Efficient methods for querying and mining real-world graph data

Zhu, Linhong

2011

Zhu, L. (2011). Efficient methods for querying and mining real-world graph data. Doctoral thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/53624>

<https://doi.org/10.32657/10356/53624>

ZHU LINHONG

**EFFICIENT METHODS FOR QUERYING AND MINING
REAL-WORLD GRAPH DATA**

ZHU LINHONG

School of Computer Engineering

A thesis submitted to the Nanyang Technological University
in partial fulfillment of the requirement for the degree of
Doctor of Philosophy

2011

Acknowledgments

I would like to express my gratitude and sincere appreciation to my supervisors, **Associate Professor Wee Keong Ng** and **Assistant Professor Byron Choi**. Only from them, I start to learn the fantastic world of research and build my own publication records. Thanks to them, for their very much guidance, constant encouragement, thoughtful criticism and invaluable suggestions. In deed, they help me even in very detailed things such as using latex, drafting email to seniors, presentation skills and so on. Without their support and invaluable patience, I think it is impossible for me to finish my Ph.D. study. They have also always kept on pushing me to advance to higher research levels.

In addition, I would like to thank **Dr. Aixin Sun**, **Dr. James Cheng**, and **Dr. Saven He**. Thanks to Dr Sun, for his help and patience in guiding me in the “Spam blog detection” project. I would also thank Dr Cheng for his much effort in the “approximate graph matching” work. Dr Cheng offered a lot of help whenever I have problems in research. From him, I start to learn how to think creatively and solve problems in a smart way. Additional thanks to Dr He, for his much help in the “graph reachability work”, constant encouragement in Ph.D. study and valuable advices for many research problems.

I am also grateful to the friends in the Center for Advanced Information System (CAIS): Han Shuguo, Wu Min, He Qi, Jianshu Wen, Ramadhana Bramandia, Ardian, Shi bin, Yao Yuxia, Hu Meishan, Wang Lin, Sijie, Youliang, Zhao Peilin, Sun Hongyang, Jin Changjiu, Zhuang Jinfeng, Chu Shumo, Liu Wenting, Wu Lei, Cao Yangjie, Liu Yong, Li Chenliang, Xia Hao, Wu Pengcheng, Li Bin, Li Hui, Li Guangxia, Liu Xin, Alex, Deng Dingxiong, Zha Wei, Huang Bo, and Hai Zhen for their efforts to discuss problems with me, assist me in many ways, bring me much happiness and maintain a good working environment. Especially, I thank Han Shuguo, Wu Min, Bramandia, Ardian, Changjiu, Shumo for their much help in discussion with me whenever I meet problems. Thanks to our Director **Dr. Sourav Bhowmick**, technician Lai Chee Keong and Chua Chiew Song, for their assistance in providing the administration and technical support during my stay at CAIS.

I want to thank some colleagues in CEMNET and ER lab: Jiang Xin, Yu Kuan, Lu Feng, Guan Wei, Cheng Xiangang, Li Gang, Lu Haifeng, Yang Shengbo, Li Yifan,

Zhongfeng, Zhang Yu, Wang Liang, Zhang Juyong, Zhang Peng, Duan Qi, Wang Zheng Xiang, Gao Shenghua, Duan Lixin, Teng Xiao, and Yang Min, who supported me in many ways throughout the duration of my Ph.D. study.

I also want to thank my flatmates and good friends: Zhang Weijia, Tang Xinyi, Shu Zhen, Su Kai and Wu Xiaoxiao, for their much help in many many things, for those happy time when we are together.

Many thanks to Xu Lin, Guo Jian, Zhang Jinbo, Hu Haoyuan, Jiang Yonglei, Xiang Shili, Liu Gengpu, Wu Yu and Cao Yu, for all you did for me.

Thanks to USTC 0211 CSers, “you never walk alone!”. Deepest thanks to my undergraduate roommates: Shao Xianting, Xin Jin, and Lou Hui. Thanks to USTCAAS Commit Members, for all your assistance.

I also want to acknowledge Dr Hanghang Tong, Dr Jiefeng Cheng and Dr. Ruoming Jin for providing us the source code of G-ray, 2-HOP labeling and Path tree labeling. I sincerely thank Prof Jignesh M. Patel, Dr Yuanyuan Tian and Dr Ning Zhang for providing their implementation of TALE. I am grateful to Dr Yufei Tao and Dr Xiaokui Xiao for providing the implementation of RKNN.

Especially, I want to thank my undergraduate mentor Prof Enhong Chen, for bringing me into the world of data mining.

Last but not least, my deepest gratitude to my parents, for their love, support and patience. I would not have had these days without their endless encouragement.

Contents

Acknowledgments	i
List of Figures	vi
List of Tables	viii
List of Algorithms	ix
List of Notations	x
Abstract	xi
1 Introduction	1
1.1 Motivation: Popularity of Graph Databases	1
1.1.1 Web Data	2
1.1.2 Chemical Data	3
1.1.3 Biological Data	4
1.1.4 Multimedia Database	5
1.2 Research Goals	6
1.2.1 Types of Knowledge Discovery Tasks on Graphs	7
1.2.2 Objectives and Scope of Work	7
1.3 Organization of the Thesis	8
2 Background Knowledge	10
2.1 Notations	11
2.2 Review of Graph Reachability Indexing	12
2.2.1 Transitive Closure Approach	13
2.2.2 Interval-Based Approach	14
2.2.3 2-HOP Approach	15
2.2.4 Other Works	17
2.2.5 Reachability Indexes Comparison	20
2.3 Review of Graph Pattern Query Indexing	22
2.3.1 Feature-based Indexing Approach	23
2.3.2 Closure-based Indexing Approach	25
2.3.3 Encoding-based Indexing Approach	26

