(A_1) = a_1$ for the latter case respectively.

Pruning Aids

It is interesting to note that several useful properties hold in the AtoM algorithm described thus far. For example, it should be evident that the generated summary graphs are potential candidates for further exploration. However, the candidate summary graphs are subjected to early and frequent filter operation in order to reduce the number of "explore-worthy" summary graphs. The

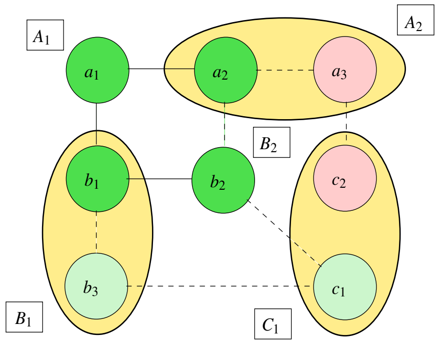


Figure 5.4: An alternate summary and matched graph.

following theorem characterizes the filtration criterion which suggests that if a summary graph fails to meet the minimum structural similarity requirement, none of its project graphs would. Hence, we could discard the candidate summary graph and exclude it from further exploration.

Theorem 5.2.3 *Let $Q = (U, E_q)$, $D = (W, E_d)$ and $R = (\mathbf{X}, E_r)$ be the query graph, data graph and the randomized summary graph respectively. Also let $G \in \pi(R)$ be any projection graph of R. Then $\forall G, G \in \pi(R)$,*

$$\gamma_e(R,Q) \leq (1-\lambda) \Rightarrow \gamma_e(G,Q) \leq (1-\lambda)$$

Proof: We use Reductio ad absurdum to establish this theorem. We know

$$\gamma_e(R,Q) = \sum_{\forall x_1,x_2 \in \mathbf{X},\ edge(x_1,x_2) \in E_r} \frac{I(edge(\phi(x_1),\phi(x_2)))}{|E_q|} \tag{5.1}$$

where $\phi(x_1), \phi(x_2) \in V_q$. From the definition of $R = (\mathbf{X}, E_r)$, we know that it is formed by the following criteria

> For $x_1 \in \mathbf{X}$ and $x_2 \in \mathbf{X}$, $e = \langle x_1, x_2 \rangle \in E_r \Leftrightarrow \exists w_1, w_2 (w_1 \in x_1 \wedge w_2 \in x_2 \wedge \langle w_1, w_2 \rangle \in E_d)$.

Let us assume that $\exists G$ such that $\gamma_e(G,Q) \geq \lambda$ but $\gamma_e(R,Q) \leq \lambda$ holds. We can restate this as $\gamma_e(G,Q) \geq \gamma_e(R,Q)$. If the two terms are equal, it does not violate the theorem. Let the set

$\mathbf{X} = \{X_1, X_2, \ldots, X_n\}$, where each $X_i = \{w_{i_1}, w_{i_2}, \ldots w_{i_m}\} \subseteq W$ for some n and m. Let us assume that $\gamma_e(G,Q) > \gamma_e(R,Q)$. This essentially means that $\exists w_i, w_j (w_i, w_j \in W)$ such that edge $\langle w_i, w_j \rangle$ exists but edge $\langle X_i, X_j \rangle$ is non-existent, where $w_i \in X_i, w_i \in X_2$. This is impossible because, For $X_1 \in \mathbf{X}$ and $X_2 \in \mathbf{X}$, $e = \langle X_1, X_2 \rangle \in E_r \Leftrightarrow \exists w_1, w_2 (w_1 \in X_1 \wedge w_2 \in X_2 \wedge \langle w_1, w_2 \rangle \in E_d)$ holds. In other words, since the edges of the summary graph are formed by the combined contribution of the actual nodes contained in the summary graph nodes, if the number of edges in R that match with Q is less than the threshold λ, there cannot be any combination of the nodes contained in G with higher number of matches. □

Summarization Algorithm

Using the definitions of σ-similarity, bucket and clustering functions, we are in a position to present two algorithms that form the core of AtoM. The first algorithm, algorithm 6, generates the relevant summary graph from a query graph and a data graph, given the similarity values between the labels L, either as a set of triples of the form $\langle l_1, l_2, s \rangle$ where l_1 and l_2 are labels and s is the similarity between 0 and 1, or as a computable function.

Once the summary graph is constructed, we can then iteratively construct the randomized summary graphs as needed using algorithm 7. It should be clear that a randomized graph will not always contain the edge pattern or structure in the query graph. Therefore, it is necessary to iterate and randomize the summary graph several times. The exact number of iteration depends upon with how much certainty we would like to compute our matching. Higher the certainty, more likely it is that AtoM will return the best possible matches, but also higher will be the cost of matching. We will discuss this connection in section 5.2.2 and quantify the number of iterations needed as a function of certainty in theorem 5.2.6.

5.2.2 Step Two: Validation

While the random partitioning helps reconstruct the skeleton of the query graph on the data graph at the summary level, it is agnostic to the edge structure of the query graph and thus cannot ensure

Algorithm 6: Summary Graph

```
   Input: Data graph $D = (W, E_d)$, query graph $Q = (U, E_q)$, and dissimilarity threshold $\varepsilon$
   Output: Summary graph $S = (\mathbf{X}, E_s)$
1  Let $\mathbf{X} = \emptyset$, and $E_s = \emptyset$;
2  for $\forall u \in U$ do
3      if bucket $X = \chi(\mu(u))$ does not exist then
4          create bucket $X$;
5          $\mathbf{X} = \mathbf{X} \cup \{X\}$;
6  for $\forall w \in W$ do
7      if $\exists u \in U$ such that $\sigma(\mu(u), \mu(w)) \geq (1 - \varepsilon)$ then
8          include $w$ in bucket $X = \chi(\mu(u))$
9  for $\forall w_i, w_j \in W$ do
10     if $\langle w_i, w_j \rangle \in E_d$, $\exists u_i, u_j$, such that $w_i \in \chi(\mu(u_i)) = X_i$, $w_j \in \chi(\mu(u_j)) = X_j$ and
       $X_i \neq X_j$ then
11         include $\langle X_i, X_j \rangle \in E_s$;
12 return $S = (\mathbf{X}, E_s)$;
```

correspondence of the edge structures of the query and data graphs. From a randomized graph, a node in a summary graph cannot contribute to the overall structure of a query graph if it cannot participate in an anticipated edge in the query graph (after mapping). For example, the node a_1 in summary node $\rho(A_1)$ in figure 5.3(e) is such a node because it does not connect with any node in $\rho(B_1)$ or $\rho(A_2)$. But, the node a_2 potentially can contribute to two edges $\langle A_1, B_1 \rangle \in E_q$ and $\langle A_1, A_2 \rangle \in E_q$ respectively as data edges $\langle a_2, b_2 \rangle \in E_d$ and $\langle a_2, a_3 \rangle \in E_d$. The potentially contributor node a_2 is called a gate node as formalized below. In the matching process, we are then able to eliminate non-gate nodes such as a_1, to further reduce the choices for the function ϕ.

Definition 5.2.4 [Gate Nodes] Let $Q = (U, E_q)$, $D = (W, E_d)$ and $R = (\mathbf{X}, E_r)$ respectively be a query graph, data graph and one of their randomized graphs. A node $x \in X \in \mathbf{X}$ is called a *gate node* if $x \in \rho(u), u \in U$, and there exists distinct x', u' such that $x' \in X' \in \mathbf{X}$, $x' \in \rho(u'), u' \in U$ and $\langle u, u' \rangle \in E_q \Rightarrow \langle x, x' \rangle \in E_d$.

Algorithm 7: Randomized Graph

```
   Input: Query graph $Q = (U, E_q)$, data graph $D = (W, E_d)$ and summary graph $S = (\mathbf{X}, E_s)$
   Output: Randomized summary graph $R = (\mathbf{Y}, E_r)$
1  Let $\mathbf{Y} = \emptyset$, and $E_r = \emptyset$;
2  for $\forall u \in U$ do
3      create bucket $Y_u$;
4      $\mathbf{Y} = \mathbf{Y} \cup \{Y_u\}$;
5  for $\forall X \in \mathbf{X}$ do
6      randomly split $X$ into non-empty subsets $s_i$s such that $1 \le i \le |\chi_U^{-1}(\chi(\mu(x)))|$ and $x$ is
       any member of $X$;
7      include each $s_i$ in distinct $Y_u$ such that $\chi(\mu(u)) = \chi(\mu(x))$ and $x \in s_i$;
8  for $\forall x, y (x, y \in W, x \neq y)$ do
9      if $\langle x, y \rangle \in E_d$, $\exists w, z (Y_w \in \mathbf{Y}, Y_z \in \mathbf{Y}, w \neq z)$, such that $x \in Y_w$ and $y \in Y_z$ then
10         include $\langle Y_w, Y_z \rangle \in E_r$;
11         if $\langle \phi^{-1}(x), \phi^{-1}(y) \rangle \in E_q$ then
12             increment number of valid incident edges $\psi(\phi^{-1}(x), x)$ and $\psi(\phi^{-1}(y), y)$
13             if $\psi(\phi^{-1}(x), x) \times (1 - \eta) \ge \psi(\phi^{-1}(x), \phi^{-1}(x))$ then
14                 mark $x$ as sufficiently connected gate node.
15             if $\psi(\phi^{-1}(y), y) \times (1 - \eta) \ge \psi(\phi^{-1}(y), \phi^{-1}(y))$ then
16                 mark $y$ as sufficiently connected gate node.
17 return $R = (\mathbf{Y}, E_r)$;
```

The above definition identifies nodes b_1, b_2, b_3, a_2, a_3 and c_1 as gate nodes (i.e., a_1 and c_2 as non-gate nodes) in figure 5.3(e). But in figure 5.4, only nodes a_3 and c_2 are non-gate nodes. Note that choosing a gate node does not guarantee a perfect match. For example, choosing to match B_1 with b_3, i.e., $\phi(B_1) = b_3$, only results in selecting the matching edge $\langle b_3, c_1 \rangle$ as we are unable to match other edges corresponding to $\langle B_1, B_2 \rangle$, $\langle B_1, A_1 \rangle$ and $\langle A_1, A_2 \rangle$, resulting in only 25% matching, i.e., $\gamma_e(Q, D) = .25$. If $\lambda = 25\%$, we are required to admit only matches equal to or above 75% and hence the matched graph $(\{b_3, c_1\}, \{\langle b_3, c_1 \rangle\})$ will be rejected. But the match $(\{b_1, b_2, a_1, a_2\}, \{\langle b_1, b_2 \rangle, \langle b_1, a_1 \rangle, \langle a_1, a_2 \rangle\})$ will meet the edge matching threshold of 75% and will be selected.

However, we can be more selective and admit a subset of the gate nodes yet meet the $(1 - \lambda)$ threshold for the edge similarity γ_e in definition 5.1.2 without compromising the set of top-k

matches we intend to compute. We can do so by pushing down the threshold to the gate node level, and require that each gate node be part of $(1-\lambda)\%$ of edges to its query graph counterpart. In fact, this selection condition can be made part of yet another user supplied parameter that would range between zero and λ forcing respectively for us to find only matches that allow no structural deviation to $\lambda\%$ structural deviation (missing edges) from the query graph. We thus introduce the concept of *sufficiently connected gate nodes* as follows.

Definition 5.2.5 [Sufficiently Connected Gate Nodes] Let $Q = (U, E_q)$ be a query graph, $R = (\mathbf{X}, E_r)$ be its randomized summary graph, $x \in X \in \mathbf{X}$ be a gate node, and $u \in U$ be its corresponding query graph node, i.e., $x \in \rho(u)$. Also let η be an admissible connectivity error tolerance such that $0 \leq \eta \leq \lambda$, and n be the number of edges in E_q involving node u. Then, x is *sufficiently connected* if E_r contains at least $n \times (1-\eta)$ edges of the form $\langle x, x' \rangle \in E_r$.

While the parameter η does not impact the ranking of the matched graphs in our top-k selection, it potentially can prevent some of the low scoring matches to be included in the selection as η approaches zero, i.e., we require 100% structural match. Thus, choosing a lower η helps prune matches early that will eventually be low scoring, and thus speed up the response time.

In other words, definitions 5.1.2 and 5.2.5 allow the control of structural similarity at two different levels. For any data graph $D = (W, E_d)$, query graph $Q = (U, E_q)$, a data graph node $w \in W$ in a summary graph node, its relationship to η is quite simple. Given w, the corresponding query graph node is $\phi^{-1}(w) = u \in U$, and $n = |E_u \subseteq E_q|$ when for every edge in E_u is of the form $\langle u, y \rangle$ for some $y \in U$. Given that ϕ is not a total function and may only map a subset of nodes in U to W, it is likely that a subset of edges in E_u will have mappings of the form $\langle \phi(u) = w, \phi(y) \rangle$, and equivalently $\langle \phi^{-1}(w) = u, \phi^{-1}(v) \rangle$ for some $v \in W$. The subset of edges in E_u, $\bar{E_u}$, having a mapping in E_w thus constitutes the fraction $\frac{|\bar{E_u}|}{n} = \eta$, the structural error tolerance, i.e., the smaller the η, larger is the error. We define a function ψ such that any two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ and a mapping function ϕ, and two nodes $v_1 \in V_1$ and $v_2 \in V_2$, it returns the number of edges of the form $\langle \phi(v_1), \phi(v) \rangle = \langle v_2, \phi(v) \rangle$ these two nodes share under the mapping function ϕ from the element of V_1 to V_2. Alternately, for a query node u and data node w,

$\psi(u,w) = |\bar{E}_u| = n \times \eta$. A caveat is that for $u = w$, i.e., both u and w come from the same query or data graph, $\psi(u,u) = |E_u| = n \times \eta$, since u will share all its edges with u under ϕ. Hence, ϕ acts as an identity function for same types of nodes.

Cost of Matching versus Accuracy

The grouping of similarly labeled nodes into summary graphs, and their subsequent randomization is aimed at reducing candidate generation and testing for match of the whole graph with the knowledge that randomized partitioning is stochastically is a structure preserving operation. In some sense, we are trying to generate a method that allows partition components are compose the filtered them as much as possible to reconstruct a match that is sufficiently similar above a threshold. By identifying sufficiently connected gate nodes as we randomize a summary node, we have now reduced the matching into a look up operation. All we have to do now is maximally pick one such node from each summary node in R corresponding to the query graph edges.

But inherent in the random partitioning is the possibility that the data graph edge structure embedded in the partitioned graph may not retain a structure sufficiently similar to the query graph. Therefore, it is necessary that that partition the summary graph in all possible ways to find all possible structures. Unfortunately, the number of such randomized graphs and possible candidates is the product of a large number of possibilities, i.e., $\prod_{i=1}^{|U|} \frac{n_i!}{(n_i-m_i)!}$ where each n_i is the number of data graph labels in each bucket and m_i is the number of query graph labels for the same bucket in the randomized graph.

Theorem 5.2.6 *Let $Q = (U,E_q)$, $D = (W,E_d)$ and $S = (\mathbf{X},E_r)$ be the query graph, data graph and the summary graph respectively. Also, for every node $u_i \in U$ and $w_j \in W$, let the number of query and data nodes mapped to each bucket j be $n_j = |\chi_W^{-1}(\chi_U(\mu(u_i))|$ and $m_j = |\chi_U^{-1}(\chi_U(\mu(u_i))|$ respectively. Then, if a matching topology of Q exists in D and the total number of buckets in S is k, the probability p of the summary graph S retaining this match is given by*

$$p = \frac{\prod\limits_{1 \le i \le k} \frac{n_i!}{(n_i - m_i)!}}{\prod\limits_{1 \le i \le k} {m_i}^{n_i}} \tag{5.2}$$

Proof: If the query graph Q exists in the data graph D then there are m_i vertices in D corresponding to nodes with label l_i in Q. These vertices need to be in different buckets in the summary graph S. The probability that these nodes will be in different groups is given by

$$\frac{m_i}{m_i} \times \frac{m_i - 1}{m_i} \times \ldots \times \frac{m_i - n_i + 1}{m_i} = \frac{\frac{n_i!}{(n_i - m_i)!}}{{m_i}^{n_i}}$$

Calculating for all $k = |\cup_{u \in U} \chi_U^{-1}(\chi_U(\mu(u_i))|$ labels we get

$$\begin{aligned} p &= \frac{\frac{n_1!}{(n_1 - m_1)!} \times \frac{n_2!}{(n_2 - m_2)!} \times \ldots \times \frac{n_k!}{(n_k - m_k)!}}{{m_1}^{n_1} \times {m_2}^{n_2} \times \ldots \times {m_k}^{n_k}} \\ &= \frac{\prod\limits_{1 \le i \le k} \frac{n_i!}{(n_i - m_i)!}}{\prod\limits_{1 \le i \le k} {m_i}^{n_i}} \end{aligned}$$

□

It is possible to establish a link between the number of times we need to randomize a summary graph increase the probability of including a matching pattern in the randomized graph, and thus reduce the chance of not finding a match that exists due to improper partitioning. Conversely, iterating less will increase the chance of missing a valid match. From theorem 5.2.6 it follows that the probability that the query structure (if exists in the data graph) is not retained in the summary graph is $1 - p$. Hence, if we perform the randomized summarization process t times the probability that the structure is missed in all t iterations is $(1 - p)^t$. Thus, to bound the acceptable error $\hat{\varepsilon}$ where $\hat{\varepsilon} = (1 - p)^t$, the number of iteration required is $\frac{\log_e \hat{\varepsilon}}{\log(1-p)}$. However, since we are interested in approximate matches, the number above serves as an the upper bound on the number of iterations.

$$t \le \frac{\log_e \hat{\varepsilon}}{\log(1 - p)} \tag{5.3}$$

The summarization process is invoked t times, we notice that the summarization process only affects nodes $x_i \in X$, such that $\chi(\mu(x_i)) = \chi(\mu(u_j))$, where $\forall u_j \in U$, $\chi(\mu(u_j))$ is the same and $\forall u_j |\{u_j\}| > 1$.

Hence after the first iteration the execution of the method can be expedited in the subsequent iteration by only concentrating on nodes x_i. Similarly, gate node calculation and filtering out non-gate nodes can be expedited since the role of gate node can change for nodes with the same property.

Validation Algorithm

The way we have designed the randomized summary graph construction algorithm, we are able to embed the validation step into the randomization process. For example, at step 16 in algorithm 7, we count for each node with how many anticipated edges it is connected and identify the node as sufficiently connected gate node, if it is. By doing so, we aid identification of valid edges as a lookup operation during the extraction phase, and avoid a separate algorithm for validation phase.

5.2.3 Step Three: Extraction

Since the ultimate goal is to find the best possible k matches, or top-k similar graphs, it is useful to gather all the graphs that meet the intended similarity threshold anticipated in the query that can be extracted from the summary graph. Once they are collected, we rank them in a dynamic fashion to only return the most similar k graphs. The set of graphs meeting or exceeding the similarity threshold is called the *projection graphs* captured formally in the definition below.