2.3.4	Graph Pattern Query Indexing Comparison	27
2.4	Review of Graph Partition-based Indexing	28
2.4.1	Geometry-based Partitioning Algorithms	29
2.4.2	Recursive Spectral Bisection	32
2.4.3	Graph Growing and Greedy Algorithms	34
2.4.4	Modularity-based Graph Partitioning Algorithms	35
2.4.5	Graph Partitioning using Semidefinite Programming	36
2.4.6	Comparison on Different Partitioning Algorithms	37
2.5	Why New Indexing Approaches Are Needed?	37
3	Uniform Framework for Ad-hoc Indexes to Satisfy Reachability Queries	40
3.1	Introduction	40
3.2	Labeling Graph Framework for Reachability Query	43
3.2.1	Preliminaries	43
3.2.2	Graph Framework	45
3.2.3	Query Processing on Graph Framework with Reachability Indexing	45
3.3	Graph Framework Construction	46
3.3.1	Heuristics for Graph Partitioning	47
3.3.2	Query Cost Estimation	49
3.4	Experimental Evaluations	50
3.4.1	Index Size and Query Performance Evaluation on Real Data . . .	50
3.4.2	Index Size and Query Performance Evaluation on Synthetic Data	53
3.5	Summary	54
4	Adaptive System for Graph Reachability Index Selection	56
4.1	Introduction	56
4.2	Representing Graphs with Theoretical Metrics	59
4.2.1	Graph-Theoretical Metrics	59
4.2.2	Feature Selection and Graph Transformation	61
4.3	Graph Classification	64
4.3.1	k -NN Classifier and RKNN Classifier	64
4.3.2	Classification using Spectral Graph Partitioning/Clustering	67
4.4	Experimental Evaluations	67
4.4.1	Dataset and Evaluation Metrics	67
4.4.2	Results	68
4.5	Summary	71

5	Structure and Attribution Index-based Approximate Graph Matching over Large Graphs	72
5.1	Introduction	72
5.2	Problem Definition and Computational Hardness	74
5.2.1	Problem Definition	74
5.2.2	Hardness of AGM Problem	76
5.3	SA-Index	77
5.3.1	The Notion of SA-Index	78
5.3.2	Why SA-Index?	82
5.3.3	Efficient Index Construction	83
5.4	On-line Query Processing	85
5.4.1	Computation of the Best Paths	85
5.4.2	Partial Answer Connection Discovery	87
5.5	Experimental Study	89
5.5.1	Performance Evaluation on Real Data	92
5.6	Summary	96
6	Conclusions and Future Work	98
6.1	Conclusions	98
6.2	Future Plan	99
6.2.1	Disk-Resident Graphs	99
6.2.2	Data Value Chain Analysis	99
6.2.3	Graph Abnormal Node Detection	101
	References	103
A	Support for Other Degree of Matching Metrics	119
B	Upper Bound of Optimum	121

List of Figures

1.1	Graph Representation for Web Text and Structure Data.	2
1.2	Graph Representation for Chemical Component.	3
1.3	Representative Graph for Protein Complex	4
1.4	A Gene Ontology Term from QuickGO	5
1.5	Graph Representation for Multimedia Databases	6
2.1	An Example of a Graph.	13
2.2	An Example of the Graph Closure.	14
2.3	Matrix Representation for Transitive Closure.	14
2.4	An Example of the Interval-Based Approach.	15
2.5	An Example of the 2-HOP Approach.	16
2.6	An Example of the DUAL Approach.	17
2.7	A Graph Database.	24
2.8	A Query Graph.	24
2.9	Another Graph Database.	25
2.10	Examples of Graph Closures.	25
2.11	An Example of Closure Tree.	26
2.12	Basic Structures in Chemical Compounds.	26
2.13	An Example of the GString Approach.	27
2.14	Problems of RCB Partition.	30
2.15	An Example for Inertial Partition in 2D.	32
2.16	A Graph and its Laplacian Matrix L	33
2.17	Eigenvalues and Eigenvectors of the Matrix L	33
2.18	An Example of Spectral Partitioning.	34
2.19	Two Examples for Graph Growing Partitioning.	34
2.20	Examples of Modularity-based Partitioning.	36
3.1	An Example of a Real Graph.	41
3.2	Overview of Our Uniform Framework for Querying Large Graphs.	42
3.3	A Partitioning of a Graph and Its Graph Skeleton.	44
3.4	Index Size and Construction Time Comparison on Synthetic Data.	54

3.5 Total Query Time and Number of Labels Accessed Comparison on Synthetic Data. 55

4.1 An example of KNN and RKNN. 64

4.2 Performance Comparison with Equally-splitting between Training and Testing Data. 69

4.3 Prediction Accuracy and Precision Comparison with Equally-splitting between Training and Testing Data. 70

4.4 Prediction Accuracy and Precision Comparison varying Number of Training Graphs. 70

5.1 An Illustrative Example. 73

5.2 Examples of Approximate Graph Matching 75

5.3 An Example of Reducing From γ -Quasi Clique to Decision Problem of Approximate Graph Matching. 76

5.4 A Running Example. 79

5.5 An Example of the SA-Index. 81

5.6 Index Construction Performance Comparison over Real Data. 92

5.7 Index Size Comparison over Real Data. 92

5.8 Effect of Data Graph Size. 93

5.9 Effect of Query Graph Size. 94

5.10 Effect of Using Index. 94

5.11 Effectiveness of Matching Algorithm. 95

5.12 Answer Graphs of Line Query. 95

5.13 Answer Graphs of Star Query. 96

5.14 Answer Graphs of Loop Query. 96

List of Tables

1	Notations and Explanations.	x
2.1	Complexity Comparisons.	21
2.2	Index (Hash table of paths).	24
2.3	Graph Pattern Query Indexing Comparison	28
2.4	Characteristics of Graph Partition Algorithms.	38
3.1	Statistics of Real Graphs.	50
3.2	Index Size Comparison on Real Graphs (Number of Integers).	51
3.3	Construction Time Comparison on Real Graphs (millisec).	52
3.4	Total Query Time Comparison on Real Graphs (millisec).	52
3.5	Labels Accessed During Query Processing on Real Graphs (Number of Integers).	53
4.1	Statistics of Graph Data.	68
4.2	Running Time and Memory Usage of Feature Selection.	68
5.1	Examples of Degree of Matching Metrics and Relationship between γ and δ	76
5.2	Statistics of Datasets.	90

List of Algorithms

2.1	Query Evaluation Comparison.	20
2.2	QueryProecssing query (q, I, D)	23
3.1	A Query Evaluation Algorithm on the Graph Framework evaluate-query	46
3.2	The Heuristic Graph Partition Algorithm partition (G)	48
3.3	Partition Refinement refinement ($G, P(G)$)	49
4.1	Discriminative Feature Selection.	63
4.2	Classifier	65
5.1	Partition-Based Index Construction partition (G)	84
5.2	Connectivity-Based Index Construction	85
5.3	Finding the Best Paths edgetopath ($P^*(G), G_s, HT, Q, \text{sim}$)	87
5.4	Greedily select path greedy (Q, P, sim)	88

Table 1: Notations and Explanations.

Symbols	Descriptions
Q	a query graph
G, G_a	a data graph/an answer graph
$V(G), E(G)$	the node (vertex)/edge set of G
$A^G(v), l^G(v)$	the value of attribute/label of v in G
$N(v, G)$	a set of neighbors of v in graph G
$d(v)$	node v 's degree
$w(v), w(e)$	the weight of a node v /an edge e
$p \triangleright G$	a path p that appears in G
u_e, v_e	the two ending nodes of an edge e
u_p, v_p	the two ending nodes of a path p
$P^*(G)$	the partition-based index of G
HT	the hash table
G_s	the summary graph of G
$h(v)$	homogeneity of a node v
$(\varphi(S))\phi(S)$	(generalized) conductance of node group S
$G \approx Q$	G approximately matches Q
$G/f/\mathcal{L}$	a graph/feature/model
NN_k	k -nearest neighbors
RKNN	reverse k -nearest neighbors
$\mathcal{G}/\mathcal{F}$	a set of graphs/features
dist	a distance function
K^*/K	(optimal)/number of clusters
x_G/y_G	a feature vector/label of a graph
$\sigma(u, v)$	the distance (the length of shortest paths) from u to v

Table 1 gives the notations used throughout the thesis.

Abstract

The ability to extract or retrieve useful knowledge has become one of the most important challenges in government, industry, and scientific communities. Much success has been achieved when the data to be mined/retrieved and their relationships are modeled as graphs. Over the past few years, research on efficient mining and querying graph data has steadily increased. Among many graph-based applications, there are special interests on two types of applications: graph reachability query and approximate graph matching query. To efficiently process the aforementioned querying and mining tasks, indexing is widely adopted. However, the performance of an indexing technique is often influenced by the structural- and attribution- properties of the graph to be indexed. Unfortunately, there has not been much work on applying suitable indexes to a graph or subgraph with a specific structure. In addition, utilizing both structures and attributes to do indexing has not been well studied. In response to the above issues, in this thesis, we establish the following objectives: (i) to investigate efficient and effective algorithms for various tasks including graph reachability query, graph reachability index selection and graph approximate matching query; and (ii) to explore strong relationships between structures and attributes for graph indexing.

To this end, we propose a uniform graph framework that facilitates indexing sub-graphs (as opposed to the entire graph), and reachability query processing. The graph framework is efficiently constructed by heuristic algorithms. We also formulate graph reachability index selection problem as a graph classification problem and demonstrate that graph classification using theoretical metrics is feasible in terms of bounded errors. Finally, we propose an efficient approximate graph matching algorithm that utilizes structure-aware and attribute-aware indexing. We first index based on the similarity of the attributed graph, that is, we partition the large search space into smaller partitions based on structure similarity and attribute similarity. We further build a connectivity-based index as a concise representation of inter-partition connection.

In summary, our novelties lie in the followings: (i) We have formally defined a framework that facilitates indexing subgraphs and demonstrated how reachability queries are evaluated in our framework; (ii) We are the first to raise and formulate the graph reachability index selection as a property-based graph classification problem, and propose a greedy algorithm to select a subset of theoretical metrics that are specifically designed for graph reachability index selection; (iii) we have used a structure- and attribute- aware index (i.e., *SA-Index*) to facilitate approximate graph matching query; (iv) we have developed a greedy algorithm with bounded time complexity and approximate ratio to process approximate graph matching query.

Our experimental results show that: (i) the proposed uniform framework yields smaller indexes and is more efficient in processing reachability queries on large graphs; (ii) our index selection framework works well in terms of both accuracy and precision, which has verified the possibility of graph reachability index selection using graph theoretical metrics and property-based graph classification; and (iii) both *SA-Index* construction and approximate graph matching query processing are very efficient.

Chapter 1

Introduction

In Section 1.1, we describe several graph representations for the data in various domains such as multimedia, web, chemical, and biological. In addition, the popularity of graph representation also motivated us to develop efficient and effective solutions for mining/querying graph data.

Next, in Section 1.2, we list those common tasks of analyzing graph data in the above domains and identify the subset of tasks that we are interested in during the following thesis study.

1.1 Motivation: Popularity of Graph Databases

Much success has been achieved when the data to be mined represent a set of independent entities and their attributes, for example, customer transactions. However, in most domains, there is interesting knowledge to be mined from the relationships between entities. This relational knowledge may take many forms from periodic patterns of transactions to complicated structural patterns of interrelated transactions. Extracting such knowledge requires the data to be represented in a form that not only captures the relational information but also supports efficient and effective mining of this data and comprehensibility of the resulting knowledge. Relational databases and first-order logic are two popular representations for relational data, but neither has sufficiently supported the data mining process.

The graph representation, that is, a collection of nodes and edges between nodes, does support all aspects of the relational data analysis process. As one of the most general forms of data representation, the graph easily represents entities, their attributes, and their relationships to other entities. In the following, we describe several diverse domains

and how graphs can be used to represent the domain. These graph databases may also serve as a benchmark set of problems for comparing and contrasting different querying methods on graph-structured databases.

1.1.1 Web Data

Figure 1.1: Graph Representation for Web Text and Structure Data.

The World Wide Web is a valuable information resource that is complex, dynamically evolving, and rich in structure. Mining the web is a research area that is almost as old as the web itself. Generally, there are three classes of mining tasks: web content mining [Ntou 06, Svor 07], web structure mining [Bar 02, Becc, Brin 98, Klei 99, Krit 06, Shen 06, Wu 05], and web usage mining [Cao 08, Tan 06, Wang 02].

Web content mining aims to answer the following questions: what’s the pattern among the content of the web? Text mining techniques, such as [Mend 96, Zain 95], have been proposed to discover patterns that facilitate classification of web pages and web queries.

When structure is added to web data in the form of hyperlinks, analysts can then perform web structure mining. Web structure mining is the process of using graph theory to analyze the node and connection structure of a web site. In a web graph, nodes represent web pages and edges represent links between the web pages. Additional nodes can be attached to the web page nodes that are labeled with keywords and other textual information found in the web page content. An example is shown in Figure 1.1, where solid arrows represent edges labeled with “hyper-link” and dashed arrows represent edges labeled “keywords”. According to the type of web structural data, web structure mining can be divided into two kinds: 1) Extracting patterns from hyperlinks in the web: a hyperlink is a structural component that connects the web page to a different location; and 2) Mining the document structure: analysis of the tree-like structure of page

CHAPTER 1. INTRODUCTION

structures to describe HTML or XML tag usage. Such tasks can be easily transformed into corresponding tasks in graph mining and have been in-depthly studied by various approaches such as HITS [Klei 99], PageRank [Brin 98], Pagelet [Bar 02], VIPS [Cai 04] and other algorithms [Becc, Krit 06, Shen 06, Wu 05]. As an example, random graph theory could be applied to web page ranking problem such as PageRank [Brin 98].