Definition 5.2.7 [Projection Graph] Let $Q = (U, E_q)$, and $D = (W, E_d)$ be a query and a data graph respectively, and $R = (\mathbf{X}, E_r)$ be one of their randomized graphs, i.e., $\forall X \in \mathbf{X}, \exists u \in U$, $X = \rho(u) \subseteq W$ holds[‖]. Then, the projection graphs P of R, denoted $P = \pi(R) = \bigcup_i \{(Y_i, E_{p_i})\}$, is the set of all graphs such that for every i, (i) $|Y_i| \leq |U|$, (ii) $\forall y_1, y_2 (y_1, y_2 \in Y_i \wedge y_1 \in \rho(u_1) \wedge y_2 \in$

[‖]Note that $\mathbf{X}$ is a set of set of nodes and E_r are edges involving two such sets, and $\rho(u) \subset \chi_W^{-1}(\chi_U(u))$ holds where $\chi_W^{-1}(\chi_U(u)) \subseteq W$.

$\rho(u_2) \Rightarrow y_1 = y_2)$ whenever $u_1 = u_2$, (iii) for every edge $\langle x,y\rangle \in E_{p_i}$, $\exists u_1,u_2(u_1,u_2 \in U \wedge x \in \rho(u_1) \wedge y \in \rho(u_2) \wedge \langle u_1,u_2\rangle \in E_q) \wedge u_1 \neq u_2$, (iv) both Y_i and E_{p_i} are maximal, and (v) all vertices in Y_i are sufficiently connected gate nodes.

Speedup without Matching Quality Loss

The definition 5.2.7 above characterizes the individual "similar-graph" candidates and how they are extracted from the summary graph. In the example in figure 5.3(h), two such projected subgraphs are depicted. In other words, projection graphs P contain all the matched subgraphs of D with sufficient node and structural similarity with Q contained in the randomized summary graph R of D. The degree of structural similarity of the projected graphs and the connectivity tolerance follows the relationship captured in the theorem 5.2.8 below which we leverage to prune candidate graphs that ensures that discarding all projection graphs involving gate nodes with less than $\eta = \lambda$ connectivity does not impact the set of similar graphs chosen by AtoM. Conversely, we could potentially set η lower to restrict generating projections that will not make it to the top-k set even though they will meet the λ threshold. Note that the filtering using λ is after the projection graphs are generated incurring processing costs, while filtering using η saves validation costs. So, η can be used to select only a few k candidates that will make the top-k matches and save time and serve as an additional pruning device. We use the relationship between η and ψ discussed in section 5.2.2 to formalize the effectiveness of pruning using sufficiently connected gate nodes in theorem 5.2.8.

Theorem 5.2.8 *Let $Q = (U,E_q)$, $D = (W,E_d)$, $R = (\mathbf{X},E_r)$, and $P = \pi(R)$ respectively be a query graph, a data graph, their randomized summary graph, and all projection graphs of R. Also let λ be structural dissimilarity tolerance. Then for each node $x \in X \in \mathbf{X}$ and $\bar{u} = \phi^{-1}(x)$, we can filter out node x as non-gate node if*

$$\psi(\bar{u},x) < 2 \times |E_q|(1-\lambda) - \sum_{\forall u \in (U \setminus \{\bar{u}\})} \psi(u,\phi(u)) \tag{5.4}$$

Proof: From the definition of λ, a subgraph is deemed valid for further exploration if the total number of matched edges is greater or equal to $|E_q|(1-\lambda)$. In other words, it is discarded if the total number of matched edges is less than $|E_q|(1-\lambda)$. In the best case the total number of matched edges equals to $|E_q|$. Therefore, the total number of matched edges

$$\begin{aligned}M &= \sum_{\forall u \in U} \psi(u,\phi(u)) - N \\ &\geq \psi(\bar{u},x) + \sum_{\forall u \in (U \setminus \{\bar{u}\})} \psi(u,\phi(u)) - N \\ &= \frac{\psi(\bar{u},x) + \sum_{\forall u \in (U \setminus \{\bar{u}\})} \psi(u,\phi(u))}{2}\end{aligned}$$

where, N is the number of shared edges. From hand-shaking lemma in graph theory, we know every edge contributes to the degree count of two distinct nodes and since an edge is shared between two nodes it is counted twice in $\sum_{\forall u \in (U \setminus \{\bar{u}\})} \psi(u,\phi(u))$. We have,

$$\begin{aligned}\frac{\psi(\bar{u},x) + \sum_{\forall u \in (U \setminus \{\bar{u}\})} \psi(u,\phi(u))}{2} &\leq |E_q|(1-\lambda) \\ \psi(\bar{u},x) &\leq 2 \times |E_q|(1-\lambda) - \sum_{\forall u \in (U \setminus \{\bar{u}\})} \psi(u,\phi(u)))\end{aligned}$$

□

Extraction and Ranking Algorithm

In the final step of our algorithm, we explore this summary graph to extract possible matching contenders. In this Figure 5.3(g), we see that the gray node can be filtered out as it does not contribute to the matching criteria. This is due to the fact that the gray node participates in edges that are not present in the query graph and they do not participate in nodes that are present in the query graph. For example in 5.3(g), one such matching is depicted, where the dark nodes represent the matched nodes. Figure 5.4 depicts a different randomized summarization of the data

Algorithm 8: Projection Graphs

```
Input: Randomized summary graph $R = (\mathbf{Y}, E_r)$, $Q = (U, E_q)$
Output: Projection graph $P = \pi(R)$
1 Let $\mathbf{P} = \emptyset$;
2 while true do
3     Extract graph $G = (Y_i, E_{p_i})$ where $|Y_i| \leq |U|$ such that
          1. $\forall y_1 \in Y_i$, $y_1$ is a sufficiently connected gate node,
          2. $\forall y_1, y_2 (y_1, y_2 \in Y_i \wedge y_1 \in \rho(u_1) \wedge y_2 \in \rho(u_2) \Rightarrow y_1 = y_2)$ whenever $u_1 = u_2$,
          3. for every edge $\langle x, y \rangle \in E_{p_i}$, $\exists u_1, u_2 (u_1, u_2 \in U \wedge$
             $x \in \rho(u_1) \wedge y \in \rho(u_2) \wedge \langle u_1, u_2 \rangle \in E_q)$, and
          4. both $Y_i$ and $E_{p_i}$ are maximal.
      $P = P \bigcup G$;
4 return $P$;
```

graph. From an intuitive standpoint, once a matching summary graph is chosen, it is possible to explore the summary graph nodes following the edges emanating from the query graph to select one node from each bucket at a time. There could be a number of possible sets of nodes that could potentially be extracted. These extracted components are subsequently evaluated according to definition 5.1.2 to extract the best possible matches.

Since definition 5.1.2 allows matching two graphs in many different ways, multiple matching scores and matches can be generated iteratively. A useful exercise is to rank the matches and accept only the best k matches, called top-k matches. We define a function η of the form $\eta : 2^E \rightarrow \mathbb{N}$ that given a set of edges, returns the number of unique nodes that contributes to the matched edges toward formally capturing the concept of top-k similar graph matching as follows. Intuitively, for data graph D and a query graph Q, our objective is to identify k sub-graphs $\mathbb{S}$ from D such that the similarity between Q and each $s_i \in \mathbb{S}$, $1 \leq i \leq k$, are the maximum among all possible subgraphs of D and higher than a threshold λ, as stated in definition 5.2.9.

Definition 5.2.9 [Top-k similar graphs] Let $D = (W, E_d)$ and $Q = (U, E_q)$ respectively be the data and query graphs. Let $\mathbb{S}$ be the set of all possible subgraphs of D. Then for a given k and similarity threshold λ, top-k matching of Q with D is the set of graphs $\mathbb{K}$ such that $\mathbb{K} \subseteq \mathbb{S}$, $|\mathbb{K}| \leq k$ and $\forall g(g \in \mathbb{K}, \forall s(s \in (\mathbb{S} \setminus \mathbb{K})(\gamma(g,Q) > \gamma(s,Q) \wedge (\gamma_e(g,Q) \geq \lambda \wedge \gamma_\sigma(g,Q) \geq \eta(E_Q) \times (1-\varepsilon)))))$.

Table 5.1: Example Similarity Matrix

	a_1	a_2	a_3	b_1	b_2	b_3	c_1	c_2
A_1	1.0	0.9	0.8	0.1	0.09	0.08	0.1	0.09
A_2	0.9	1.0	0.9	0.09	0.1	0.09	0.09	0.1
B_1	0.1	0.09	0.08	1.0	0.9	0.8	0.1	0.09
B_2	0.09	0.1	0.08	0.9	1.0	0.8	.09	0.1
C_1	0.1	0.09	0.08	0.1	0.09	0.08	1.0	0.9

For example, for Figures 5.3(a) and 5.3(b), given the above similarity table, the top 2 subgraphs chosen by AtoM consists of nodes $\{\phi(A_1) = a_2, \phi(A_2) = a_3, \phi(B_2) = b_1, \phi(B1) = b_2, \phi(C_1) = c_1\}$ and $\{\phi(A_1) = a_2, \phi(A_2) = a_1, \phi(B_2) = b_1, \phi(B_1) = b_2, \phi(C_1) = c_1\}$ with a score of 4.6. Similarly the set consisting of nodes $\{\phi(A_1) = a_1, \phi(A_2) = a_2, \phi(B_1) = b_1, \phi(B_2) = b_2, \phi(C_1) = c_1\}$ has a score of 3.75 and would be ranked lower than the above two. If $k = 1$, according to definition 5.2.9, AtoM will still return the first two matches as both have score 4.6, i.e., there are two graphs with a single score. This is because we made a design choice to not choose between matches with identical scores. But for $k = 2$, we still have the first two because in this case we will not have to make a choice yet, we can return 2 matches. To include the last match with a score of 3.75, we will need to set $k = 3$.

5.2.4 The overall algorithm

The main AtoM procedure in 9 accepts three error tolerance thresholds λ, η and ε in addition to the label similarity function σ, size of the number of top matches k and of course the data graph D and query graph Q. For exact label matches, we require that $\sigma(l_i, l_j) \in \{0, 1\}$, and for approximate matches, $\sigma(l_i, l_j) \in [0, 1]$. Note that if exact edge matching is required, we can set both λ and η equal to zero, and force AtoM to behave like a subgraph isomorph matching algorithm. Once supplied with the above parameters, using the query graph as the template, we morph the data graph into a condensed graph that assumes the structure of the query graph and proceed to find the matches in three distinct phases as shown in algorithm 9 with the help of three component procedures that implement the three steps as outlined in section 5.1 to capture the semantics of matching in AtoM.

Algorithm 9: AtoM

```
   Input: $D = (W, E_d)$, $Q = (U, E_q)$, similarity function $\sigma$, structural dissimilarity threshold
          $\lambda$, number of best matches $k$, label matching error tolerance $\varepsilon$, and structure
          matching error tolerance $\eta$;
   Output: Top $k$ matched graphs of $D$;
1  Initialize priority queue $\bar{Q}$ as empty;
2  Compute iteration limit $t$ using equation 5.3;
3  Compute summary graph $S =$ Summary Graph$(D, Q, \varepsilon)$;
4  for $i = 1$ to $t$ do
5      Compute randomized summary graph $R =$ Randomized Graph$(D, Q, S, \eta)$;
6      Compute projection graph $P =$ Projection Graph$(R)$;
7      for every graph $G \in P$ do
8          if $\gamma_e(G, Q) \geq (1 - \lambda)$ then
9              Insert $G$ in priority queue $\bar{Q}$;
10 Collect topmost $k$ graphs in queue $\bar{Q}$ into set $M$;
11 return $M$;
```

In this algorithm, $\bar{Q}$ is a priority queue that stores the set of edges of subgraphs of the data graph D in decreasing γ order from which the top k subgraphs is read off into M. The randomized summary graphs according to definition 5.2.2 are generated iteratively by calling the Randomize Graph procedure in line 5. The summary graph needed is computed once with a call to the Summary Graph procedure in line 3. The extraction phase is divided into two steps. First, all the valid matches are collected as projection graphs in line 6. If the projection graphs meet the structural similarity threshold, then they are pushed into the priority queue $\bar{Q}$ so that they are ranked according to the similarity function γ. In the final step in line 10, the top most k matches are read off of the queue $\bar{Q}$, completing the matching process.

5.3 Accounting for Edge Similarity

The notion of the graph similarity can be extended and tailored to cater for weighted edges as well. Protein-protein interaction networks, for example, can have numeric values associated with edges representing the confidence score of the interactions. We have extended the definition of graph similarity by introducing the notion of Υ function. We have replaced the identity function with this new function which essentially represents the edge similarity between the chosen data graph edge and the corresponding query graph edge. The denominator term κ is a normalization factor to cap the value of γ_e within $[0,1]$.

Definition 5.3.1 [Graph Similarity with Edge Labels] Let $G_1 = (V_{g_1}, E_{g_1})$ and $G_2 = (V_{g_2}, E_{g_2})$ be two graphs and ϕ be a mapping function. Then, the similarity γ between two graphs G_1 and G_2 under a mapping function ϕ is

$$\gamma(G_1,G_2) = n \times \gamma_\sigma(G_1,G_2) \times \gamma_e(G_1,G_2)$$

where

$$\gamma_\sigma(G_1,G_2) = \sum_{\forall v_1 \in V_{g_1}, \forall \phi(v_1) \in V_{g_2}} \frac{\sigma(v_1,\phi(v_1))}{|V_{g_1}|}$$

$$\gamma_e(G_1,G_2) = \sum_{\forall v_1,v_2 \in V_{g_1} s.t. \{v_1,V_2\} \in E_{g_1}} \frac{\Upsilon(\{\phi(v_1),\phi(v_2)\})}{\kappa}$$

5.4 Pattern Querying

A Query graph $Q = (V_q, E_q)$ is called a pattern graph if Q represents a set of graphs $\hat{Q}$ loosely conforming to the topological structure of Q and containing zero or more instances of certain nodes from Q, referred to as repeat-nodes.

For example, in Figure 5.1, the node *ST* refers to a repeat-node for the query graph. Thus in the extracted result multiple instances of *ST* is permitted to appear. AtoM can be morphed to handle pattern queries as well. To extend AtoM to handle query patterns, the repeat nodes in the same summary nodes and having a common neighbor from a different summary nodes are augmented with a pseudo-edge to connect with each other. The definition of projection graph is modified such that condition (iii) becomes: for every edge $\langle x, y \rangle \in E_{p_i}$, $\exists u_1, u_2 (u_1, u_2 \in U \wedge x \in \rho(u_1) \wedge y \in \rho(u_2) \wedge \langle u_1, u_2 \rangle \in E_q)$

The rest of the process remains same. Since the result obtained from pattern search may return a variable number of nodes, the similarity measure needs to be adjusted as well. We use a gaussian function to cater to the variable number of expected nodes in the matched graph.

$$\gamma(G_1, G_2, \theta_1, \theta_2) = \frac{1}{\theta_1 \times \sqrt[2]{2\pi}} \times e^{\frac{-1}{2} \times (\frac{(n-\theta_2)}{\theta_1})^2} \times \gamma_\sigma(G_1, G_2) \times \gamma_e(G_1, G_2) \tag{5.5}$$

Here θ_2 represents the expected number of nodes from the matching process, and θ_1 represents the decay parameter that penalizes matches with more of less number of nodes. For example, we can refer to arnetminer data set [91]. We ran AtoM to look for a team of atmost 4 researchers who have published together and have published either in VLDB and KDD or only KDD. We obtained the following set of researchers in the top results:

Table 5.2: Partial Matching result on DBLP data set

Martin Ester	Jiawei Han	Heikki Mannila	Marcel Holsheimer
Micheline Kamber	Jiawei Han	Heikki Mannila	Marcel Holsheimer
Charu Aggarwal	Jiawei Han	Heikki Mannila	Marcel Holsheimer
Martin Ester	Jiawei Han	Heikki Mannila	Usama Fayyad
Micheline Kamber	Jiawei Han	Heikki Mannila	Usama Fayyad
Charu Aggarwal	Jiawei Han	Heikki Mannila	Usama Fayyad
Usama Fayyad	Heikki Mannila	Jiawei Han	Micheline Kamber

5.5 AtoM in Map-Reduce Framework

Map-Reduce is a programming model catering to large data sets in a parallel and distributed manner on a cluster. Map-Reduce (or MR in short) framework is a two step procedure commencing with "Map" that generally is applied to filter or sort data and followed by "Reduce" step to perform summary or aggregation on a group of data. The infra-structure provides distributed data management including marshalling, parallelizing, communication and fault tolerance leaving the Map-Reduce framework free from the intricate details of the distributed system. The model is inspired by functional programming language model.

“Map” step: In the map step the input is divided into smaller sub-problems and distributed in processor nodes by the master node. The smaller problem is processed in the processor nodes and results are sent back to the master node.

“Reduce” step: The master node accumulates partial results from all the sum-problem and processes them in a batch.

In Map-Reduce frame-work each mapping operation is independent of the others and all maps can be executed in parallel. Analogously, a set of reducers can be executed in parallel provided that all outputs of the map operation that share the same key are presented to the same reducer at the same time, or that the reduction function is associative.

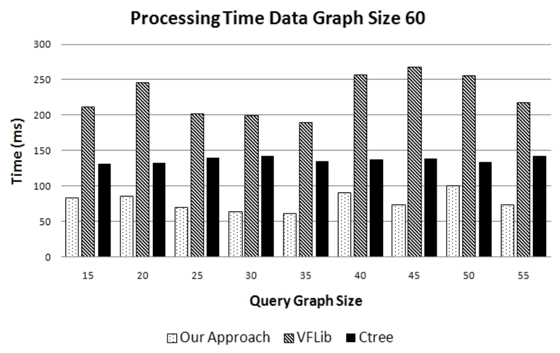
The algorithm can be easily extended and ported for Map-reduce implementation. The algorithm proceeds by segmenting the data graph into a set of mutually set of exclusive sections. This segmentation process is directed by the nodes of the query graph. AtoM algorithm lends itself to be seamlessly converted in to map-reduce framework. The summarization phase is driven by grouping the set of data graph nodes in multiple disjoint sets. These sets represent grouping of semantically similar nodes centered around one of the query graph nodes. Hence in the MR framework, data-graph nodes are compared with the query graph nodes and grouped together with the most similar one. This process is followed by a two-step map-reduce process to generate tuples of the form $\langle w_1, w_2, x_1, x_2 \rangle$ where w_1 and w_2 are data-graph nodes and x_1 and x_2 are the query graph nodes that they are mapped to. Note that here summary graph nodes are essentially semantic mapping of the query graph nodes for the data graph nodes, hence, we can refer to x_1 and x_2 as summary graph nodes without loss of generality. Finally the ProjectionGraph algorithm is a natural candidates for MR framework and has a direct MR connotation.

CHAPTER 6: EXPERIMENTAL RESULTS

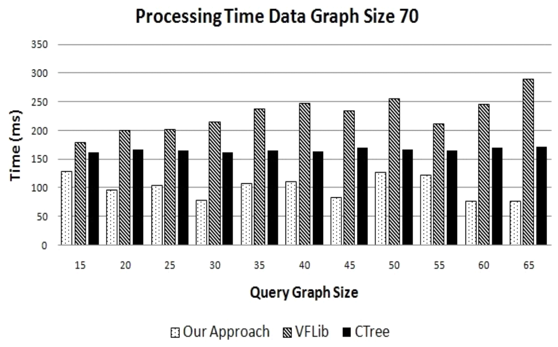
6.1 Experimental Results: TraM

The system has been implemented in Java. As the back-end database we used MySQL. In order to facilitate storage of graph database as well as the query graph and their labels, we used a number of database tables. Internally, the algorithm converts the labels of the nodes into positive integers. Thus in order to retain the mapping between the labels of the original graph and the translated labels, we keep two database tables *datanode(id, label)* and *querynode(id, label)*. Subsequently the entire graph is stored as pairs of ordered nodes representing the edges. In the tables *datagraph(source, destination, hop)* and *querygraph(source, destination, hop)*, the source and the destination fields are foreign keys referring to the id fields of the datanode and querynode tables respectively. The hop field keeps track of the reachability information between pairs of nodes. In order to incorporate σ-Similarity between pairs of nodes, we retain similarity information in *Sim(node1, node2, score)* table. We executed our method with λ=0.50, $\beta \in \{0.80, 0.60\}$, $\mu_1 = 0$ and $\mu_2 = 0$. For σ-similarity we chose to perform exact labeled search i.e. $\sigma(i, j) = 1$ if $i = j$, it is 0 otherwise.

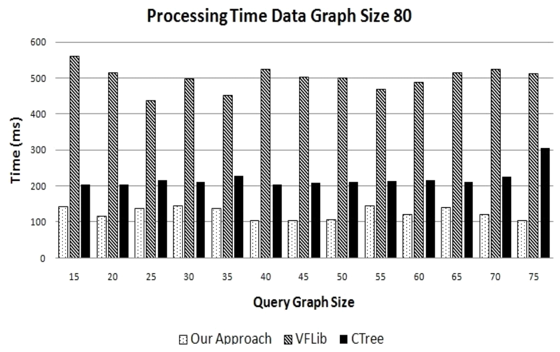
To evaluate the performance of our method, we thoroughly tested the algorithm on 4 standard data sets each having their own properties. The data sets range from artificially generated graphs and their isomorphic subgraphs to real pathways, interaction networks and literature mined document networks. On one hand, the interaction network contains numerous nodes and edges resulting in a large data graph. On the other hand, the average number of nodes and edges in the literature mined network is relatively small. We chose these data sets to portray the efficacy and scalability of our system from different perspectives. While matching between smaller data sets are easy to visualize, huge data sets convey the scalability threshold of the system.



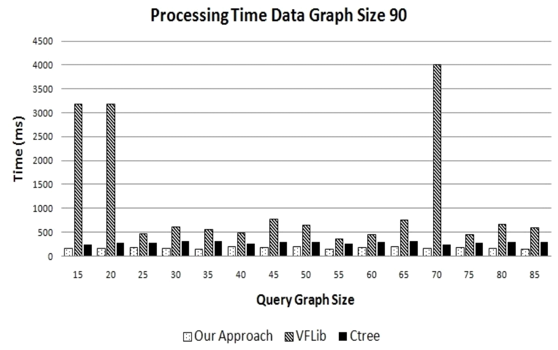
(a) Data graph size 60



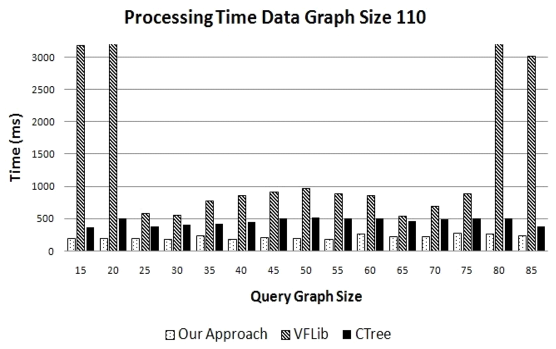
(b) Data graph size 70



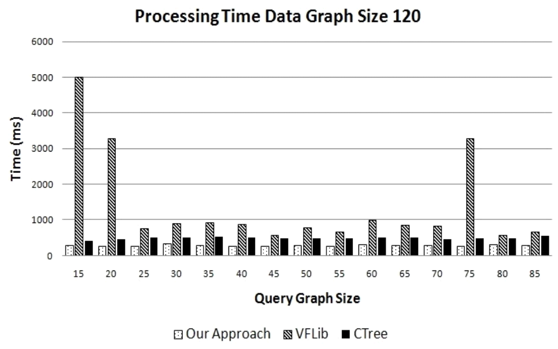
(c) Data graph size 80



(d) Data graph size 90



(e) Data graph size 110



(f) Data graph size 120

Figure 6.1: Performance Evaluation on various data graph sizes.

We have extensively tested the performance of our algorithm on graph benchmark database [37]. The database provides tools to generate several types of graphs and their associated isomorphic subgraphs. We have compared the performance of VFLib [29], closure-tree [45] and the proposed algorithm, TraM on 153 data sets of different sizes. Figure 6.1 depicts the results. We varied the data size from 60 nodes to 120 nodes and subsequently varied the number of nodes in the query graph for each type of data graph. It is evident from the result that the proposed method outperforms VFLib and closure-tree in terms of query time. The pre-processing time entails calculating of reachability information at the database end. The rest of the functions are encompassed in the query time. It can be seen that the performance of the closure-tree and our approach is comparable and the performance of VFLib sometimes oscillates significantly. This is due to the fact that in some of the cases, the number of isomorphic sub-graphs may be too large and since VFLib enumerates all of them, it takes more time. Although VFLib computes exact sub-graph isomorphism whereas we enumerate approximate matches, we decided to include VFLib in our comparison since it serves as a benchmark for query performance.

6.1.1 Precision and Recall

We generated numerous graphs and randomly extracted subgraphs from them [37]. We annotated the super graph as the positive class and all the other randomly generated graphs as negative classes. To introduce approximation, we rewired 1%-2% of the edges from the isomorphic sub graphs. If the result of our graph similarity value attained more than the specified threshold value we predicted the data graph as belonging to the positive class and vice versa. Hence, we calculated the precision and recall measure of the algorithm as shown in figure 6.6(a). Precision and Recall can be defined as follows:

$$Precision = \frac{TP}{TP+FP} \tag{6.1}$$

$$Recall = \frac{TP}{TP+FN} \tag{6.2}$$

where TP, FP and TN represent true positive, false positive and true negative respectively. It can be observed from the figure that we obtained an area under curve of about 83%. This can be explained by the fact that since we are interested in approximate matches, many graphs, artificially identified as belonging to the random class, may have high similarity values as well. We have compared the performance with C-tree 6.6(b) which achieves an area under ROC of 78%.

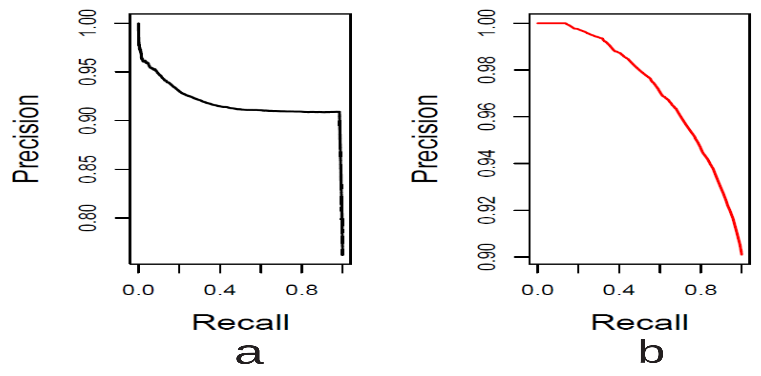


Figure 6.2: Precision-Recall Curves for (a) TraM and (b) C-Tree.

6.1.2 Scalability

To measure the scalability of the approach for large data graphs, we also experimented with PPI network data [113]. We considered high confidence interactions with scores higher than 0.95 or 0.50. There are 23598 interactions among 2345 entities for the first case and 5811 entities and 38064 interactions in the second case. We created query graphs of different sizes from the interaction network and measured processing time as shown in figure 6.3. Multiple query graphs of the same size were generated and average extraction time was calculated for the experiment. For query graph generation, a seed node was selected randomly which was then expanded for possible inclusion of its one hop neighbors. A neighbor is included with a probability value iteratively varying from 0.50 to 0.90. This expansion process is continued until a sufficient number of nodes has been added to the extracted query graph. Subsequently, we randomly deleted 5% and 10% of the edges to create the final query graph.

The difference in the matching time can be explained by two parameters: the radius of the query graph and the extent to which individual nodes in the data graph can be filtered. In cases where most of the nodes can be filtered from the data graph due to their neighbors topology, the processing is much faster. Similarly the radius of the query graph determines the number of nodes in G_{N_i}, which in turn can affect the processing time. Figure 6.3 shows that for the larger data graph (confidence score: 0.5), the matching time is relatively higher than the data graph with confidence score 0.95. However, the matching time in any specific group does not monotonically

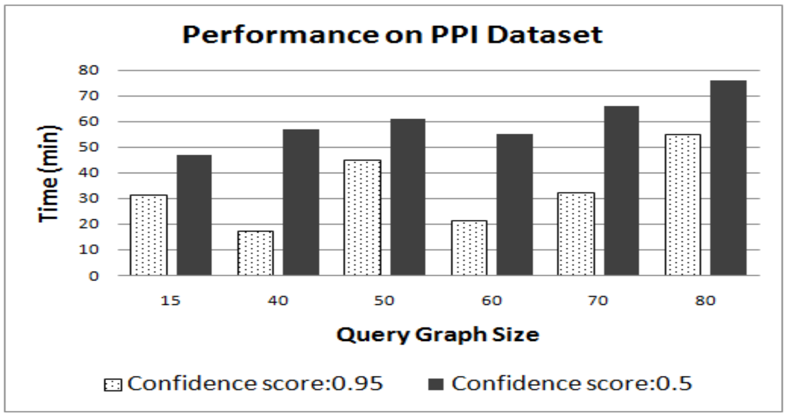


Figure 6.3: Performance Evaluation on PPI Data.

increase with the number of nodes in the query graph. This is due to the fact that the structural and semantic attributes of the query graph play an important role in orchestrating the matching functions.

6.1.3 Effectiveness Analysis

We also explore the efficacy of the method on KEGG [54] and biological literature mined dataset as used by SAGA [92]. SAGA is a flexible matching tool. From the perspective of structural matching the TraM performs functions parallel to SAGA. However, our approach is capable of handling large graphs since it is database driven and performs neighborhood biased computation. Moreover, we have enhanced the flexibility of our approach by leveraging biological information by means of σ-similarity. σ-similarity enables us to include multiple domain information into the calculations to help direct the matching process. The pathway database contains 173 pathways. We iteratively matched each of the pathways to all other pathways in the database and enumerated the time required for the matching process. For SAGA, we chose the default parameters. Figure 6.4 shows the comparative time requirement between our approach and SAGA for the first 100 pathways. With SAGA, we observe spikes in case of a number of query graphs. This is related to the number of matching pathways that SAGA was able to unveil. In our case, the

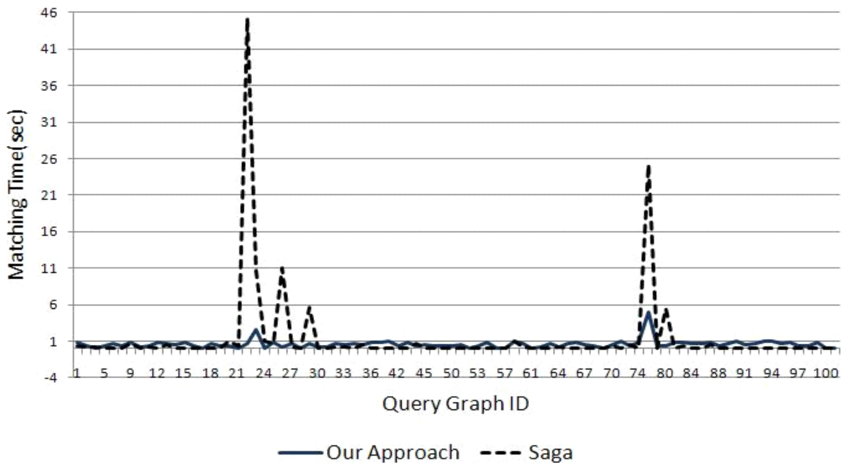


Figure 6.4: Performance Evaluation on KEGG.

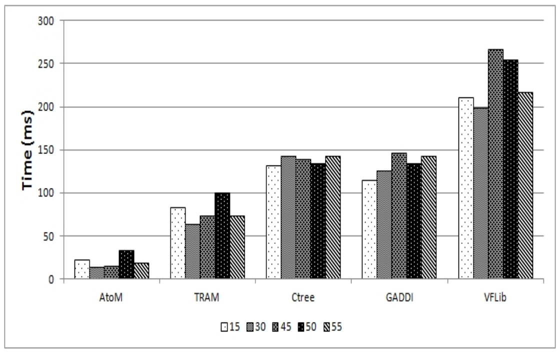
time requirement is independent of the number of matches. As in SAGA, we compared disease-associated pathways with the rest of the pathways in KEGG. We found that *Type II diabetes mellitus - Homo sapiens* pathway (hsa04930) has close association with both the *Insulin signaling pathway* (hsa04910) and the *Adipocytokine signaling pathway*(hsa04920). This association has been identified in SAGA as well. Moreover, *Maturity onset diabetes of the young*(hsa04950) has a high association with the *insulin signaling pathway*. We also identified that the pathways for *insulin signaling*, *GnRH signaling*, *melanogenesis*, and *adipocytokine signaling* possess high association among the group, as might be expected since each of these pathways involves the endocrine system. We also observed the well-established relationship between *endocytosis pathway* (hsa05040) and the *insulin signaling pathway* (e.g., [17]). We found a number of other expected associations, including those among *carbohydrate metabolism pathways* (e.g., hsa00040, hsa00052, hsa00500, and hsa00520), *insulin signaling* (hsa04910) and *glycolysis/gluconeogenesis* (hsa00010) pathways, and *galactose metabolism* (hsa00052) and *sphingolipid metabolism* (hsa00600). Combined these results demonstrate that TraM can be useful for finding biologically meaningful relationship among pathways.

SAGA applied its algorithm on literature mined graph data-set in an information retrieval setting to identify similarities among documents. It extracted graphs from the literature and performed graph matching techniques to identify semantic similarity among the documents. PubMed documents are represented by a graph where nodes indicate genes studied in that document. An edge is drawn between two genes if they appear in the same sentence, a potential indication of association between the genes. As in SAGA, we query the graph representations of documents to identify those that study the same research topics. The data set contains 48,445 documents and on average, there are 5.0 nodes and 18.76 edges per graph. After executing our method on the literature mined data-set, we were able to extract all the relevant documents as identified by SAGA. Since a pair of query and data documents can potentially contain multiple alignments in multiple spatial regions, we can essentially derive a closer match between documents in such cases (see Appendix 7.1 for details).

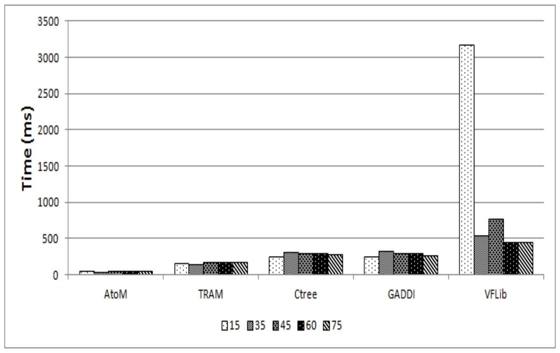
6.2 Experimental Results: AtoM

To evaluate the performance of AtoM, we thoroughly tested the algorithm on standard data sets each having their own properties. The data sets range from artificially generated graphs and their isomorphic subgraphs to real interaction networks.

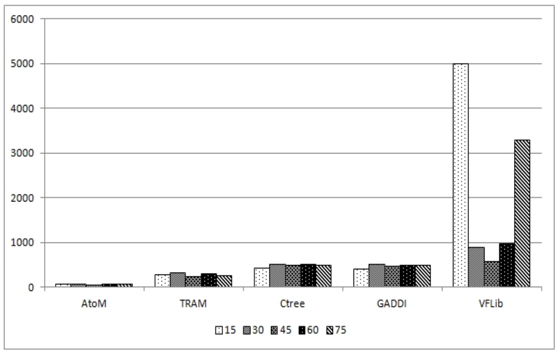
As with TraM, we have extensively tested the performance of our algorithm on graph benchmark database [37]. We used randomly connected graphs, regular meshes and modified mesh graphs from benchmark database of graph isomorphism and sub-graph isomorphism. We also generated different types of graph using Barabasi-Albert and random models (Erdos-Renyi) using a range of parameters. For the Barabasi-Albert model we varied the number of nodes from 50 - 120, and preferential attachment variable from 1 to 4. We have compared the performance of VFLib, GADDI, closure-tree, TraM and AtoM on 153 data sets of different sizes. Figure 6.5 depicts the results. We varied the data size from 60 nodes to 120 nodes and subsequently varied the number of nodes in the query graph for each type of data graph. It is evident from the result that the proposed method outperforms the state of the art in terms of query time. It can be seen that the performance of the closure-tree and gaddi is comparable. However, the time requirement for AtoM is relatively smaller. The performance of VFLib sometimes oscillates significantly.



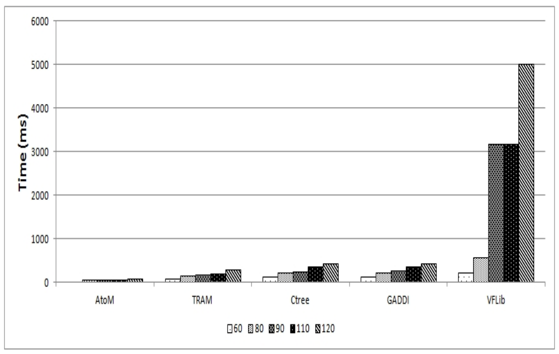
(a) Data graph size 60



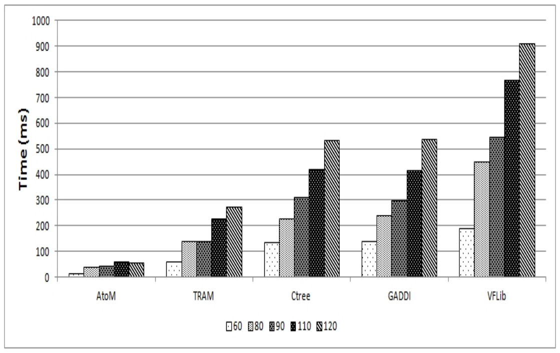
(b) Data graph size 90



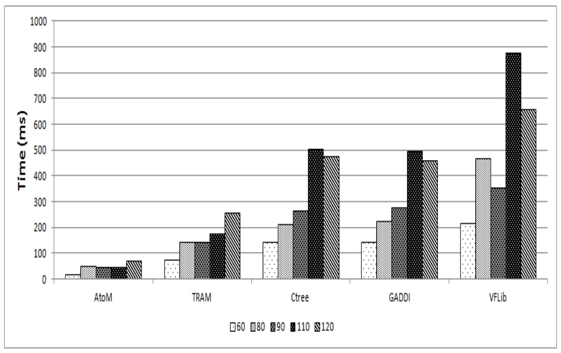
(c) Data graph size 120



(d) Query graph size 15



(e) Query graph size 35



(f) Query graph size 55

Figure 6.5: Performance evaluation on various data and query graph sizes.