The rapid growth of web data also provides potential tasks in web usage mining [Cool 97, Sriv 00]. Web usage mining is a process of finding out what users are looking for on the Internet. For instance, we can extract useful information from server logs (i.e., users history). Some users might be looking at only textual data, whereas some others might be interested in multimedia data. Hence, it becomes more and more important to work out research projects and commercial products that analyze user access patterns for different purposes. The applications generated from this analysis can be classified as personalization [Eiri 03], system improvement [Cho 02], site modification [Tyag 10], business intelligence [Abra 03] and usage characterization [Cao 08].

1.1.2 Chemical Data

Figure 1.2: Graph Representation for Chemical Component.

Graphs provide a natural representation for the structural information contained in chemical compounds. Nodes correspond to atoms and edges represent bonds. The node label is the atom type and the edge label is the bond type. Alternatively, separate nodes can be used to represent attributes of the atoms and the bonds, as shown in Figure 1.2. In this case information about the atom's chemical element, charge, and type (whether it is part of an aromatic ring) is given along with attributes of the bond such as type (single, double, triple) and relative three-dimensional (3D) orientation.

Given a chemical compound dataset, a common task is explicitly defined as follows: Given the molecular structure of a compound, identify the compound as mutagenic or not mutagenic. Mutation is a structural alteration in DNA (deoxyribonucleic acid).

CHAPTER 1. INTRODUCTION

Mutagenicity cannot be practically determined for every compound using biological experiments, therefore, accurate evaluation of mutagenic activity from chemical structure is very desirable. After representing each chemical compound as a graph, the common task for the data is to look for frequent substructure in these graphs to perform graph classification. Analysis of these graphs may answer the following question: what commonalities exists in mutagenic or not mutagenic compounds that will help us to learn concepts for classifying the two classes? This question has been addressed by researchers such as [Desh 05, Kram 01].

1.1.3 Biological Data

Figure 1.3: Representative Graph for Protein Complex

In biological data, various relationships could be modeled as graphs, such as gene ontology, protein-protein interaction, gene-regulatory networks, metabolic networks, and so on. Figure 1.3 shows one graph that represents protein-protein interaction relationship. In this graph, each node represents a protein, and each edge models the interaction between two proteins. Biologists may be interested in whether two proteins are in the same complex [Robb 02] and want the biology database to support this “fundamental” query.

Consider the gene ontology (GO) database [Euro] which stores a large number of subgraphs with diverse structures. Gene ontology terms can be considered as vocabularies for describing a gene (a protein) and can be, in turn, described by some other terms. Figure 1.4 shows a screenshot of one gene ontology term (GO:0001674) from QUICKGO [Euro]. Each node is a GO term and each edge represents the relationship between two

Figure 1.4: A Gene Ontology Term from QuickGO

terms. Consider the gene-ontology database, for a known GO term, users may perform a keyword search on QUICKGO to retrieve a set of ancestor or descendant GO terms.

The common task described in above, if the related datasets have been represented as graphs, could be solved by examining whether two nodes are connected to each other via a path, which has been widely studied in recent research [Chen 05b, Cohe 03, Diet 82, He 05, Jin 08, Tris 07, Wang 06, Wu 06, Wu 04].

1.1.4 Multimedia Database

The internet Movie Database (IMDB)¹ maintains a large database of movie and television information. Currently, the IMDB has information on around one million people and four hundred thousand titles. The database includes filmographies for actors, directors, writers, composers, producers, and editors as well as movie information such as titles, release dates, production companies and countries, plot summaries, reviews and ratings, alternative names, genres and awards.

By representing movie information as a graph, relationships between movies, people, and attributes can be captured and included in the analysis. Figure 1.5.a shows one possible representation of information related to a single movie, which represents each movie as a node, with links to attributes describing the movie. With this representation, one task we can perform is to answer the following question:

¹<http://www.imdb.com/interfaces>

1.5.a: One Graph Representation for IMDB. 1.5.b: Another Graph Representation for IMDB.

Figure 1.5: Graph Representation for Multimedia Databases

Given a movie, how could we find its commonalties in the database?

Commonalties of a movie can be implicitly defined as a set of movies that share a subset of attributes with the given movie. For instance, given a movie, a set of other movies that are produced by the same studio and win the same award can be considered as commonalties of the given movie. Using a graph matching algorithm [TSAI 79, TSAI 83, Ullm 76], we could query and report a large fraction of the movie graphs which are similar to the given movie graph.

By connecting people, movies, and other objects that have relationships to each other, a single connected graph can be constructed. For example, Figure 1.5.b shows how different movies may have actors, directors and studios in common. In a similar way, different actors may appear in the same movie, forming relationship between these people. Analysis of this connected graph may answer questions such as:

Is there any interaction between two given actors or what common relationships can we find between objects in the database?

These analysis can be made regarding to the connectivity discovery in graph [Chen 05b, Cohe 03, Diet 82, He 05, Jin 08, Tris 07, Wang 06, Wu 06, Wu 04] and graph pattern mining [Huan 04, Kura 01, Thom 06, Vane 02, Yan 03].

1.2 Research Goals

In all, data and their relationships in multiple domains could be modeled as graphs. In addition, related common mining or analysis tasks in each domain can be transformed into basic analysis techniques on graphs such as graph reachability query, graph pattern mining and graph matching query. In this section, we list some common tasks on graphs

(not exhaustive) that we are interested. Specifically, in this thesis, we focus on developing efficient methods for querying/mining large real-life graphs using basic techniques such as indexing.

1.2.1 Types of Knowledge Discovery Tasks on Graphs

While data mining has become somewhat synonymous with finding frequent patterns in transactional data, the more general term of knowledge discovery encompasses this and other tasks as well.

From another aspect, there are various query or retrieval tasks in graphs such as *graph reachability query* and *graph matching query*. Graph reachability query examines that whether a node can reach another via a path in a graph. Reachability queries have many emerging applications in graph-structured databases. For example, in XML, the ancestor and descendant axes of Xpath can be implemented with reachability queries on the graph representation of XML. Reachability queries are also useful for building a query language [W3C] for the Semantic Web. As required by these applications, it is desirable to have efficient reachability query processing.