This is due to the fact that in some of the cases, the number of isomorphic sub-graphs may be too large and since VFLib enumerates all of them, it takes more time. Results from VFLib has been added in the comparison as a baseline.

6.2.1 Precision and Recall

We generated numerous graphs and randomly extracted subgraphs from them [37]. We annotated the super graph as the positive class and all the other randomly generated graphs as negative classes. To introduce approximation, we rewired 1%-2% of the edges from the isomorphic sub graphs. If the result of our graph similarity value attained more than the specified threshold value we predicted the data graph as belonging to the positive class and vice versa. Hence, we calculated the precision and recall measure of the algorithm. It can be observed from the figure that we obtained an area under curve of about 83.59%. This can be explained by the fact that since we are interested in approximate matches, the notion of true positive, false positive, true negative and false negatives are somewhat fuzzy in the randomly generated graphs.Many graphs, artificially identified as belonging to the random class, may have high similarity values as well. We have compared the performance with TraM 6.6(b) which achieves a similar area under ROC of 83%.

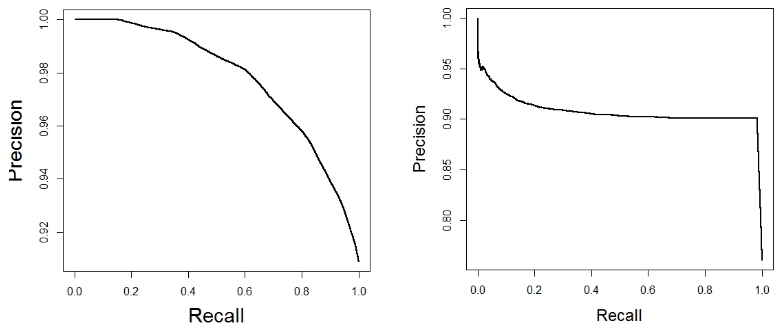


Figure 6.6: Precision-Recall Curves for (a) AtoM and (b) TraM

6.2.2 Scalability

To measure the scalability of the approach for large data graphs, we also experimented with PPI network data [113]. We considered high confidence interactions with scores higher than 0.95 or 0.50. There are 23598 interactions among 2345 entities for the first case and 5811 entities and 38064 interactions in the second case. We created query graphs of different sizes from the interaction network and measured processing time as shown in figure 6.3.

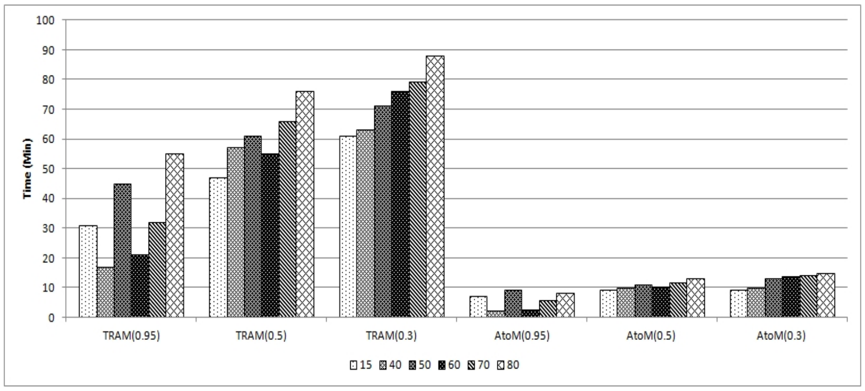


Figure 6.7: Comparing AtoM with TraM on large PPI data set.

Multiple query graphs of the same size were generated and average extraction time was calculated for the experiment. For query graph generation, a seed node was selected randomly which was then expanded for possible inclusion of its one hop neighbors. A neighbor is included with a probability value iteratively varying from 0.50 to 0.90. This expansion process is continued until a sufficient number of nodes has been added to the extracted query graph. Subsequently, we randomly deleted 5% and 10% of the edges to create the final query graph.

The difference in the matching time for TraM be explained by two parameters: the radius of the query graph and the extent to which individual nodes in the data graph can be filtered. In cases where most of the nodes can be filtered from the data graph due to their neighbors topology, the processing is much faster. In figure 6.3, we can see that for the larger data graph (confidence score: 0.5), the matching time is comparatively higher than the data graph with confidence score 0.95.

CHAPTER 7: CONCLUSIONS AND FUTURE WORK

The proposed method is a heuristic algorithm and hence, does not guarantee extraction of all matches. However, the method is scalable for large numbers of nodes and edges since the our first approach delegates a significant amount of processing to the database end and our second approach relies on the domain knowledge to to focus on promising search spaces. In both cases the early filtration helps prune the search space and makes the process highly efficient. In TraM, The matching is done by means of assigning importance values, measured in terms of their topological behavior, to each node in the data graph in their local neighborhood and comparing them with that of the query graph. AtoM harnesses the power of randomized summary graph generation followed by approximate node to node matching which makes the process efficient and scalable for a large number of nodes and edges. The method we propose here is a general tool for approximate subgraph matching queries, and can be easily customized to meet the requirement of different applications. Our empirical evaluations demonstrate the improved effectiveness and efficacy of the proposed method over existing methods.

One thing to note that the definition of node and graph similarity above empowers us to design a matching algorithm that is well suited for traditional single processor machines, as well as a parallel version based on the Map-Reduce framework. In this dissertation, our focus is to demonstrate single processor implementation of our method and their advantages over contemporary algorithms in terms of efficiency and scalability. We, however, present an outline of the Map-Reduce version of AtoM and discuss how it can be implemented. To that end compatibility studies of Map-Reduce based graph analysis languages and engines [2, 3] and ATOM can be explored. As a future direction an extensive study on AtoM's map-reduce compatibility is warranted. Moreover, application of our approaches to discover novel biological insights, as well as in other applications, such as social networks and RDF graph datasets, would be interesting.

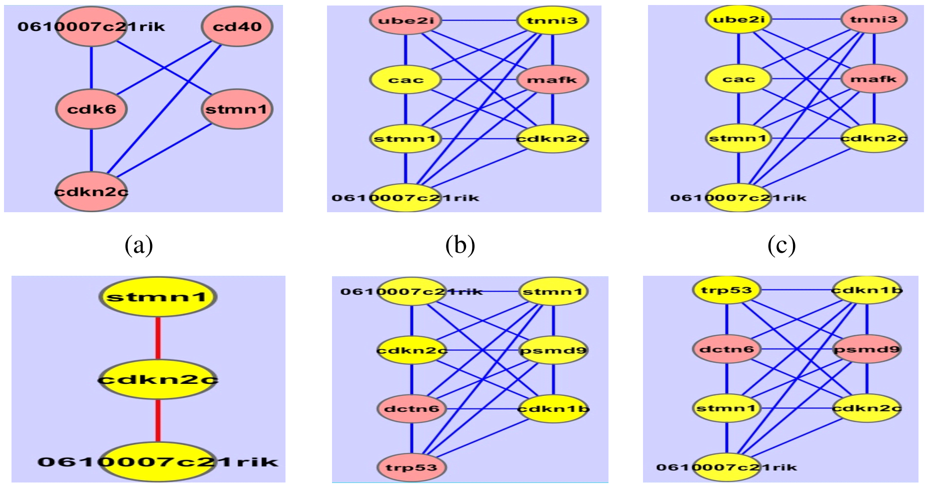
APPENDIX A

7.1 Comparison with SAGA

We thoroughly experimented with the documents mined from literature and compared them with each other. Online SAGA tool *http://saga.ncibi.org/* by default uses the query document *GR12196289* and matches it against other documents in the database. We performed the same query and identified the same set of documents as well. However, since we enumerate a complete alignment of graphs, a query graph and a data graph may contain multiple plausible alignments which may denote a closer match between the documents. For example figures 7.1(e) and 7.1(f) refer to two different alignments with the query graph as shown in figure 7.1(a) for the same data graph. Analogously figures 7.1(b) and 7.1(c) denote alignments for the same document as well. Table 7.1 shows the alignment score as well as the actual alignment of the query document with other documents. Different sets of matching are separated by double lines in this table.

7.2 Result on KEGG Data set

Table 7.2 shows a subset of the pair-wise matching on the KEGG data set used in SAGA. The complete result and additional details may be found at:
`http://mapbase.nkn.uidaho.edu/TRAMTool/Kegg.txt`.



(a) (b) (c)

(d) (e) (f)

Figure 7.1: (a) A snapshot of document matching for SAGA dataset for query document GR12196289 and (b-c)Alignment for document GR9915769, (d) Alignment for document GR12556487 and (e-f) Alignments for document GR15584024.

Table 7.1: Partial Matching result on literature mined data set

Document ID	Score	Matching
GR12556487	1.0	{0610007c21rik,cdkn2c,stmn1}
GR12556487	0.4319750944975411	{cd40,cdkn2c,cdk6}
GR12668976	1.0	{0610007c21rik,cdkn2c,stmn1}
GR12668976	0.4319750944975411	{cd40,cdkn2c,cdk6}
GR15584024	0.889901788709605	{0610007c21rik,cdkn2c,stmn1,cdkn1b,psmd9}
GR15584024	0.8147265498682156	{0610007c21rik,cdkn2c,stmn1,cdkn1b,trp53}
GR15584024	0.7823407854996212	{0610007c21rik,cdkn2c,stmn1,cdkn1b,dctn6}
GR15584024	0.6460079842147224	{cdkn1b,cdkn2c,stmn1,dctn6,trp53}
GR15584024	0.573265635057969	{cdkn1b,cdkn2c,stmn1,dctn6,psmd9}
GR15584024	0.3747124312437049	{cdkn2c,dctn6,stmn1,psmd9,trp53}
GR9915769	0.9364838167686333	{0610007c21rik,cdkn2c,stmn1,cac,tnni3}
GR9915769	0.8001283898817831	{0610007c21rik,cdkn2c,stmn1,cac,ube2i}
GR9915769	0.7749907541995343	{0610007c21rik,cdkn2c,stmn1,cac,mafk}
GR9915769	0.732507085798144	{cac,cdkn2c,stmn1,mafk,ube2i}
GR9915769	0.5640047743565308	{cac,cdkn2c,stmn1,mafk,tnni3}
GR9915769	0.3622860289816281	{cdkn2c,mafk,stmn1,tnni3,ube2i}
GR15591344	1.0	{ccne1,cdkn1b,dctn6,psmd9,tnfsf13b}
GR12062451	0.8717771798337681	{cdkn2a,cdkn2c,stmn1,cdkn2d,sp1}
GR12062451	0.7298091592620806	{cdkn2a,cdkn2c,stmn1,cdkn2d,il23a}
GR12062451	0.4991713844105058	{cdkn2c,cdkn2d,stmn1,il23a,sp1}
GR12941628	0.8192328566939717	{cdkn2a,cdkn2c,stmn1,cdkn2d,ss18l1}
GR12941628	0.7298091592620806	{cdkn2a,cdkn2c,stmn1,cdkn2d,msx1}
GR12941628	0.4991713844105058	{cdkn2c,cdkn2d,stmn1,msx1,ss18l1}
GR12077144	0.8046374044604117	{c2orf28,cdkn2c,stmn1,e2f2,znf197}
GR12077144	0.7165820438224613	{c2orf28,cdkn2c,stmn1,e2f2,sp1}
GR12077144	0.4683905390546311	{cdkn2c,e2f2,stmn1,sp1,znf197}
GR12761212	0.9215258537279282	{c2orf28,cdkn2c,stmn1,ns2,txndc12}
GR12761212	0.838154525673113	{c2orf28,cdkn2c,stmn1,ns2,znf197}
GR12761212	0.7191533589644806	{c2orf28,cdkn2c,stmn1,ns2,txn}
GR12761212	0.598291689050069	{cdkn2c,ns2,stmn1,txn,znf197}
GR12761212	0.4621649695919185	{cdkn2c,ns2,stmn1,txn,txndc12}
GR12761212	0.25	{ns2,stmn1,txn,txndc12,znf197}
GR11906209	0.9523771242651886	{cdk8,cdkn2c,stmn1,cdkn1b,dctn6}
GR11906209	0.9435893852930267	{cdkn1b,cdkn2c,stmn1,cdkn2a,psmd9}
GR11906209	0.9369125905052358	{cdkn2a,cdkn2c,stmn1,cdkn2d,psmd9}
GR11906209	0.8704375490196967	{cdkn1b,cdkn2c,stmn1,cdkn2a,dctn6}
GR11906209	0.7944553089980879	{cdk8,cdkn2c,stmn1,cdkn1b,cdkn2d}
GR11906209	0.7907730215885698	{cdk8,cdkn2c,stmn1,cdkn1b,psmd9}
GR11906209	0.7368563356558973	{cdkn1b,cdkn2c,stmn1,cdkn2a,cdkn2d}
GR11906209	0.7298091592620806	{cdkn2a,cdkn2c,stmn1,cdkn2d,dctn6}
GR11906209	0.7274343299488867	{cdk8,cdkn2c,stmn1,cdkn1b,cdkn2a}
GR11906209	0.4991713844105058	{cdkn2c,cdkn2d,stmn1,dctn6,psmd9}
GR14681224	0.9538143007274169	{c2orf28,cdkn2c,stmn1,cdkn1a,kras}
GR14681224	0.9175969984435695	{cdkn1a,cdkn2c,stmn1,d4s234e,znf197}
GR14681224	0.8737921425432178	{c2orf28,cdkn2c,stmn1,cdkn1a,tceal1}
GR14681224	0.8217136586428023	{cdkn1a,cdkn2c,stmn1,d4s234e,tceal1}
GR14681224	0.7745600144936264	{c2orf28,cdkn2c,stmn1,cdkn1a,znf197}
GR14681224	0.7322960155077387	{cdkn1a,cdkn2c,stmn1,d4s234e,kras}
GR14681224	0.730931928009264	{c2orf28,cdkn2c,stmn1,cdkn1a,d4s234e}
GR14681224	0.6918344198062379	{cdkn2c,d4s234e,stmn1,kras,znf197}
GR14681224	0.6373892844731164	{cdkn2c,d4s234e,stmn1,kras,tceal1}
GR15680327	0.9553030068601917	{c2orf28,cdkn2c,stmn1,mmab,tp53}
GR15680327	0.9349132365710915	{atm,cdkn2c,stmn1,atr,znf197}
GR15680327	0.9299509202310943	{antxr1,cdkn2c,stmn1,atm,mmab}
GR15680327	0.9255845753379568	{atr,cdkn2c,stmn1,c2orf28,znf197}
GR15680327	0.9182359573714657	{antxr1,cdkn2c,stmn1,atm,tp53}
GR15680327	0.9120445603290845	{c2orf28,cdkn2c,stmn1,mmab,znf197}
GR15680327	0.8993954530945191	{atm,cdkn2c,stmn1,atr,mmab}
GR15680327	0.8636123584523138	{atm,cdkn2c,stmn1,atr,serpina2}
GR15680327	0.827006785335414	{atr,cdkn2c,stmn1,c2orf28,tp53}
GR15680327	0.7819189700603164	{atm,cdkn2c,stmn1,atr,tp53}
GR15680327	0.7544552255091185	{antxr1,cdkn2c,stmn1,atm,serpina2}
GR15680327	0.7477414302275683	{atr,cdkn2c,stmn1,c2orf28,serpina2}
GR15680327	0.7353037709373866	{antxr1,cdkn2c,stmn1,atm,c2orf28}
GR15680327	0.7323031041508802	{antxr1,cdkn2c,stmn1,atm,znf197}
GR15680327	0.7307775903814041	{atr,cdkn2c,stmn1,c2orf28,mmab}
GR15680327	0.7292443500465866	{atm,cdkn2c,stmn1,atr,c2orf28}
GR15680327	0.7272378074962538	{antxr1,cdkn2c,stmn1,atm,atr}
GR15680327	0.7252145792197061	{c2orf28,cdkn2c,stmn1,mmab,serpina2}
GR15680327	0.689909706058663	{cdkn2c,mmab,stmn1,serpina2,znf197}
GR15680327	0.46989075455575446	{cdkn2c,mmab,stmn1,serpina2,tp53}
GR15680327	0.4528575362270351	{mmab,serpina2,stmn1,tp53,znf197}
PMID:11777974	0.9658972971856652	{atp8a1,cdkn2c,stmn1,c2orf28,tnfrsf10b}
PMID:11777974	0.7746615028211239	{atp8a1,cdkn2c,stmn1,c2orf28,mafk}
PMID:11777974	0.7441662342917272	{atp8a1,cdkn2c,stmn1,c2orf28}
PMID:11777974	0.7405985131369885	{atp8a1,cdkn2c,stmn1,c2orf28,spp1}
PMID:11777974	0.729675878311312	{atp8a1,cdkn2c,stmn1,c2orf28,ube2i}
PMID:11777974	0.7285369046447443	{atp8a1,cdkn2c,stmn1,c2orf28,csnk2a2}
GR15107822	0.9451409443317211	{cdkn1a,cdkn2c,stmn1,cdkn2a}
GR15107822	0.9354576218615624	{cdkn1a,cdkn2c,stmn1,cdkn2a,hdac6}
GR15107822	0.9232959901621369	{cdkn2a,cdkn2c,cdkn2c,cdkn2d}
GR15107822	0.9192927210145894	{cdkn1a,cdkn2c,stmn1,cdkn2a}
GR15107822	0.9083853122384856	{cdkn2a,cdkn2c,cdkn2c,cdkn2d,hdac4}
GR15107822	0.9069243986530706	{cdkn2a,cdkn2c,cdkn2c,cdkn2d,hdac6}
GR15107822	0.9004803719872181	{cdkn2a,cdkn2c,cdkn2c,cdkn2d}
GR15107822	0.8890767048335564	{cdkn2a,cdkn2c,cdkn2c,cdkn2d,il23a}
GR15107822	0.8588112628495589	{cdkn1a,cdkn2c,stmn1,cdkn2a}
GR15107822	0.8468422418663428	{cdkn2a,cdkn2c,cdkn2c,cdkn2d,hdac3}
GR15107822	0.8031485947892333	{cdkn2a,cdkn2c,cdkn2c,cdkn2d,hdac9}
GR15107822	0.7855876284650885	{cdkn2a,cdkn2c,cdkn2c,cdkn2d,kras}
GR15107822	0.7811788770690845	{cdkn2a,cdkn2c,cdkn2c,cdkn2d}
GR15107822	0.7636481690459593	{cdkn1a,cdkn2c,stmn1,cdkn2a,stmn1}
GR15107822	0.7560479524014393	{cdkn1a,cdkn2c,stmn1,cdkn2a,d4s234e}
GR15107822	0.7447576377423377	{cdkn1a,cdkn2c,stmn1,cdkn2a,hdac4}
GR15107822	0.7263984197973209	{cdkn2c,hdac9,cdkn2d,hdac4}
GR15107822	0.7223085679686758	{cdkn1a,cdkn2c,stmn1,cdkn2a,hdac9}
GR15107822	0.7183405942861486	{cdkn2d,hdac3,stmn1,d4s234e,hdac6}
GR15107822	0.7157859832021588	{d4s234e,hdac3,stmn1,hdac4}
GR15107822	0.7136014888710671	{cdkn1a,cdkn2c,stmn1,cdkn2a,hdac3}
GR15107822	0.7122071328178063	{d4s234e,hdac3,stmn1,hdac4,nsg1}
GR15107822	0.7100602632356102	{cdkn1a,cdkn2c,stmn1,cdkn2a}
GR15107822	0.7078909195134646	{cdkn2a,cdkn2c,cdkn2c,cdkn2d,d4s234e}
GR15107822	0.7047713061423854	{hdac4,hdac6,stmn1,hdac9,tceal1}
GR15107822	0.7040863344220352	{cdkn1a,cdkn2c,stmn1,cdkn2a,cdkn2d}
GR15107822	0.7028672447266058	{cdkn2d,hdac3,stmn1,d4s234e,nsg1}
GR15107822	0.6890221386687291	{cdkn2d,hdac3,stmn1,d4s234e,il23a}
GR15107822	0.6889358032074109	{hdac4,hdac6,stmn1,hdac9,nsg1}
GR15107822	0.6777268893784156	{hdac6,hdac9,stmn1,il23a,slc12a9}
GR15107822	0.66337747808447	{d4s234e,hdac3,stmn1,hdac4,slc12a9}
GR15107822	0.6626649936232969	{d4s234e,hdac3,stmn1,hdac4,il23a}
GR15107822	0.6610150933143135	{cdkn2d,hdac3,stmn1,d4s234e,hdac9}
GR15107822	0.6566398560913453	{hdac3,tceal1,stmn1,hdac4,slc12a9}
GR15107822	0.6459607287841115	{cdkn2c,hdac9,cdkn2d,kras}
GR15107822	0.6448077677051283	{hdac3,tceal1,stmn1,hdac4,nsg1}
GR15107822	0.6416111182674311	{hdac9,il23a,stmn1,kras,tceal1}
GR15107822	0.6397986247213692	{il23a,kras,stmn1,nsg1,tceal1}
GR15107822	0.6333859123886905	{cdkn2d,hdac3,stmn1,d4s234e}
GR15107822	0.6050632033407527	{hdac3,tceal1,stmn1,hdac4,kras}
GR15107822	0.6035260659041303	{hdac3,tceal1,stmn1,hdac4,hdac9}
GR15107822	0.5942298339273153	{hdac6,hdac9,stmn1,il23a,nsg1}
GR15107822	0.5939778646379406	{cdkn2d,hdac3,stmn1,d4s234e}
GR15107822	0.5861999410046075	{cdkn2c,hdac9,cdkn2d,hdac9}
GR15107822	0.5805781629562994	{hdac4,hdac6,stmn1,hdac9,kras}
GR15107822	0.5712028522101719	{hdac9,il23a,stmn1,kras,slc12a9}
GR15107822	0.563117705303187	{cdkn2c,hdac9,cdkn2d,hdac3}
GR15107822	0.5495593409629573	{d4s234e,hdac3,stmn1,hdac4,kras}
GR15107822	0.5444852914830443	{cdkn2c,hdac9,cdkn2d,il23a}
GR15107822	0.5411070731924542	{cdkn2c,hdac9,cdkn2d,hdac6}
GR15107822	0.5400452756889714	{cdkn2d,hdac3,stmn1,d4s234e,kras}
GR15107822	0.537584289864823	{hdac3,tceal1,stmn1,hdac4,il23a}
GR15107822	0.5272412397180306	{cdkn2c,hdac9,cdkn2d,nsg1}
GR15107822	0.5257917394383782	{d4s234e,hdac3,stmn1,hdac4,hdac9}
GR15107822	0.5208269966813158	{cdkn2c,hdac9,cdkn2d}
GR15107822	0.5167184738645229	{hdac6,hdac9,stmn1,il23a,tceal1}
GR15107822	0.5123728977954004	{hdac4,hdac6,stmn1,hdac9,slc12a9}
GR15107822	0.4984921128314069	{hdac4,hdac6,stmn1,hdac9,il23a}
GR15107822	0.4960781635411471	{hdac6,hdac9,stmn1,il23a,kras}
GR15107822	0.49552730352642704	{hdac3,tceal1,stmn1,hdac4,hdac6}
GR15107822	0.49282957356843493	{d4s234e,hdac3,stmn1,hdac4,hdac6}
GR15107822	0.49060692772771886	{cdkn2d,hdac3,stmn1,d4s234e,hdac4}

Table 7.2: Partial matching result on KEGG database

Pathway1	Pathway2	Score
hsa00632	hsa00903	0.958520628926046
hsa00072	hsa00650	0.930198376212001
hsa00950	hsa00350	0.910028345212001
hsa00900	hsa00100	0.897810741538553
hsa00950	hsa00360	0.862087090505005
hsa00632	hsa00903	0.810986322010449
hsa00720	hsa00020	0.720702977208637
hsa00360	hsa00350	0.719724712596848
hsa00500	hsa00052	0.702605131613838
hsa00072	hsa00280	0.679590854336905
hsa00650	hsa00280	0.652030779642093
hsa00910	hsa00251	0.617720161458517
hsa00520	hsa00052	0.609904664972119
hsa00140	hsa00150	0.593958036561787
hsa00040	hsa00500	0.56113704026524
hsa00450	hsa00271	0.557943367134741
hsa00565	hsa00564	0.543503138561839
hsa00360	hsa00340	0.525147318462108
hsa00640	hsa00280	0.499566029209854
hsa00340	hsa00380	0.47514630295471
hsa00520	hsa00500	0.463230117573564
hsa00410	hsa00280	0.461580316269474
hsa00630	hsa00020	0.448229933427644
hsa00240	hsa00230	0.441421741177308
hsa00770	hsa00410	0.437200956742212
hsa04150	hsa04910	0.436125814407344
hsa04912	hsa04540	0.434924965012993
hsa00310	hsa00380	0.430961015977778
hsa04930	hsa04910	0.429746019875289
hsa00770	hsa00240	0.422438144452044
hsa00071	hsa00280	0.421432534807331
hsa00601	hsa00602	0.416535247445683
hsa00360	hsa00380	0.416056548913924
hsa00340	hsa00350	0.398277636433689
hsa00030	hsa00010	0.391414814331596
hsa00603	hsa00604	0.386821550819256
hsa00410	hsa00340	0.380442868258012
hsa04912	hsa04010	0.380268397650829
hsa00330	hsa00220	0.378366167515977
hsa00071	hsa00380	0.376726032641632
hsa00650	hsa00071	0.376259561611575
hsa04916	hsa04912	0.375841477828226
hsa04916	hsa04912	0.375841477828226
hsa00410	hsa00240	0.374267632605089
hsa00071	hsa03320	0.373284463978904
hsa00410	hsa00650	0.36671116861823
hsa00561	hsa00564	0.363567163281109
hsa04916	hsa04310	0.34849947297084
hsa04810	hsa04510	0.336310389392774
hsa04740	hsa04720	0.328239328354333
hsa04740	hsa04916	0.325763031437259
hsa00051	hsa00010	0.323478364458559
hsa00071	hsa00310	0.321602688767554
hsa00052	hsa00010	0.320103936816843
hsa04916	hsa04540	0.318028819459711
hsa00340	hsa00220	0.317250963658232
hsa04740	hsa04912	0.313781569992293
hsa00252	hsa00251	0.310819385859129
hsa00640	hsa00071	0.301876241948114

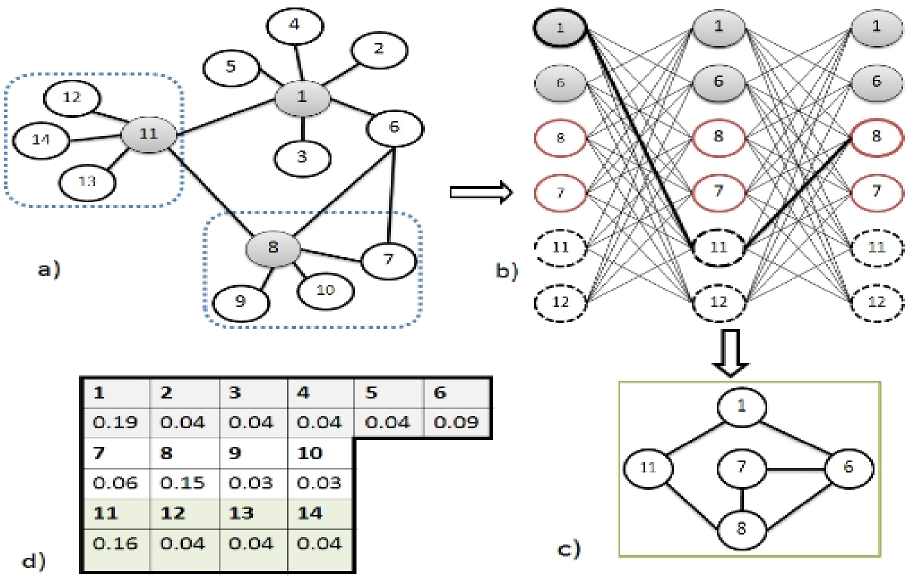
APPENDIX B

7.3 EXTRACTING DISEASE SUB-NETWORK USING SEED DISEASE GENES

Experimental methods are beginning to define the networks of interacting genes and proteins that control most biological processes. There is significant interest in developing computational approaches to identify subnetworks that control specific processes or that may be involved in specific human diseases. Because genes associated with a particular disease (i.e., disease genes) are likely to be well connected within the interaction network, the challenge is to identify the most well-connected subnetworks from a large number of possible subnetworks. One way to do this is to search through chromosomal loci, each of which has many candidate disease genes, to find a subset of genes well connected in the interaction network. In order to identify a significantly connected subnetwork, however, an efficient method of selecting candidate genes from each locus needs to be addressed. In the current study, we describe a method to extract important candidate subnetworks from a set of loci, each containing numerous genes. The method is scalable with the size of the interaction networks. We have conducted simulations with our method and observed promising performance.

7.3.1 Method and Implementation

The inputs to the algorithm are a set of genes ($\mathscr{G}$) to be assessed for a particular disease, loci associated in the disease ($\mathscr{L}$), genes contained in those loci (l_i) and a set of known disease genes ($\mathscr{S}$) involved in the disease. The method we propose in this paper proceeds in three stages. In the first stage we iteratively employ a random walk method to extract from disease loci, genes with the highest potential of being involved in a structurally significant subnetwork (Lines 4-11, Algorithm 10). A random walk is a mathematical formalization of a trajectory that consists of



1	2	3	4	5	6
0.19	0.04	0.04	0.04	0.04	0.09
7	8	9	10		
0.06	0.15	0.03	0.03		
11	12	13	14		
0.16	0.04	0.04	0.04		

Figure 7.2: Execution of the method on a synthetic example: a) A synthetic network where with three loci (color-coded) and three disease genes (highlighted), b) Functional correlation analysis where nodes selected from the same locus are highlighted using same color and highlighted nodes and edges are the highest probable path constructing the subnetwork, c) Final extracted significant subnetwork, d)Random-walk probabilities for the nodes in the network

taking successive random steps through an interaction network. The steps constitute a series of transitions from the walker's current node to a randomly selected neighbor starting at a given set of seed nodes, $\mathscr{S}$. The initial seed nodes are known disease genes for a particular disease (Line 3, Algorithm 10). At the end of each walk the genes from each locus associated with that disease are prioritized based on the probability that the random walker will end up in that particular gene (Lines 9-10, Algorithm 10). In the following iterations, the top priority genes from each locus are used as new seed genes and this process is repeated until the set of seed genes do not change (Lines 5-12, Algorithm 10). We also enumerate the number of times a specific edge is visited ($\mathscr{E}$) during the random walk process. This edge measure delineates the probabilistic edge-visit frequency associated with each edge which is used in the subsequent phases of the method. Once we have extracted the consensus set of candidate genes from each locus, we choose top k (k is a user specified parameter) most visited genes according to the random walk scores ($\mathscr{N}$) and extract the induced subgraph involving these genes from the interaction network (Lines 13-14, Algorithm 10).

Algorithm 10: Algorithm Find-Candidate-SubNet

```
1  $\mathscr{G}$ = set of genes
2  $\mathscr{L} = \{l_1, l_2, l_3, \ldots, l_x\},\ l_i = \{g_{i1}, g_{i2}, g_{i3}, \ldots, g_{im}\}$
3  $\mathscr{S}$ = seed gene set
4  $\mathscr{NR} = \mathscr{S}$
5  while $\mathscr{NR}$ converges do
6      $(\mathscr{N}, \mathscr{E})$ = random-walk($\mathscr{NR}$)
7      $\mathscr{NR} = \mathscr{S}$
8      For each $l_i \in \mathscr{L}$, $l_i \cap \mathscr{S} = \phi$ sort $l_i$ according to descending values of $\mathscr{N}$
9      $\mathscr{NR} = \mathscr{NR} \cup g_{i1}$
10 $\mathscr{CG} = \bigcup_{l_i \in \mathscr{L}} g_{i1}, g_{i2}, \ldots, g_{ik}$
11 $(\mathscr{CG}, \mathscr{CE})$ = Induced-Network($\mathscr{N}, \mathscr{E}, \mathscr{CG}$)
12 $Nets$ = Find-M-Best-Network()
13 $Subnets = \phi$
14 for each net $\in$ $Nets$ do
15     $Subnets = Subnets \cup$ Simple-Net($net$)
16 return $Subnets$
```

Algorithm 11: Algorithm Simple-Net

```
1  Fix a positive relevance threshold $\theta_e$
2  while net is disconnected do
3      $RefinedNet = net$
4      $net$ = Refined $net$ after keeping edges with relative relevance above $\theta_e$
5      Adjust threshold $\theta_e$ to guarantee connectedness
6  return $RefinedNet$
```

In the second stage, we analyze the network by running the algorithm "Find-M-Best-Network" that dynamically calculates the most probable path traversed by the random walk method involving the nodes identified in line 5–12 of Algorithm 10 and returns the top-M most likely biologically significant structures. In the final stage, we extract the subgraph involving all the nodes from each of the top-M structures that may possibly contain other genes that are required to

Algorithm 12: Algorithm Find-*M*-Best-Network

```
1  $a$ = Normalized edge visit probabilities
2  $b$ = Normalized node visit probabilities
3  $N$ = Number of genes, $T$ = Number of loci
4  $p$ = 1 for seed genes, 0 otherwise
5  **Initialization:**
6  $M_{net} = \phi$
7  **for** $i = 1 : N$ **do**
8      $\delta_1(i) = p_i \times b_i$
9      $\Phi_1(i) = 0$
10     $S_1(i) = \phi$
11 **Recursion:**
12 **for** $t = 2 : T$ **do**
13     **for** $j = 1 : N$ **do**
14         $\delta_t(j) = \max_{i \leq N}[\delta_{t-1}(i)a_{ij}] \times b_j | j \notin S_{t-1}(i) \cup \{i\}$
15         $\phi_t(j) = \text{argmax}_i[\delta_{t-1}(i)a_{ij}]$
16         $S_t(j) = \{i\} \cup S_{t-1}(i)$
17 **Termination:** $P^* = M\text{-}\max_i[\delta_T(i)]$
18 $i_T^* = M\text{-argmax}_i[\delta_T(i)]$
19 **Reconstruction of the optimal path:**
20 **for** *each* $j = 1 : |P^*|$ **do**
21     **for** $t = T : -1 : 2$ **do**
22         $i_{t-1}^* = \Phi_t(i_t^*(j))$
23     $M_{net} = M_{net} \cup$ network induced by the path
24 **return** $M_{net}$;
```

ensure connectedness of the induced subgraph by using the Algorithm 11. The algorithm "Find-*M*-Best-Network" works in the following way. Once we have computed the edge visit probability and the random walk centrality measure associated with each network node, we can normalize the node measures for the set of top-k nodes from each locus from the previous stage, so that they sum up to 1. We also compute the transition probability matrix a_{ij} that denotes the probability of making a transition from $node_i$ to $node_j$ via zero of more intermediate nodes inferred from the edge-visit frequency measures computed from the random-walk procedure. The following equations summarize the method:

$$\delta_t(j) = max_{1 \le i \le N}[\delta_{t-1}(i)a_{ij}] \times b_j | j \notin S_{t-1}(i) \cup \{i\}$$
$$\phi_t(j) = argmax_i[\delta_{t-1}(i)a_{ij}]$$
$$S_t(j) = \{i\} \cup S_{t-1}(i)$$

Here we define $\delta_t(j)$ as the probability of choosing a path ending at $node_j$ in locus t where none of the predecessor nodes in the path visited any nodes in the set S. Thus we dynamically compute this value for all the nodes in the network as well as the path they traversed and rank the highest M number of paths for subsequent processing. The nodes pertaining to the most likely visited path may not be sufficient to identify a connected sub-graph from the input interaction network. Hence at the final stage of our algorithm we employ the "Simple-Net" algorithm for each of the M most likely paths to obtain a final substructure that is subsequently tested for significance. To extract a simple subgraph that best captures the relationships between a set of given seed nodes in a large complex graph, we follow the approach described in [32]. In this approach, edge relevances (calculated by Algorithm 10) are treated as proportional to the expected edge passage times while randomly walking from one node of interest to the other(s). This approach is precisely based on the expected number of times a given node or a given edge is used along any random walk connecting the nodes of interest as described in Section 7.3.1. This relative frequency is interpreted as its relevance to explain the relationships between the nodes of interest. Discarding any edge the relevance of which falls below a threshold purges the input graph to obtain the subgraph of interest through a filtration process. A more stringent threshold would result in a smaller subgraph.