Another important type of query is graph matching query. Generally, exact graph matching query is defined as follows: given a query pattern and a large graph (network), return a set of subgraphs that exact match (i.e., are graph isomorphism to) the query pattern. In recent years, the accelerated growth of applications have sparked renewed interests in approximate (inexact) graph matching queries on a large attributed graph. Approximate matching is an important extension to the exact graph matching. As illustrated by Cook and Holder [Cook 06], “graph matching can be compared to regular expression matching; where regular expressions search for patterns in text files, graph matching searches for patterns in abstract graphs”. Hence, approximate graph matching provides analogous capability to wildcards in regular expression matching and is vital for coping with real-world graph data.

1.2.2 Objectives and Scope of Work

In this thesis, we study efficient approaches for common tasks in graph data analysis. Specifically, our research goals are listed as follows:

- To develop efficient algorithms and techniques for reachability query processing in large graphs;

CHAPTER 1. INTRODUCTION

- To propose efficient and effective approaches for approximate graph matching query in large graphs.

In particular, we make the following contributions.

- As we know, the efficiency of a query algorithm determines whether it is applicable to many time-sensitive applications in real-life. Therefore, we have developed a framework which supports graph reachability query efficiently in terms of both offline index construction time and online query processing time. The novelties of proposed framework are listed as follows: (1) We propose a novel concept of “granularity of graph data”, which allows users to build multiple indexes on graph data. To verify this concept, we develop a uniform framework which supports two graph reachability indexes, namely Interval [Agra 89] and HOPI [Chen 06]. (2) We raise an interesting graph reachability index selection problem that recommends the best reachability index for any input graph or any subgraph of a large graph. To deal with this problem, we also propose an efficient knowledge-based approach with bounded prediction error.
- Approximate/exact graph matching query problem itself is non-trivial due to its NP-hard computational complexity. In addition, efficiency issues must be addressed without compromising the quality of applications. We address these issues by the combination of two techniques: indexing techniques and heuristic techniques. Specifically, we are among the first few works that index both structure and attributes of graphs. Moreover, the state-of-the-art of exact or approximate graph matching algorithms do not utilize any indexing structure by preprocessing the given data graph; while our heuristic approximate graph matching algorithm that well utilizes index.

1.3 Organization of the Thesis

The remainder of the thesis is structured as follows:

- Chapter 2 provides the terminologies from the field of graph theory and preliminaries for graph indexing.

CHAPTER 1. INTRODUCTION

- In Chapter 3, we propose a graph framework that supports multiple indexes for reachability tests on the graph, without modifying the indexing scheme. The graph framework is defined as a set of partitions and a graph skeleton. We illustrate how existing query evaluation techniques can be extended to our graph framework. In particular, a reachability query is cast into inter-partition and intra-partition reachability queries on indexes.
- Instead of proposing a new indexing approach to facilitate graph reachability query, in Chapter 4, we formulate an index selection problem and proposed an ADAPTIVE INDEX SELECTION SYSTEM (AISS) that recommends the best reachability index for any input graph. AISS first transforms a set of training graphs into feature vectors, where graph similarity is measured by Euclidean distance in the feature vector space. Next, AISS predicts the best reachability index by classification/clustering using discriminative features.
- We study the approximate graph matching query problem in Chapter 5. Approximate/exact graph matching query problem itself is non-trivial due to its NP-hard computational complexity. We address this issue by the combination of two techniques: indexing techniques and heuristic techniques. In the first phase, we propose a structure-aware and attribution-aware indexing to perform preprocessing on graph data. In the second phase, we evaluate the approximate graph match query using a heuristic algorithm. Meanwhile, we also utilize the indexing computed in the first phase to facilitate query processing.
- We conclude our work and discuss the future research plan in Chapter 6.

Chapter 2

Background Knowledge

As introduced in Section 1.2, much effort has been made to find efficient algorithms to handle those common tasks on graph data. Among those works, we may notice that indexing is one of the fundamental techniques that is widely adopted. Using indexing to facilitate query processing is originated from relational database system management [S Ab 95]. Generally, an index is a data structure that organizes data records to optimize certain kinds of retrieval operations. An index allows us to efficiently retrieve all records that satisfy search conditions on the search key fields of the index. We can also create additional indexes on a given collection of data records, each with a different search key, to speed up search operations that are not efficiently supported by the organization used to store the data records.

Later, people also studied how could we utilize the index to optimize query performance in semi-structured data or XML. A variety of query languages [W3C] have been proposed to query semi-structured data or XML. All of these query languages are built around path expressions, which are used to traverse graph-structured data. Typically, a data node is selected by a path expression if some path to the node has a sequence of labels matched by the expression. The navigation of the graph structure underlying the semi-structured data and XML is therefore an essential component for querying these data. A naive evaluation of path expressions that scans all data is obviously very computationally expensive. Researchers would like to build an index structure to speed up the evaluation process. A structural summary can be used to prune the search space significantly, thus improving the evaluation performance. Alternatively, an index graph, consisting of a structural summary along with stored mapping from index nodes to data nodes, may be directly used to evaluate such path expressions.

CHAPTER 2. BACKGROUND KNOWLEDGE

Existing indexing approach for semi-structured data is based on the notation of bisimilarity [Gent 03, Henz 95]. For example, Milo and Suciu [Milo 99] proposed a one-index approach, which computes an accurate structural summary that considers incoming paths up to the root of the whole graph. The one-index summary is safe and sound. Path expressions can be directly evaluated in the index graph and can retrieve label-matching nodes without referring to the original data graph. Unfortunately, one-index structural summaries are usually quite large and are considered not efficient enough to speed up the evaluation. Exploiting the observation that long and complex paths tend to contribute disproportionately to the complexity of an accurate summary structure, the $A(k)$ -Index [Kaus 02] relaxes the equivalence condition and considers only incoming paths whose lengths are no longer than k . By taking advantage of the similarity of short paths, the $A(k)$ -Index has been experimentally shown to have a substantially reduced index size. However, the performance of $A(k)$ -Index highly depends on how to choose parameter k . Later, Chen et al. [Chen 03] address this issue by proposing an adaptive index approach $D(k)$ -Index.

In this Chapter, we aim to provide preliminaries about those indexing techniques related to graph data. Specifically, in the following, we given a brief introduction on graph reachability indexing, partition-based indexing and indexing for graph pattern mining and graph matching query. For easy presentation, first in Section 2.1, we would provide some terminologies from the field of graph data. We review existing works on graph reachability indexing in Section 2.2; and in Section 2.3, we take a short overview on a set of representative graph indexing approaches for graph pattern query. We discuss the existing graph partitioning algorithms that are used to build partition-based indexing in Section 2.4. Those aforementioned issues lead to our discussion about the demand to propose new indexing approach in Section 2.5.

2.1 Notations

We denote a data graph (or simply a graph) as a 4-tuple, $G=(V, E, l, \Sigma)$, where V is the set of nodes (vertices) of G ; $E \subseteq V \times V$ is the set of edges of G ; and $l: V \rightarrow \Sigma$ is a function that maps a node to a label (e.g., name of people, name of protein, etc.) in Σ , and Σ is the alphabet set.

If the graph is an attributed graph, then each node in a graph is associated with an attribute, which is denoted as A^G . If the graph is a node- and edge- weighted graph, then each node $v \in V$ /edge $e \in E$ is associated with a numeric value $w(v)/w(e)$.

CHAPTER 2. BACKGROUND KNOWLEDGE

For easy representation, throughout the thesis, we use $V(G)$ and $E(G)$ to denote the node (vertex) set and edge set of a graph G . For a node $v \in V(G)$, we use $A^G(v)$ to denote the value of its attribute, $l^G(v)$ its label, $d(v)$ its degree and $N(v, G)$ its neighbor set. We use $p \triangleright G$ to denote a path p that appears in G . In addition, we use u_e and v_e , and u_p and v_p to denote the two ending nodes of an edge $e \in E(G)$ and a path $p \triangleright G$, respectively. We define an auxiliary function $\text{reach}(v_1, v_2)$ that returns true *iff* v_1 can reach v_2 .

A query graph (or simply a query) is defined as $Q=(V, E, l^Q, \Sigma \cup \{\lambda\})$, which is the same as a data graph except that $l^Q: V \rightarrow \Sigma \cup \{\lambda\}$, where λ is the *null* string.

In addition, we use G to denote a single graph and $\mathcal{G}$ to denote a graph set. Given a graph G , we use x_G to denote its feature vector representation and $I_G \in \{1, 2, \dots, \Gamma\}$ to denote the labels that represent the i th reachability index. We use f to denote a dimension in the feature vector space.

2.2 Review of Graph Reachability Indexing

Specifically, given two vertices u and v , a reachability query returns true if and only if there is a directed path from u and v (denoted $u \rightsquigarrow v$); otherwise, the query returns false. In order to speed up the reachability query applications, reachability indexing has been extensively studied. If we only consider the reachability in trees, the interval-based method is a nice solution that takes linear space and supports reachability queries in constant time. This approach can be dated back to [Diet 82], and has been used in recent work [Zhan 01]. It assigns each node u in the tree by an interval $[\text{start}(u), \text{end}(u)]$. The intervals can be computed with a depth-first traversal of the tree, using a counter that is incremented whenever the traversal enters or leaves a node; $\text{start}(u)$ and $\text{end}(u)$ are assigned the value of the counter when the traversal enters or leaves u , respectively. It is not difficult to see that interval indexing has the following property: Given two tree nodes u and v , u can reach v *iff* $\text{start}(u) < \text{start}(v)$ and $\text{end}(u) \geq \text{end}(v)$. Thus, reachability can be verified in constant time. [Wu 04] proposes to use prime numbers to encode reachability information of a tree. A node is labeled with a product of prime numbers. A node is reachable from another node *iff* its label is divisible by the label of the other node. Unfortunately, these two approaches could not be directly applicable to graphs.

Labeling a general graph to support efficient reachability queries is a challenging problem. Briefly, the naive approach is the transitive approach. And the two most

CHAPTER 2. BACKGROUND KNOWLEDGE

relevant and popular schemes are the interval-based approach for DAG (INTERVAL) by Agrawal *et al.* [Agra 89] and the 2-HOP approach by Cohen *et al.* [Coh03]. The details are discussed in Section 2.2.1 to Section 2.2.3.

Other works, [Wu 06] extends previous work on [Wu 04] to support DAGs. Dual labeling (DUAL) [Wang 06] combines the interval approach for trees and a technique for processing non-tree edges of a graph to yield constant query time and small index size. [Tris 07] proposes an efficient relational-based implementation that bases on the interval and 2-HOP techniques to index directed graphs. The details of some selected indexing approaches would be presented in Section 2.2.4.

Finally, we analyze and compare the performance of different indexing approaches in section 2.2.5. Section 2.2.5 mainly provides some intuitions for index selection procedure, discussed in Chapter 4.

2.2.1 Transitive Closure Approach

Definition 2.1 Consider a directed graph G , the transitive closure of G is a graph $G^+ = (V, E^+, l, \Sigma)$ such that $\forall v, w \in V$ there is an edge (v, w) in E^+ iff there is a non-null path from v to w in G .

Figure 2.1: An Example of a Graph.

The naive approach to handle reachability query is to compute and store the transitive closure of the graph. After the preprocessing of constructing the transitive closure, all reachability queries can be answered in constant time by simply reporting a matrix entry (if assuming that transitive closure relationship is stored as a matrix). Transitive closure is fundamental in propagating the consequences of modified attributes of a graph.

Example 2.1 Consider an example of a graph shown in Figure 2.1. Its graph closure is shown in Figure 2.2 and matrix representation is given in Figure 2.3.

Figure 2.2: An Example of the Graph Closure.

However, it need $O(|V|^2)$ space to store the transitive closure of graph G in worst case. In addition, when a node or an edge is updated in the graph, the value of all entries must also be updated in its matrix representation for transitive closure. Many applications involve massive graphs, yet require fast answering of reachability queries. This makes the naive approach infeasible.

	r	a	b	c	d	e	f
r	1	1	1	1	1	1	1
a	0	1	0	0	1	1	1
b	0	0	1	0	1	0	1
c	0	0	0	1	1	0	1
d	0	0	0	0	1	0	0
e	0	0	0	0	0	1	0
f	0	0	0	0	0	0	1

Figure 2.3: Matrix Representation for Transitive Closure.

2.2.2 Interval-Based Approach

Although originally proposed for trees [Diet 82], the interval-based approach was extended by Agrawal *et al.* [Agra 89] to DAGs. Each node u is assigned a set of non-overlapping intervals $L(u)$; u can reach v iff every interval in $L(v)$ is contained in some interval in $L(u)$. Labeling is done by first finding a spanning forest and assigning interval labels for nodes in the forest. Next, to capture reachability relationships through non-spanning-forest edges, we add additional intervals to labels in reverse topological order of the DAG; specifically, if (u, v) is an edge not in the spanning forest, then all intervals in $L(v)$ are added to $L(u)$ (as well as labels of all nodes that can reach u).