Figure 7.2 depicts the three stages of the algorithm "Find-Candidate-SubNet" with a synthetic example. Assume that we have a scale-free interaction network (Figure 7.2-a, $\lambda = 0.901$ i.e. the power coefficient in the power law) with three genomic loci. Locus 1, 2 and 3 contain genes 1-6, 7-10, and 11-14 respectively. Let us assume that genes 1, 11 and 8 are disease genes among which only the disease association of gene 11 is known. So, we expect to find the module formed

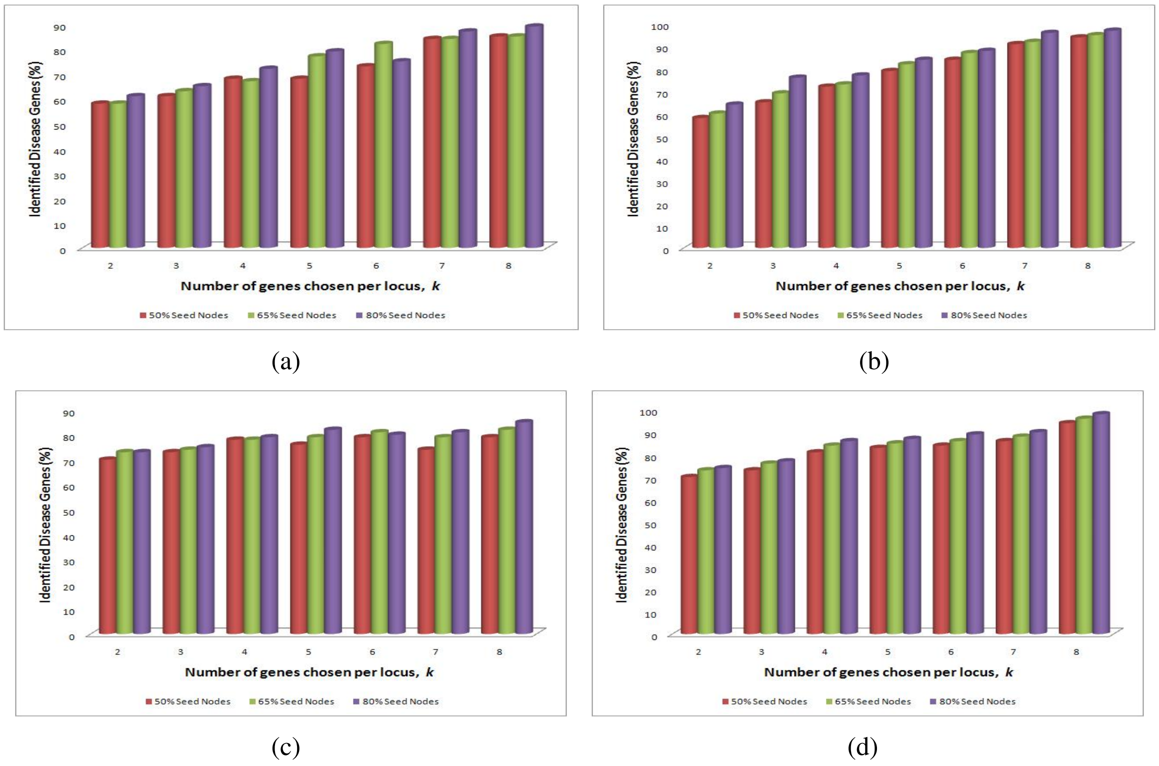


Figure 7.3: (a) Before applying the “Simple-Net” algorithm ($M = 5$), (b) After applying the “Simple-Net” algorithm ($M = 5$) (c) Before applying the “Simple-Net” algorithm ($M = 10$), (d) After applying the “Simple-Net” algorithm ($M = 10$)

by these disease genes after the application of the algorithm. After running the first iteration of the algorithm, based on the corresponding random-walk scores we select the top-ranked gene from every loci for which no seed genes are given. This ensures the method’s biasness towards selection of genes from every possible locus. After running the first stage of the algorithm, we obtain the numeric ranks of the genes, with respect to the seed genes, which are shown adjacent to the network in Figure 7.2-a. Subsequently, we extract the top k genes (k=2 for this example) from each locus and create the network depicted in Figure 7.2-b. It is noticeable that we obtained genes 1, 6, 8, 7, 11, 12 as the top genes from each locus. We run algorithm 12 on the induced subgraph to extract the most likely path as shown with a dotted line in the figure consisting of nodes 1, 11 and 8. When we run Algorithm 11 on the extracted path we obtain the most likely

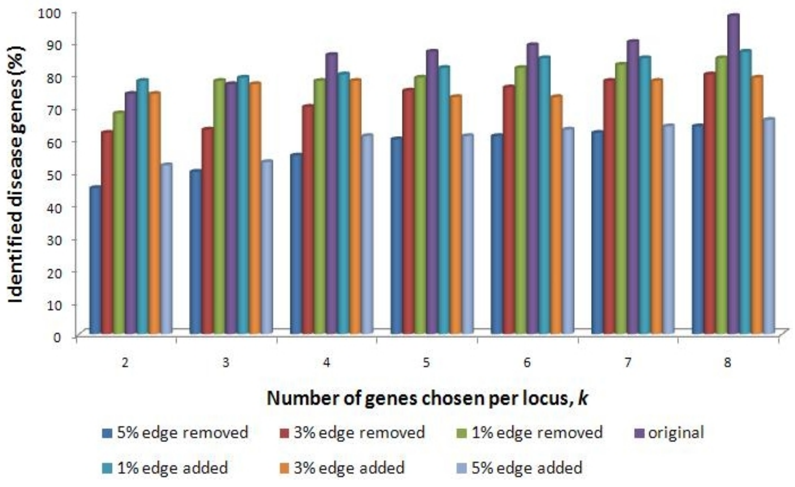


Figure 7.4: Robustness of the algorithm

structure as shown in Figure 7.2-c. Here the nodes 7 and 6 are included in the structure to ensure connectivity and inclusion of other equally significant edges in the resulting structure.

7.3.2 Evaluation

In order to validate our method, we first selected a number of disease genes pertaining to a specific disease from the OMIM Morbid map [69]. For each of the disease genes, similar to [57], we created an artificial locus by defining a region containing fifty genes nearest to the disease gene according to genomic distance measure. We obtain the protein-protein interaction network from NCBI Entrez Gene FTP site [1]. Hence we varied the percentage of known disease genes considering them as seed nodes and applied our approach to extract the other disease genes in the extracted subgraphs.

We assessed the efficacy of the approach on two different sets of nodes i.e. the path nodes identified from the "Find-M-Best-Network" function that returns the network consisting only of the nodes participating in top-k loci gene set and then on the network nodes returned from the "Simple-Net" function. Figure 7.3 depicts the performance of our approach. For a fixed value of $M = 5$, in the x axis we varied the value of k ranging from 2 to 8 and in the y axis we projected the percentage of the rest of the disease genes that we were able to extract through our experiments.

From Figure 7.3, it is clear that, the accuracy of the algorithm is significantly high ($\geq 60\%$ for all the cases). Figure 7.3(a) illustrates that though the accuracy is high, it is not monotonically increasing with the increasing value of k. This is due to the fact that, the resulting nodes may not enforce the extracted subgraph to be connected. Figure 7.3(b) demonstrates that, if we use the "Simple-Net" algorithm further, it enforces the extracted subgraph to be connected with many of the bottleneck candidate nodes resulting a monotonically increasing accuracy. Moreover, it is noticeable that, the accuracy of the algorithm increases with the increase in the value of M. From Figure 7.3(c) and Figure 7.3(d) we see that, as the number of prioritized genes from each locus (denoted as k) that is incorporated in the ensuing network calculation is incremented, the performance of the method improves as expected. Moreover, it is trivial to find better performance for higher percentage of seed nodes as the seed node set guides the extraction process for the rest of the candidate genes. Hence, the performance improves significantly when the "Simple-Net" algorithm is applied since the "Simple-Net" algorithm extends the "Find-M-Best-Network" set of genes to maintain connectivity of the subgraph. Thus, interaction proteins of the initial set of genes and bottleneck proteins also get included in the network.

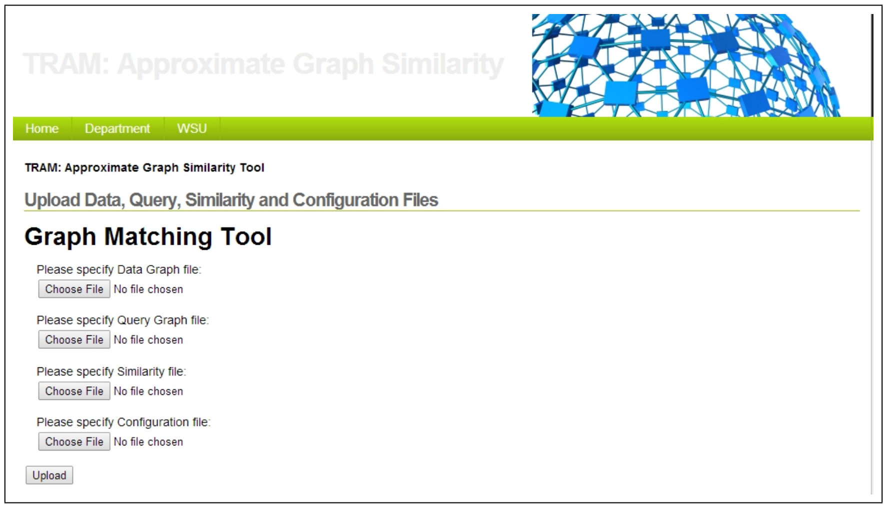
In order to assess the robustness of our algorithm, we randomly add and delete a very small fraction of the edges of the input network and execute the algorithm. The corresponding graph is shown in Figure 7.4. This validation is required since the actual protein protein interaction network is noisy and incomplete, and so we need to demonstrate that our algorithm does not break down due to the presence of noisy data in the form of erroneous edges. In the graph in Figure 7.4, it can be observed that when we randomly modify the network by adding and deleting 1%, 3% and 5% of the edges, the performance of the algorithm still remains stable.

In this section we demonstrated a stochastic approach to identify relevant subgraphs from an interaction network in order to prioritize candidate disease genes. Our experimental results reveal that we are able to identify most of the relevant subgraphs in an interaction network. The stochastic nature of our algorithm guarantees significantly improved turn-around time compared to the brute-force approach. Thus the approach is likely to be scalable to any biological network data. Moreover, further study can be conducted on pathway prediction and validation that may aid in the understanding of underlying biological functionality of genetic networks.

APPENDIX C

7.4 TRAM as a graph matching toolkit

The toolkit is written in Java and MySQL. The web interface is implemented using JSP servlets. For graph visualization dracula graph drawing tool (http://www.graphdracula.net/) is used. The TRAM toolkit is a web based tool that can take as input the data graph, query graph, similarity matrix and a configuration file to tweak the matching algorithm and returns the top-*k* matched sub-graphs. Figure 7.5 depicts the main page of the toolkit consisting of provisions to upload the data, query, similarity matrix and the configuration files. The data graph and the query graph files


TRAM: Approximate Graph Similarity

TRAM: Approximate Graph Similarity Tool

Upload Data, Query, Similarity and Configuration Files

Graph Matching Tool

Please specify Data Graph file:
No file chosen

Please specify Query Graph file:
No file chosen

Please specify Similarity file:
No file chosen

Please specify Configuration file:
No file chosen

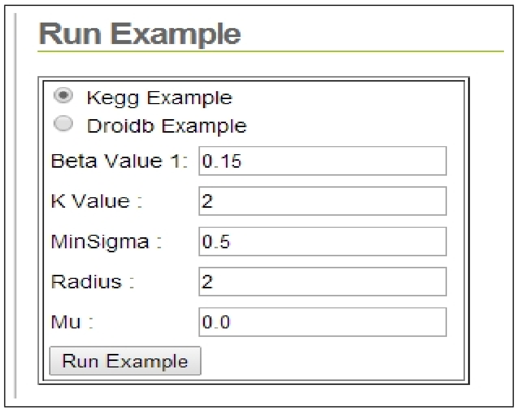
Figure 7.5: Main page of TRAM Tool

have the same structure; each file starts with an integer indicating the number of nodes. Each following line is a pair of string indicating an edge belonging to the graph. The similarity values are stated by similarity files, where each line is consists of three values separated by white space: the data graph node label, the query graph node label and the similarity value. The configuration file is an xml file having the following structure:

```
<?xml version="1.0"?>
```

```
<config>
<attr><name>K</name><value>4</value></attr>
<attr><name>beta</name><value>0.15</value></attr>
<attr><name>beta</name><value>0.55</value></attr>
<attr><name>mu</name><value>0.1</value></attr>
<attr><name>radius</name><value>2</value></attr>
<attr><name>minsigma</name><value>0.5</value></attr>
</config>
```

In the structure above all the necessary parameters including the list β values to use to compute β-signature can be added in the input. To depict an example of how the toolkit works, we have included examples of real life data sets in the toolkit. The toolkit has embedded example on Kegg and Droidb databases. Figure 7.6 shows the section that can spawn the execution of the example section. Figures 7.8, 7.9, 7.10 depict the data graph from Driodb, and example query graph and a matched graph respectively. Figure 7.7 shows the page that lists the top-*k* results.

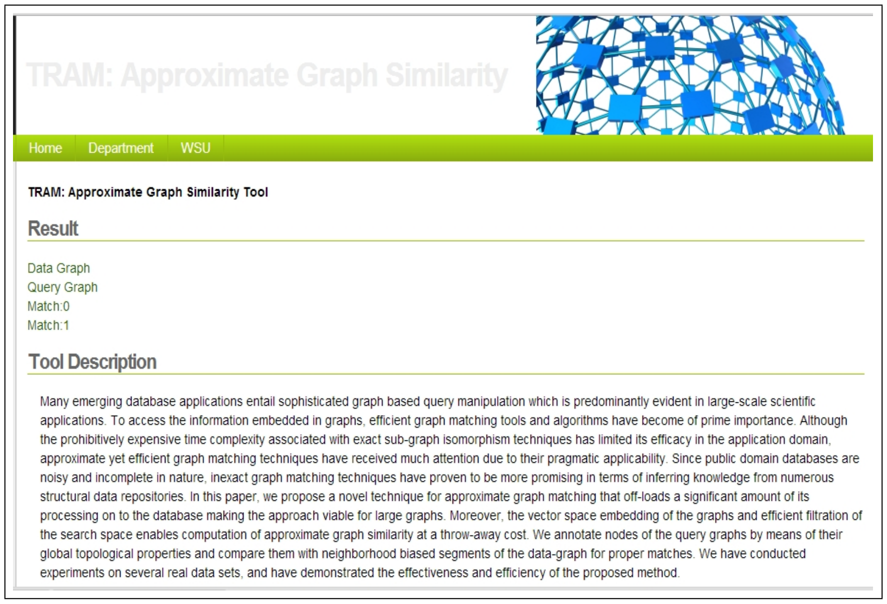

Run Example

[✓ Kegg Example]
Droidb Example

Beta Value 1: 0.15
K Value : 2
MinSigma : 0.5
Radius : 2
Mu : 0.0

Figure 7.6: Example section of the tool

This example section has preselected default parameter values and the input graph can be chosen from either droid or kegg database. We have also included a predefined query graph for the example purpose. The result page lists the top=*k* matches. Clicking on any of the links will open the matched graph for further processing The stand-alone TraM library is also available to



TRAM: Approximate Graph Similarity Tool

Result

Data Graph
Query Graph
Match:0
Match:1

Tool Description

Many emerging database applications entail sophisticated graph based query manipulation which is predominantly evident in large-scale scientific applications. To access the information embedded in graphs, efficient graph matching tools and algorithms have become of prime importance. Although the prohibitively expensive time complexity associated with exact sub-graph isomorphism techniques has limited its efficacy in the application domain, approximate yet efficient graph matching techniques have received much attention due to their pragmatic applicability. Since public domain databases are noisy and incomplete in nature, inexact graph matching techniques have proven to be more promising in terms of inferring knowledge from numerous structural data repositories. In this paper, we propose a novel technique for approximate graph matching that off-loads a significant amount of its processing on to the database making the approach viable for large graphs. Moreover, the vector space embedding of the graphs and efficient filtration of the search space enables computation of approximate graph similarity at a throw-away cost. We annotate nodes of the query graphs by means of their global topological properties and compare them with neighborhood biased segments of the data-graph for proper matches. We have conducted experiments on several real data sets, and have demonstrated the effectiveness and efficiency of the proposed method.

Figure 7.7: Result page of the tool

download and use as part of a larger project. The web toolkit and source code is available for download at: `http://dblab.nkn.uidaho.edu/TraM/`, and
`https://github.com/shafkatdu9212/TraMCode`

Figure 7.11 shows a sub-set of classes from the class diagram for the TraM top-*k* tool.

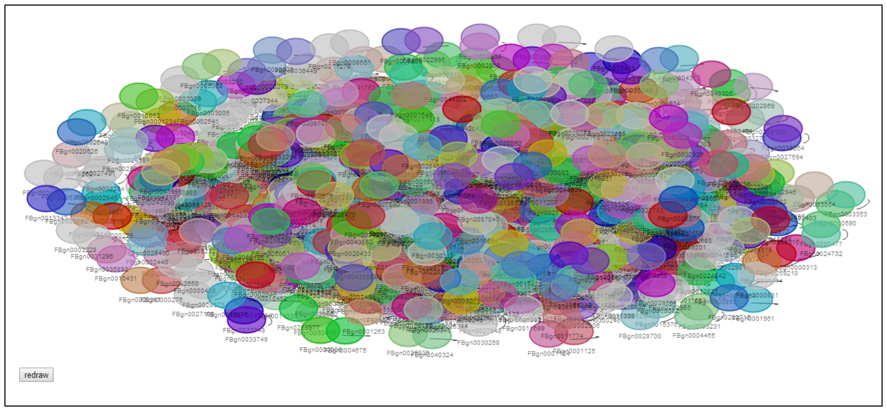


Figure 7.8: Data graph: DroiDB

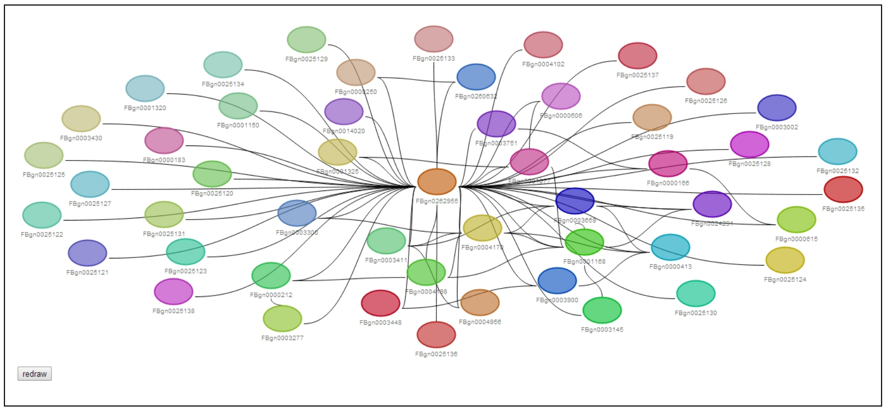


Figure 7.9: Query Graph

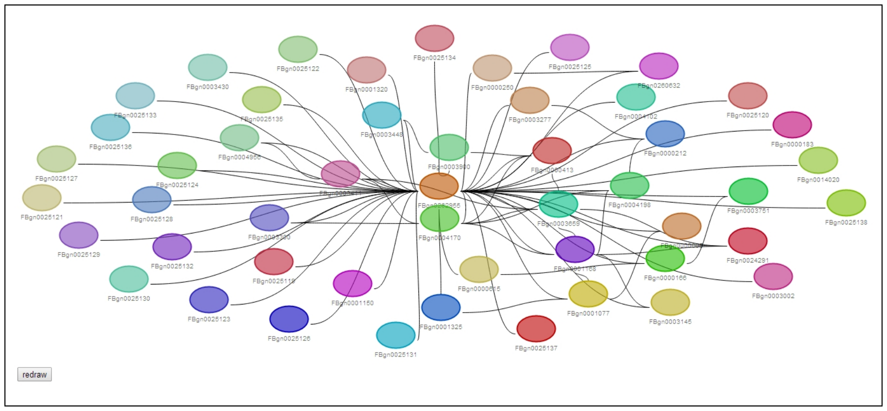


Figure 7.10: Matched Result

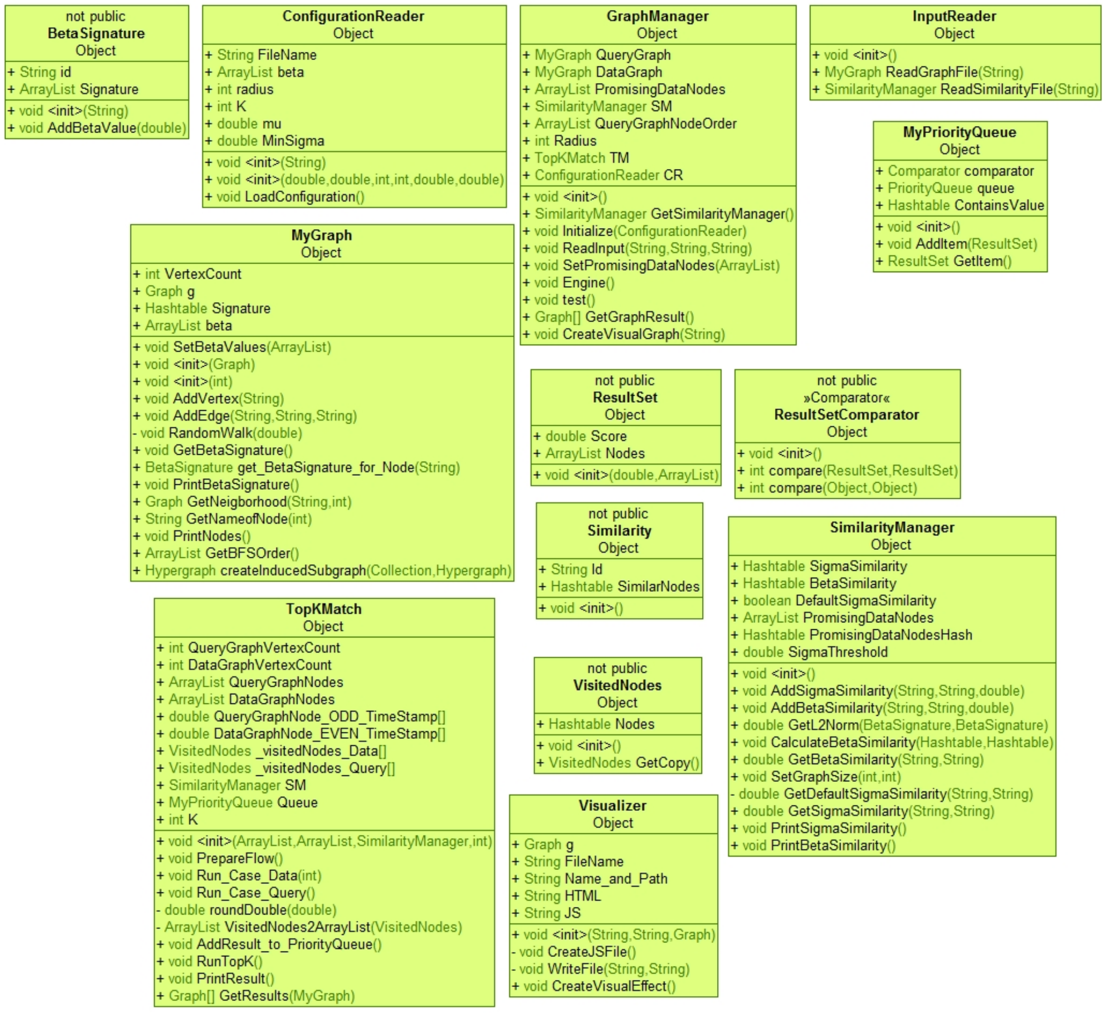

not public
BetaSignature
Object
+ String id
+ ArrayList Signature
+ void <init>(String)
+ void AddBetaValue(double)
ConfigurationReader
Object
+ String FileName
+ ArrayList beta
+ int radius
+ int K
+ double mu
+ double MinSigma
+ void <init>(String)
+ void <init>(double,double,int,int,double,double)
+ void LoadConfiguration()
GraphManager
Object
+ MyGraph QueryGraph
+ MyGraph DataGraph
+ ArrayList PromisingDataNodes
+ SimilarityManager SM
+ ArrayList QueryGraphNodeOrder
+ int Radius
+ TopKMatch TM
+ ConfigurationReader CR
+ void <init>()
+ SimilarityManager GetSimilarityManager()
+ void Initialize(ConfigurationReader)
+ void ReadInput(String,String,String)
+ void SetPromisingDataNodes(ArrayList)
+ void Engine()
+ void test()
+ Graph[] GetGraphResult()
+ void CreateVisualGraph(String)
InputReader
Object
+ void <init>()
+ MyGraph ReadGraphFile(String)
+ SimilarityManager ReadSimilarityFile(String)
MyPriorityQueue
Object
+ Comparator comparator
+ PriorityQueue queue
+ Hashtable ContainsValue
+ void <init>()
+ void AddItem(ResultSet)
+ ResultSet GetItem()
MyGraph
Object
+ int VertexCount
+ Graph g
+ Hashtable Signature
+ ArrayList beta
+ void SetBetaValues(ArrayList)
+ void <init>(Graph)
+ void <init>(int)
+ void AddVertex(String)
+ void AddEdge(String,String,String)
- void RandomWalk(double)
+ void GetBetaSignature()
+ BetaSignature get_BetaSignature_for_Node(String)
+ void PrintBetaSignature()
+ Graph GetNeigborhood(String,int)
+ String GetNameofNode(int)
+ void PrintNodes()
+ ArrayList GetBFSOrder()
+ Hypergraph createInducedSubgraph(Collection,Hypergraph)
not public
ResultSet
Object
+ double Score
+ ArrayList Nodes
+ void <init>(double,ArrayList)
not public
»Comparator«
ResultSetComparator
Object
+ void <init>()
+ int compare(ResultSet,ResultSet)
+ int compare(Object,Object)
not public
Similarity
Object
+ String Id
+ Hashtable SimilarNodes
+ void <init>()
SimilarityManager
Object
+ Hashtable SigmaSimilarity
+ Hashtable BetaSimilarity
+ boolean DefaultSigmaSimilarity
+ ArrayList PromisingDataNodes
+ Hashtable PromisingDataNodesHash
+ double SigmaThreshold
+ void <init>()
+ void AddSigmaSimilarity(String,String,double)
+ void AddBetaSimilarity(String,String,double)
+ double GetL2Norm(BetaSignature,BetaSignature)
+ void CalculateBetaSimilarity(Hashtable,Hashtable)
+ double GetBetaSimilarity(String,String)
+ void SetGraphSize(int,int)
- double GetDefaultSigmaSimilarity(String,String)
+ double GetSigmaSimilarity(String,String)
+ void PrintSigmaSimilarity()
+ void PrintBetaSimilarity()
TopKMatch
Object
+ int QueryGraphVertexCount
+ int DataGraphVertexCount
+ ArrayList QueryGraphNodes
+ ArrayList DataGraphNodes
+ double QueryGraphNode_ODD_TimeStamp[]
+ double DataGraphNode_EVEN_TimeStamp[]
+ VisitedNodes _visitedNodes_Data[]
+ VisitedNodes _visitedNodes_Query[]
+ SimilarityManager SM
+ MyPriorityQueue Queue
+ int K
+ void <init>(ArrayList,ArrayList,SimilarityManager,int)
+ void PrepareFlow()
+ void Run_Case_Data(int)
+ void Run_Case_Query()
- double roundDouble(double)
- ArrayList VisitedNodes2ArrayList(VisitedNodes)
+ void AddResult_to_PriorityQueue()
+ void RunTopK()
+ void PrintResult()
+ Graph[] GetResults(MyGraph)
not public
VisitedNodes
Object
+ Hashtable Nodes
+ void <init>()
+ VisitedNodes GetCopy()
Visualizer
Object
+ Graph g
+ String FileName
+ String Name_and_Path
+ String HTML
+ String JS
+ void <init>(String,String,Graph)
- void CreateJSFile()
- void WriteFile(String,String)
+ void CreateVisualEffect()

Figure 7.11: Class diagram for TraM

REFERENCES

[1] Entrez gene. http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?db=gene.

[2] Faunus:graph analytics engine. http://thinkaurelius.github.io/faunus/.

[3] Titan: Distributed graph database. http://thinkaurelius.github.io/titan/.

[4] Rakesh Agrawal and H. V. Jagadish. Direct algorithms for computing the transitive closure of database relations. In *Proc. 13th Int'l Conf. on Very Large Data Bases*, pages 255–266. Morgan Kaufmann, 1987.

[5] Boanerges Aleman-Meza, Christian Halaschek-Wiener, Satya Sanket Sahoo, Amit P. Sheth, and Ismailcem Budak Arpinar. Template based semantic similarity for security applications. In *ISI*, volume 3495 of *LNCS*, pages 621–622. Springer, 2005.

[6] Mohammad Shafkat Amin, Anupam Bhattacharjee, and Hasan M. Jamil. A cytoscape based framework for efficient sub-graph isomorphic protein-protein interaction motif lookup. In *SAC*, pages 1572–1576, 2010.

[7] Mohammad Shafkat Amin, Anupam Bhattacharjee, Russell L. Finley Jr., and Hasan M. Jamil. A stochastic approach to candidate disease gene subnetwork extraction. In *SAC*, pages 1534–1538, 2010.

[8] Mohammad Shafkat Amin and Hasan M. Jamil. Atom: An efficient and scalable graph similarity tool for top-k approximate graph matches. In *Technical Report, Wayne State University*. 2014.

[9] Mohammad Shafkat Amin and Hasan M. Jamil. Tram: A web based graph similarity tool for top-k graph matches. In *Technical Report, Wayne State University*. 2014.

[10] Mohammad Shafkat Amin, Russell L. Finley Jr., and Hasan M. Jamil. Top-k similar graph matching using tram in biological networks. *IEEE/ACM Trans. Comput. Biology Bioinform.*, 9(6):1790–1804, 2012.

[11] Mohammad Shafkat Amin, Russell L. Finley Jr., and Hasan M. Jamil. Randomized summary and match validation based efficient similarity querying in large graph databases. In *Technical Report, Wayne State University*. 2014.

[12] GD Bader and CW Hogue. An automated method for finding molecular complexes in large protein interaction networks. *BMC Bioinformatics*, 4(2), 2003.

[13] Pamela Bhattacharya, Marios Iliofotou, Iulian Neamtiu, and Michalis Faloutsos. Graph-based analysis and prediction for software evolution. In *ICSE*, pages 419–429, 2012.

[14] Paolo Boldi, Massimo Santini, and Sebastiano Vigna. A deeper investigation of pagerank as a function of the damping factor. In *Web Information Retrieval and Linear Algebra Algorithms*, 2007.

[15] Paolo Boldi, Massimo Santini, and Sebastiano Vignadsi. Pagerank as a function of the damping factor. In *Fourteenth International World Wide Web Conference*, pages 557–566. ACM Press, 2005.

[16] Bobby-Joe Breitkreutz, Chris Stark, Teresa Reguly, Lorrie Boucher, Ashton Breitkreutz, Michael Livstone, Rose Oughtred, Daniel H. Lackner, Jürg Bähler, Valerie Wood, Kara Dolinski, and Mike Tyers. The BioGRID Interaction Database: 2008 update. *Nucleic Acids Research*, 36:D637–D640, January 2008.

[17] Cecilia Brnnmark, Robert Palmr, S. Torkel Glad, Gunnar Cedersund, and Peter Strlfors. Mass and Information Feedbacks through Receptor Endocytosis Govern Insulin Signaling as Revealed Using a Parameter-free Modeling Framework. *Journal of Biological Chemistry*, 285(26):20171–20179, 2010.

[18] Kevin Brown and Igor Jurisica. Unequal evolutionary conservation of human protein interactions in interologous networks. *Genome Biology*, 8(5):R95+, May 2007.

[19] Christine Brun, Francois Chevenet, David Martin, Jerome Wojcik, Alain Guenoche, and Bernard Jacq. Functional classification of proteins for the prediction of cellular function from a protein-protein interaction network. *Genome Biology*, 5(1), 2003.

[20] Dongbo Bu, Yi Zhao, Lun Cai, Hong Xue, Xiaopeng Zhu, Hongchao Lu, Jingfen Zhang, Shiwei Sun, Lunjiang Ling, Nan Zhang, Guojie Li, and Runsheng Chen. Topological structure analysis of the protein-protein interaction network in budding yeast. *Nucl. Acids Res.*, 31(9):2443–2450, May 2003.

[21] Tolga Can, Orhan Çamoğlu, and Ambuj K. Singh. Analysis of protein-protein interaction networks using random walks. In *BIOKDD '05: Proceedings of the 5th international workshop on Bioinformatics*, pages 61–68, New York, NY, USA, 2005. ACM.

[22] Deepayan Chakrabarti and Christos Faloutsos. Graph mining: Laws, generators, and algorithms. *ACM Computing Surveys*, 38(1), June 2006.

[23] Deepayan Chakrabarti, Christos Faloutsos, and Yiping Zhan. Visualization of large networks with min-cut plots, a-plots and r-mat. *Int. J. Hum.-Comput. Stud.*, 65(5):434–445, May 2007.

[24] Gang Chen, Jianxin Wang, Yi Pan, and Jianer Chen. Identification of breast cancer gene signature in protein interaction network using graph centrality. In *BIBM*, pages 402–405, 2011.

[25] Jing Chen, Bruce Aronow, and Anil Jegga. Disease candidate gene identification and prioritization using protein interaction networks. *BMC Bioinformatics*, 10(1):73, 2009.

[26] Thayne Coffman, Seth Greenblatt, and Sherry Marcus. Graph-based technologies for intelligence analysis. *Communications of the ACM*, 47(3):45–47, March 2004.

[27] Donatello Conte, Pasquale Foggia, Carlo Sansone, and Mario Vento. Thirty years of graph matching in pattern recognition. *IJPRAI*, 18(3):265–298, 2004.

[28] Diane J. Cook and Lawrence B. Holder. Substructure discovery using minimum description length and background knowledge. *Journal of Artificial Intelligence Research*, 1:231–255, 1994.

[29] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento. A (sub)graph isomorphism algorithm for matching large graphs. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 26(10):1367–1372, 2004.

[30] Michael E. Cusick, Niels Klitgord, Marc Vidal, and David E. Hill. Interactome: gateway into systems biology. *Human molecular genetics*, 14 Spec No. 2, October 2005.

[31] Jishnu Das, Jaaved Mohammed, and Haiyuan Yu. Genome-scale analysis of interaction dynamics reveals organization of biological networks. *Bioinformatics*, 28(14):1873–1878, 2012.

[32] Pierre Dupont, Jerome Callut, Gregoire Dooms, Jean-Noel Monette, and Yves Deville. Relevant subgraph extraction from random walks in a graph, (30): 2006-2007. Research Report.

[33] Michael Dürr, Valentin Protschky, and Claudia Linnhoff-Popien. Modeling social network interaction graphs. In *ASONAM*, pages 660–667, 2012.

[34] Wenfei Fan. Graph pattern matching revised for social network analysis. In *ICDT*, pages 8–21, 2012.

[35] Nicola Ferraro, Luigi Palopoli, Simona Panni, and Simona E. Rombo. Asymmetric comparison and querying of biological networks. *IEEE/ACM Trans. Comput. Biology Bioinform.*, 8(4):876–889, 2011.

[36] Andreas Fischer, ChingY. Suen, Volkmar Frinken, Kaspar Riesen, and Horst Bunke. A fast matching algorithm for graph-based handwriting recognition. In *Graph-Based Representations in Pattern Recognition*, volume 7877 of *Lecture Notes in Computer Science*, pages 194–203. Springer Berlin Heidelberg, 2013.

[37] P. Foggia, C. Sansone, and M. Vento. A database of graphs for isomorphism and sub-graph isomorphism benchmarking. In *CoRR*, pages 176–187, 2001.

[38] B. Gallagher. Finding frequent patterns in a large sparse graph. *AAAI Fall Symposium on Capturing and Using Patterns for Evidence Detection*, pages 45–53, 2006.

[39] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.

[40] L. Giot, J. S. Bader, C. Brouwer, A. Chaudhuri, B. Kuang, Y. Li, Y. L. Hao, C. E. Ooi, B. Godwin, E. Vitols, G. Vijayadamodar, P. Pochart, H. Machineni, M. Welsh, Y. Kong, B. Zerhusen, R. Malcolm, Z. Varrone, A. Collis, M. Minto, S. Burgess, L. McDaniel, E. Stimpson, F. Spriggs, J. Williams, K. Neurath, N. Ioime, M. Agee, E. Voss, K. Furtak, R. Renzulli, N. Aanensen, S. Carrolla, E. Bickelhaupt, Y. Lazovatsky, A. DaSilva, J. Zhong, C. A. Stanyon, R. L. Finley, K. P. White, M. Braverman, T. Jarvie, S. Gold, M. Leach, J. Knight, R. A. Shimkets, M. P. McKenna, J. Chant, and J. M. Rothberg. A Protein Interaction Map of Drosophila melanogaster. *Science*, 302(5651):1727–1736, December 2003.

[41] Marco Gori, Marco Maggini, and Lorenzo Sarti. Exact and approximate graph matching using random walks. *IEEE Trans. Pattern Anal. Mach. Intell.*, 27(7):1100–1111, 2005.

[42] N. Guarino, C. Masolo, and G. Vetere. OntoSeek: content-based access to the web. *Intelligent Systems and Their Applications, IEEE*, 14(3):70–80, 1999.

[43] G T Hart, A K Ramani, and E M Marcotte. How complete are current yeast and human protein-interaction networks? *Genome Biol*, 7(11):120–120, 2006.

[44] L. H. Hartwell, J. J. Hopfield, S. Leibler, and A. W. Murray. From molecular to modular cell biology. *Nature*, 402(6761 Suppl):C47–C52, December 1999.

[45] Huahai He and Ambuj K. Singh. Closure-tree: An index structure for graph queries. In *ICDE '06: Proceedings of the 22nd International Conference on Data Engineering*, pages 38–49, Washington, DC, USA, 2006. IEEE Computer Society.

[46] H Hishigaki, K Nakai, T Ono, A Tanigami, and T Takagi. Assessment of prediction accuracy of protein function from protein–protein interaction data. *Yeast*, 18(6):523–531, 2001.

[47] Sun-Yuan Hsieh. Finding maximal leaf-agreement isomorphic descendent subtrees from phylogenetic trees with different species. *Theor. Comput. Sci.*, 370(1-3):299–308, 2007.

[48] Jun Huan, Wei Wang, Jan Prins, and Jiong Yang. Spin: mining maximal frequent subgraphs from graph databases. In *ACM SIGKDD*, pages 581–586, 2004.

[49] Liang Huang, Ruixuan Li, Yuhua Li, Xiwu Gu, Kunmei Wen, and Zhiyong Xu. 1-graph based community detection in online social networks. In *APWeb*, pages 644–651, 2012.

[50] Trey Ideker and Roded Sharan. Protein networks in disease. *Genome research*, 18(4):644–652, April 2008.

[51] Murat Ilsever and Cem Ünsalam. Graph matching based change detection in satellite images. In *IGARSS*, pages 6213–6215, 2012.

[52] Haoliang Jiang, Haixun Wang, Philip S. Yu, and Shuigeng Zhou. Gstring: A novel approach for efficient search in graph databases. In *ICDE*, pages 566–575, 2007.