CHAPTER 2. BACKGROUND KNOWLEDGE

Figure 2.4: An Example of the Interval-Based Approach.

Example 2.2 Consider the same graph in Figure 2.1. Using the spanning tree rooted at r (which is shown in dashed lines in Figure 2.4), we label a , d , e , and f with $[1, 5)$, $[2, 3)$, $[3, 4)$, and $[4, 5)$, respectively; we also label b and c with $[5, 6)$ and $[6, 7)$. As shown in Figure 2.4, both b and c also receive intervals from d and f , resulting in $L(b) = \{[5, 6), [2, 3), [4, 5)\}$ and $L(c) = \{[6, 7), [2, 3), [4, 5)\}$.

With more complicated graphs, the size of a label can become linear in the graph size. For example, if d and f have many non spanning-forest descendants, their labels will become larger, which will in turn cause ancestors of b and c to have larger labels. Since reachability queries involve checking containment for all intervals in a label, large labels can seriously deteriorate query performance.

2.2.3 2-HOP Approach

The 2-HOP approach was recently proposed as an alternative to INTERVAL by Cohen *et al.* [Coh03]. Let G be a directed graph. For each node $v \in V(G)$, let $L_{in}(v)$ denote the set of nodes that can reach v , and $L_{out}(v)$ be the set of nodes that can be reached by v . A 2-HOP reachability indexing on graph G assigns every node $v \in V$ a label $L(v) = (L_{in}(v), L_{out}(v))$. The key observation of this approach is that every node in $L_{in}(v)$ can reach every node in $L_{out}(v)$. That is to say, a node v is reachable from a node u if and only if $L_{out}(u)$ intersects the $L_{in}(v)$, i.e., $L_{out}(u) \cap L_{in}(v) \neq \emptyset$.

Figure 2.5: An Example of the 2-HOP Approach.

Example 2.3 Consider an example shown in Figure 2.5, $L_{in}(a) = \{a, r\}$ and $L_{out}(a) = \{a, d, e, f\}$; while $L_{in}(d) = \{r, a, b, c, d\}$ and $L_{out}(d) = \{d\}$. Therefore, node d can be reachable from the node a as the $L_{out}(a)$ intersects the $L_{in}(d)$.

As there exists lots of possible 2-HOP labels for one given graph, researchers mainly focus on finding a 2-HOP label with minimum size. This is called “2-HOP cover” problem, which has been proved as NP-hard in [Cohe 03]. [Cohe 03] showed that a greedy algorithm exists to compute a nearly optimal solution for the 2-HOP cover problem. The resulting size of the greedy algorithm is larger than the optimal by at most $O(\log n)$.

As the computation for transitive closure, one of the input of 2-HOP cover, is quite computation expensive. Later on, [Sche 04] proposed a divide-and-conquer algorithm to build the indexes, which reduce the index construction complexity from $O(n^4)$ to $O(n^3)$. However, it has no improvement in reducing indexing size. In response to it, [Chen 06] proposed a novel geometry-based algorithm which computes high-quality 2-HOP cover fast. Instead of the pre-construction requirement for transitive closure table, [Chen 06] build multiple interval labeling for the given graph and represent it as reachability map. Then, similar to the greedy choosing the vertex v whose bipartite graph covers the most uncovered edges in [Cohe 03], they choose the vertex v covering the most uncovered area in the reachability map. The experimental results in the work [Chen 06] verified that their approach works efficiently in reducing the index construction complexity and the index size.

CHAPTER 2. BACKGROUND KNOWLEDGE

2.2.4 Other Works

DUAL Labeling. Wang *et al.* [Wang 06] proposed a novel method called dual labeling (DUAL) to handle reachability queries for massive sparse graphs. In this approach, it considered a graph as having two components: a tree (spanning tree) plus a set of non-tree edges, which is denoted as t . The dual labeling scheme seamlessly integrates 1) interval-based labels, which encode reachability in the spanning tree; and 2) non-tree labels, which encode additional reachability in the rest of the graph. At query time, first it consults the interval-based labels to see if two nodes are connected by tree edges, if not, then consults non-tree labels, and checks if they are connected by paths that involve non-tree edges. The non-tree edge information can either be stored as the transitive link table or transitive link count table (TLC).

Definition 2.2 If there is a non-tree edge from a node labeled $[a, b)$ to a node labeled $[c, d)$, then we record the edge in a transitive link table. We denote this link by: $a \rightarrow [c, d)$.

Definition 2.3 The TLC function $N(x \cdot y)$ computes the number of links $i \rightarrow [j, k)$ in the transitive link table that satisfy $i \geq x$ and $y \in [j, k)$.

Figure 2.6: An Example of the DUAL Approach.

Example 2.4 As shown in Figure 2.6, based on the tree edges with dashed lines, we assign the tree labels to each node: $L(a)=[1, 5)$, $L(b)=[5, 6)$, $L(c)=[6, 7)$, $L(d)=[2, 3)$, $L(e)=[3, 4)$, $L(f)=[4, 5)$. Then, the transitive link table for the non-tree edges are: $6 \rightarrow [2, 3)$, $6 \rightarrow [4, 5)$, $5 \rightarrow [2, 3)$, $5 \rightarrow [4, 5)$. And the transitive link count table for the non-tree edges are: $N(6 \cdot 2)=1$, $N(6 \cdot 4)=1$, $N(5 \cdot 2)=2$, $N(5 \cdot 4)=2$.

CHAPTER 2. BACKGROUND KNOWLEDGE

This approach consists of two schemes, DUAL-I and DUAL-II. The DUAL-I labeling scheme has constant query time while DUAL-II labeling has logarithmic query time. For sparse graphs, the labeling complexities of both DUAL-I and DUAL-II are almost linear. The DUAL-II scheme has higher query complexity but uses less space in practice. We summarize the main idea of DUAL-I and DUAL-II algorithms as following:

1. Merge the strongly connected components for the given input digraph.
2. Label the set of tree edges based on the minimum equivalent graph.
3. Compute the interval-based labels for the tree edges.
4. Construct the transitive-link table or counting for the non-tree edges.

The difference between the DUAL-I and DUAL-II algorithms lies on the storage for the transitive-link counting for the non-tree edges. In DUAL-II, they propose a TLC search tree as an alternative to the TLC matrix, so that users can efficiently search for the value of $N(x \cdot y)$ for any input pair without remembering which grid point (x, y) snaps to. Another difference in DUAL-II is by casting the problem of checking reachability through the transitive link table as a range-temporal aggregation problem. With the above two approaches, DUAL-II reduced the index size a lot in practise, though the query time increased to logarithmic time compared to $O(1)$ time for DUAL-I.