[53] R. Jin, C. Wang, D. Polshakov, S. Parthasarathy, and G. Agrawal. Discovering frequent topological structures from graph datasets. In *SIGKDD*, pages 606–611, 2005.

[54] Minoru Kanehisa, Michihiro Araki, Susumu Goto, Masahiro Hattori, Mika Hirakawa, Masumi Itoh, Toshiaki Katayama, Shuichi Kawashima, Shujiro Okuda, Toshiaki Tokimatsu,

and Yoshihiro Yamanishi. KEGG for linking genomes to life and the environment. *NAR*, 36(Database Issue):480–484, 2008.

[55] Maricel G. Kann. Protein interactions and disease: computational approaches to uncover the etiology of diseases. *Brief Bioinform*, 8(5):333–346, 2007.

[56] Arijit Khan, Yinghui Wu, and Xifeng Yan. Emerging graph queries in linked data. In *ICDE*, pages 1218–1221. IEEE Computer Society, 2012.

[57] Sebastian Khler, Sebastian Bauer, Denise Horn, and Peter N. Robinson. Walking the interactome for prioritization of candidate disease genes. *The American Journal of Human Genetics*, 82(4):949 – 958, 2008.

[58] Kwang In Kim, James Tompkin, Martin Theobald, Jan Kautz, and Christian Theobalt. Match graph construction for large image databases. In *ECCV (1)*, pages 272–285, 2012.

[59] Mehmet Koyuturk, Yohan Kim, Shankar Subramanium, Wojciech Szpankowski, and Ananth Grama. Detecting conserved interaction patterns in biological networks. *Journal of Computational Biology*, 3(7):12991322, 2006.

[60] Karl G. Kugler, Laurin A. J. Mueller, Armin Graber, and Matthias Dehmer. Integrative network biology: Graph prototyping for co-expression cancer networks. *PLoS ONE*, 6(7):e22843, 07 2011.

[61] Michihiro Kuramochi and George Karypis. Finding frequent patterns in a large sparse graph. *DMKD*, 11(3):243–271, November 2005.

[62] Kasper Lage, E. Olof Karlberg, Zenia M. Storling, Pall I. Olason, Anders G. Pedersen, Olga Rigina, Anders M. Hinsby, Zeynep Tumer, Flemming Pociot, Niels Tommerup, Yves Moreau, and Soren Brunak. A human phenome-interactome network of protein complexes implicated in genetic disorders. *Nature Biotechnology*, 25(3):309–316, March 2007.

[63] Jungmin Lee, Minsu Cho, and Kyoung Mu Lee. Hyper-graph matching via reweighted random walks. In *CVPR*, pages 1633–1640, 2011.

[64] Kyubum Lee, Sunwon Lee, Minji Jeon, Jaehoon Choi, and Jaewoo Kang. Drug-drug interaction analysis using heterogeneous biological information network. In *BIBM*, pages 1–5, 2012.

[65] Siming Li, Christopher M. Armstrong, Nicolas Bertin, Hui Ge, Stuart Milstein, Mike Boxem, Pierre-Olivier O. Vidalain, Jing-Dong D. Han, Alban Chesneau, Tong Hao, Debra S. Goldberg, Ning Li, Monica Martinez, Jean-François F. Rual, Philippe Lamesch, Lai Xu, Muneesh Tewari, Sharyl L. Wong, Lan V. Zhang, Gabriel F. Berriz, Laurent Jacotot, Philippe Vaglio, Jérôme Reboul, Tomoko Hirozane-Kishikawa, Qianru Li, Harrison W. Gabel, Ahmed Elewa, Bridget Baumgartner, Debra J. Rose, Haiyuan Yu, Stephanie Bosak, Reynaldo Sequerra, Andrew Fraser, Susan E. Mango, William M. Saxton, Susan Strome, Sander Van Den Heuvel, Fabio Piano, Jean Vandenhaute, Claude Sardet, Mark Gerstein, Lynn Doucette-Stamm, Kristin C. Gunsalus, J. Wade Harper, Michael E. Cusick, Frederick P. Roth, David E. Hill, and Marc Vidal. A map of the interactome network of the metazoan C. elegans. *Science (New York, N.Y.)*, 303(5657):540–543, January 2004.

[66] Liang Lin, Yongyi Lu, Yan Pan, and Xiaowu Chen. Integrating graph partitioning and matching for trajectory analysis in video surveillance. *IEEE TIP*, 21(12):4844–4857, 2012.

[67] Chao Liu, Chen Chen, Jiawei Han, and Philip S. Yu. GPLAG: Detection of software plagiarism by program dependence graph analysis. In *ACM SIGKDD*, pages 872–881, 2006.

[68] Shuai Ma, Yang Cao, Jinpeng Huai, and Tianyu Wo. Distributed graph pattern matching. In *WWW*, pages 949–958, 2012.

[69] V. A. McKusick. *Mendelian Inheritance in Man. A Catalog of Human Genes and Genetic Disorders*. Johns Hopkins University Press, Baltimore, 12 edition, 1998.

[70] Wolverton Michael, Berry Pauline, Harrison Ian, Lowrance John, Morley David, Rodriguez Andres, Ruspini Enrique, and Thomere Jerome. LAW: A workbench for approximate pattern matching in relational data. In *IAAI*, pages 143–150, 2003.

[71] Laurin Mueller, Karl Kugler, Michael Netzer, Armin Graber, and Matthias Dehmer. A Network-Based approach to classify the three domains of life. *Biology Direct*, 6:53+, October 2011.

[72] T Murali, S Pacifico, J Yu, S Guest, GG Roberts, and Finley RLJ. DroID 2011: A comprehensive integrated reources for protein, transcription factor, RNA, and gene interactions for Drosophila. *Nucleic Acids Research*, in press:D637–D640, 2011.

[73] M. E. J. Newman. The structure and function of complex networks. *SIAM Review*, 45(2):167–256, 2003.

[74] M. E. J. Newman and M. Girvan. Finding and evaluating community structure in networks. *Physical Review*, E 69(026113), 2004.

[75] Arzucan Ozgür, Thuy Vu, Günes Erkan, and Dragomir R. Radev. Identifying gene-disease associations using centrality on a literature mined gene-interaction network. *Bioinformatics (Oxford, England)*, 24(13):i277–285, July 2008.

[76] Xiaoning Qian and Byung-Jun Yoon. Comparative analysis of protein interaction networks reveals that conserved pathways are susceptible to hiv-1 interception. *BMC Bioinformatics*, 12(Suppl 1):S19, 2011.

[77] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2009. ISBN 3-900051-07-0.

[78] Kaspar Riesen, Michel Neuhaus, and Horst Bunke. Graph embedding in vector spaces by means of prototype selection. In *GbRPR*, volume 4538, pages 383–393. Springer, 2007.

[79] Chanchal K. Roy, James R. Cordy, and Rainer Koschke. Comparison and evaluation of code clone detection techniques and tools: A qualitative approach. *Sci. Comput. Program.*, 74:470–495, 2009.

[80] Manoj P. Samanta and Shoudan Liang. Predicting protein functions from redundancies in large-scale protein interaction networks. *Proceedings of the National Academy of Sciences of the United States of America*, 100(22):12579–12583, October 2003.

[81] Kamran Sartipi and Kostas Kontogiannis. On modeling software architecture recovery as graph matching. In *ICSM*, pages 224–234, 2003.

[82] Falk Schreiber and Henning Schwöbbermeyer. Mavisto: a tool for the exploration of network motifs. *Bioinformatics*, 21:3572–3574, 2005.

[83] B. Schwikowski, P. Uetz, and S. Fields. A network of protein-protein interactions in yeast. *Nature biotechnology*, 18(12):1257–1261, December 2000.

[84] Jacob Scott, Trey Ideker, Richard M. Karp, and Roded Sharan. Efficient algorithms for detecting signaling pathways in protein interaction networks. *Journal of computational biology : a journal of computational molecular cell biology*, 13(2):133–144, March 2006.

[85] Roded Sharan and Trey Ideker. Modeling cellular machinery through biological network comparison. *Nature biotechnology*, 24(4):427–433, April 2006.

[86] Roded Sharan, Silpa Suthram, Ryan M. Kelley, Tanja Kuhn, Scott McCuine, Peter Uetz, Taylor Sittler, Richard M. Karp, and Trey Ideker. Conserved patterns of protein interaction in multiple species. *Proceedings of the National Academy of Sciences of the United States of America*, 102(6):1974–1979, February 2005.

[87] Dennis Shasha, Jason T. L. Wang, and Rosalba Giugno. Algorithmics and applications of tree and graph searching. In *PODS '02: Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 39–52, New York, NY, USA, 2002. ACM.

[88] Victor Spirin and Leonid A. Mirny. Protein complexes and functional modules in molecular networks. *Proceedings of the National Academy of Sciences of the United States of America*, 100(21):12123–12128, October 2003.

[89] M. Steffen, A. Petti, J. Aach, P. D'haeseleer, and G. Church. Automated modelling of signal transduction networks. *BMC Bioinformatics*, 3, November 2002.

[90] Jimeng Sun, Yinglian Xie, Hui Zhang, and Christos Faloutsos. Less is more: Sparse graph mining with compact matrix decomposition. *Stat. Anal. Data Min.*, 1(1):6–22, February 2008.

[91] Jie Tang, Jing Zhang, Limin Yao, Juanzi Li, Li Zhang, and Zhong Su. ArnetMiner: extraction and mining of academic social networks. KDD, pages 990–998, 2008.

[92] Yuanyuan Tian, Richard C. McEachin, Carlos Santos, David J. States, and Jignesh M. Patel. SAGA: a subgraph matching tool for biological graphs. *Bioinformatics*, 23(2):232–239, 2007.

[93] Yuanyuan Tian and Jignesh M. Patel. Tale: A tool for approximate large graph matching. *International Conference on Data Engineering*, 0:963–972, 2008.

[94] Devis Tuia, Jordi Muñoz-Marí, Luis Gómez-Chova, and Jesus Malo. Graph matching for adaptation in remote sensing. *IEEE T. Geoscience and Remote Sensing*, 51(1):329–341, 2013.

[95] P. Uetz and R. L. Finley. From protein networks to biological systems. *FEBS Lett*, 579(8):1821–1827, March 2005.

[96] J. R. Ullmann. An algorithm for subgraph isomorphism. *J. ACM*, 23(1):31–42, 1976.

[97] A. Vazquez, A. Flammini, A. Maritan, and A. Vespignani. Global protein function prediction from protein-protein interaction networks. *Nat Biotechnol*, 21(6):697–700, June 2003.

[98] Kavitha Venkatesan, Jean-Francois Rual, Alexei Vazquez, Ulrich Stelzl, Irma Lemmens, Tomoko Hirozane-Kishikawa, Tong Hao, Martina Zenkner, Xiaofeng Xin, Kwang-Il Goh, Muhammed A. Yildirim, Nicolas Simonis, Kathrin Heinzmann, Fana Gebreab, Julie M Sahalie, Sebiha Cevik, Christophe Simon, Anne-Sophie De Smet, Elizabeth Dann, Alex Smolyar, Arunachalam Vinayagam, Haiyuan Yu, David Szeto, Heather Borick, Amelie Dricot, Niels Klitgord, Ryan R. Murray, Chenwei Lin, Maciej Lalowski, Jan Timm, Kirstin Rau, Charles Boone, Pascal Braun, Michael E. Cusick, Frederick P. Roth, David E. Hill, Jan Tavernier, Erich E. Wanker, Albert-Laszlo Barabasi, and Marc Vidal. An empirical framework for binary interactome mapping. *NATURE METHODS*, 6(1):83–90, 2009.

[99] Peggy I. Wang and Edward M. Marcotte. It's the machine that matters: Predicting gene function and phenotype from protein networks. *Journal of Proteomics*, 73(11):2277–2289, October 2010.

[100] Wei Wang, Chen Wang, Yongtai Zhu, Baile Shi, Jian Pei, Xifeng Yan, and Jiawei Han. GraphMiner: a structural pattern-mining system for large disk-based graph databases and its applications. In *ACM SIGMOD*, pages 879–881, 2005.

[101] Takashi Washio and Hiroshi Motoda. State of the art of graph-based data mining. *SIGKDD Explor. Newsl.*, 5(1):59–68, July 2003.

[102] Joshua S. Weitz, Philip N. Benfey, and Ned S. Wingreen. Evolution, Interactions, and Biological Networks. *PLoS Biology*, 5(1):e11+, January 2007.

[103] Sebastian Wernicke and Florian Rasche. Fanmod: a tool for fast network motif detection. *Bioinformatics*, 22:1152–1153, 2006.

[104] David W. Williams, Jun Huan, and Wei Wang. Graph database indexing using structured graph decomposition. In *ICDE*, pages 976–985, 2007.

[105] Xiaowei Xu, Nurcan Yuruk, Zhidan Feng, and Thomas A. J. Schweiger. SCAN: a structural clustering algorithm for networks. In *ACM SIGKDD*, pages 824–833, 2007.

[106] Takuji Yamada and Peer Bork. Evolution of biomolecular networks:lessons from metabolic and protein interactions. *Nature Reviews Molecular Cell Biology*, 10(11):791–803, November 2009.

[107] Xifeng Yan and Jiawei Han. gSpan: Graph-based substructure pattern mining. In *ICDM*, pages 721–724, 2002.

[108] Xifeng Yan, Philip S. Yu, and Jiawei Han. Graph indexing: a frequent structure-based approach. In *SIGMOD '04: Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 335–346, New York, NY, USA, 2004. ACM.

[109] Xifeng Yan, Philip S. Yu, and Jiawei Han. Graph indexing: a frequent structure-based approach. In *SIGMOD*, pages 335–346, 2004.

[110] Xifeng Yan, Philip S. Yu, and Jiawei Han. Substructure similarity search in graph databases. In *SIGMOD '05: Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 766–777, New York, NY, USA, 2005. ACM.

[111] Xifeng Yan, Feida Zhu, Jiawei Han, and Philip S. Yu. Searching substructures with superimposed distance. In *ICDE '06: Proceedings of the 22nd International Conference on Data Engineering*, page 88, Washington, DC, USA, 2006. IEEE Computer Society.

[112] Yip, Y. Kevin, Yu, Haiyuan, Kim, M. Philip, Schultz, Martin, Gerstein, and Mark. The tyna platform for comparative interactomics: a web tool for managing, comparing and mining multiple networks. *Bioinformatics*, 22:2968–2970, December 2006.

[113] Jingkai Yu and Jr Finley, Russell L. Combining multiple positive training sets to generate confidence scores for protein-protein interactions. *Bioinformatics*, 25(1):105–111, 2009.

[114] Mikhail Zaslavskiy, Francis R. Bach, and Jean-Philippe Vert. Global alignment of protein-protein interaction networks by graph matching methods. *Bioinformatics*, 25(12), 2009.

[115] Shijie Zhang, Shirong Li, and Jiong Yang. GADDI: distance index based subgraph matching in biological networks. In *EDBT*, pages 192–203, 2009.

ABSTRACT

ON RANKED APPROXIMATE MATCHING OF LARGE ATTRIBUTED GRAPHS

by

MOHAMMAD SHAFKAT AMIN

August 2014

Advisor: Dr. Hasan Jamil

Major: Computer Science

Degree: Doctor of Philosophy

Many emerging database applications entail sophisticated graph based query manipulation, predominantly evident in large-scale scientific applications. To access the information embedded in graphs, efficient graph matching tools and algorithms have become of prime importance. Although the prohibitively expensive time complexity associated with exact sub-graph isomorphism techniques has limited its efficacy in the application domain, approximate yet efficient graph matching techniques have received much attention due to their pragmatic applicability. Since public domain databases are noisy and incomplete in nature, inexact graph matching techniques have proven to be more promising in terms of inferring knowledge from numerous structural data repositories. Contemporary algorithms for approximate graph matching incur substantial cost to generate candidates, and then test and rank them for possible match. Leading algorithms balance processing time and overall resource consumption cost by leveraging sophisticated data structures and graph properties to improve overall performance.

In this dissertation, we propose novel techniques for approximate graph matching based on two different techniques called *TraM* (Top-*k* Graph Matching) and Approximate Network

Matching or *AtoM* respectively. While TraM off-loads a significant amount of its processing on to the database making the approach viable for large graphs, AtoM provides improved turn around time by means of graph summarization prior to matching. The summarization process is aided by domain sensitive similarity matrices, which in turn helps improve the matching performance. The vector space embedding of the graphs and efficient filtration of the search space enables computation of approximate graph similarity at a throw-away cost. We combine domain similarity and topological similarity to obtain overall graph similarity and compare them with neighborhood biased segments of the data-graph for proper matches. We show that our approach can naturally support the emerging trend in graph pattern queries and discuss its suitability for large networks as it can be seamlessly transformed to adhere to map-reduce framework. We have conducted thorough experiments on several synthetic and real data sets, and have demonstrated the effectiveness and efficiency of the proposed method.

AUTOBIOGRAPHICAL STATEMENT

MOHAMMAD SHAFKAT AMIN

EDUCATION

- Doctor of Philosophy (Computer Science), August 2014
  Wayne State University, Detroit, Michigan, United States
- Master of Science (Computer Science), July 2007
  University of Dhaka, Bangladesh

PUBLICATIONS

- Haishan Liu, Mohammad Shafkat Amin, Baoshi Yan, Anmol Bhasin: Generating supplemental content information using virtual profiles. RecSys 2013: 295-302
- Mohammad Shafkat Amin, Russell L. Finley Jr., Hasan M. Jamil: Top-k Similar Graph Matching Using TraM in Biological Networks. IEEE/ACM Trans. Comput. Biology Bioinform. 9(6): 1790-1804 (2012)
- Mohammad Shafkat Amin, Baoshi Yan, Sripad Sriram, Anmol Bhasin, Christian Posse: Social referral: leveraging network connections to deliver recommendations. RecSys 2012: 273-276
- Mohammad Shafkat Amin, Hasan M. Jamil: An Efficient Web-Based Wrapper and Annotator for Tabular Data. International Journal of Software Engineering and Knowledge Engineering 20(2): 215-231 (2010)
- Mohammad Shafkat Amin, Anupam Bhattacharjee, Russell L. Finley Jr., Hasan M. Jamil: A stochastic approach to candidate disease gene subnetwork extraction. SAC 2010: 1534-1538
- Mohammad Shafkat Amin, Anupam Bhattacharjee, Hasan M. Jamil: A cytoscape based framework for efficient sub-graph isomorphic protein-protein interaction motif lookup. SAC 2010: 1572-1576
- Mohammad Shafkat Amin, Anupam Bhattacharjee, Hasan M. Jamil: Wikipedia driven autonomous label assignment in wrapper induced tables with missing column names. SAC 2010: 1656-1660
- Mohammad Shafkat Amin, Hasan M. Jamil: FastWrap: An Efficient Wrapper for Tabular Data Extraction from the Web. IRI 2009: 354-359
- Mohammad Shafkat Amin, Hasan M. Jamil: Ontology Guided Autonomous Label Assignment in Wrapper Induced Tables with Missing Column Names. IRI 2009: 424-425

AWARDS AND HONORS

- General Motor Corporations Academic Award, Department of Computer Science Wayne State University, 2010.