Prime Labeling. As the graph reachability indexing has been intensively studied, there are a variety of works related to it. Wu *et al.* [Wu 06] presented a prime-labeling approach for DAGs. Prime-labeling scheme for DAG assigns three number $(p[v], C_a[v], C_p[v])$ to each vertex v , where $p[v]$ (self-label) is an exclusive prime number, $C_p[v]$ (parents-label) is the product of primer number of its parent, and $C_a[v]$ (ancestor-label) is the product of parents-label of its parent. This scheme is relatively update-friendly.

SSPI Labeling. Chen *et al.* [Chen 05a, Chen 05b] proposed a new indexing, called SSPI, to represent the DAG. A DAG is represented as a node-interval table plus Surrogate & Surplus Predecessor Index (SSPI). Specifically, a path in a DAG can be represented as a mixture of tree-edges and non-tree edges. For a mixed path p contains at least one remaining edge, $p = (\text{tree-edge} | \text{non-tree edge})^* . e . (\text{tree-edge})^*$ where e is the last non-tree edge followed by none or more tree edges. Given a path $p = (\text{tree-edge} | \text{non-tree edge})^* . e . (\text{tree-edge})^*$ from x to y , w is a Surrogate predecessor of y if w is the target vertex of e and $w \neq y$. Otherwise, the source vertex of e is denoted as Surplus Predecessor

CHAPTER 2. BACKGROUND KNOWLEDGE

---

of y . The SSPI table keeps the Surrogate & Surplus Predecessor Index of nodes which are contained in a mixed path from root.

Path-Tree Labeling. Path-Tree labeling [Jin 08] applies a path-decomposition method to partition a DAG into paths. Next, a path-path graph is proposed to capture the path relationship in a DAG and is indexed with the INTERVAL labeling. Each node u in a DAG is assigned with an X label denoting the DFS order, a Y label denoting the path order and the interval labels I of u 's corresponding path in the path-path graph. The reachability query between two nodes u and v can be answered by comparing their X , Y , and I labels.

3-HOP Labeling. Jin et al. [Jin 09] proposed a new 3-HOP indexing scheme for directed graphs with higher density. The basic idea of 3-HOP indexing is to use chain structures in combination with hops to minimize the number of structures that must be indexed. Specifically, they illustrate that it is not necessary to compute the transitive closure. Instead, they develop an efficient algorithm to discover a transitive closure contour, which yields near optimal index size. Given a chain decomposition, the 3-HOP contour construction algorithm is outlined as follows:

1. Calculate contour points, where each pair of contour points represents the link from one chain to another.
2. Build chain-centered bipartite graphs.
3. Then keep doing the following until all contour points are covered:
  - Find the densest subgraph among all chain-centered bipartite graphs
  - Label vertices according to the selected densest subgraph, delete the subgraph from the corresponding chain-centered bipartite graph, and update covering information.

Path-HOP Labeling. In 3-HOP labeling [Jin 09], the chain decomposition has an important effect on the size of the transitive closure contour and hence the size of the final index. In order to address this issue, path-HOP labeling [Cai 10] explores the use of a spanning tree, in place of a chain decomposition, as the underlying graph structure. They verify that a spanning tree can cover more reachable pairs than a chain decomposition.

CHAPTER 2. BACKGROUND KNOWLEDGE

GRAIL Labeling. Yildirim *et al.* [Yild 10] proposed the GRAIL labeling, a very simple indexing scheme, for fast and scalable reachability testing in very large graphs, based on randomized multiple interval labeling. However, in this approach, the number of iterations d controls the trade-off between the size of the index and the query time. That is, a large number of iterations leads to a large size of the index as well as good query response. Therefore, it is possible that one need to manually tune the parameter d to achieve a relatively good performance in real applications.

2.2.5 Reachability Indexes Comparison

Comparison on Query Evaluation. As the indexing approach varies, the reachability query evaluation technique changes as well. For example, one may only need a constant time to access the transitive closure table to test whether one vertex u can reach another vertex v for the naive TC indexing approach. But for other indexes, such as 2-HOP, one shall access the sets $L_{out}(u)$ and $L_{in}(v)$ and test whether $L_{out}(u)$ intersects the $L_{in}(v)$ in order to evaluate the reachability query between u and v . The details are presented in Algorithm 2.1.

Algorithm 2.1: Query Evaluation Comparison.

Input: A graph G with indexes, index type **type**, node x and y

Output: **true** if x can reach y , **false** otherwise

- 1: switch **type**
  - 2: case **interval**
  - 3: $x:[i_{x_1}, j_{x_1}], \dots, [i_{x_n}, j_{x_n}], y:[i_{y_1}, j_{y_1}], \dots, [i_{y_n}, j_{y_n}]$
  - 4:   **return true** if there exists $[i_{x_m}, j_{x_m}] \supseteq [i_{y_n}, j_{y_n}]$
  - 5: case **2-HOP**
  - 6: $x:L_{in}(x), L_{out}(x), y:L_{in}(y), L_{out}(y)$
  - 7:   **return true** if $L_{out}(x) \cap L_{in}(y) \neq \emptyset$
  - 8: case **Dual-I**
  - 9: $x : [a, b), y : [c, d],$ transitive link table: $i \rightarrow [j, k)$
  - 10:   **return true** if $c \in [a, b)$ or there exists $i \rightarrow [j, k)$
       such that $i \subset [a, b]$ and $c \subset [j, k)$
  - 11: case **Dual-II**
  - 12: $x : [a, b), y : [c, d],$ transitive link count $N(\cdot)$
  - 13:   **return true** if $c \in [a, b)$ or $N(a, c) > N(b, c)$
-

CHAPTER 2. BACKGROUND KNOWLEDGE

Table 2.1: Complexity Comparisons.

Indexing	Index size	Indexing time	Query time
INTERVAL [Agra 89]	$O(V ^2)$	$O(V + E)$	$O(V)$
DUAL [Wang 06]	$O(V +t^2)$	$O(V + E +t^3)$	$O(1)$
2-HOP [Cohé 03]	$O(V E ^{\frac{1}{2}})$	$O(V ^4)$	$O(E ^{\frac{1}{2}})$
HOPI [Sche 04]	$O(V E ^{\frac{1}{2}})$	$O(V ^3)$	$O(E ^{\frac{1}{2}})$
Prime [Wu 06]	$O(V ^2)$	$O(V + E)$	$O(V)$
SSPI [Chen 05b]	$O(V + E)$	$O(V + E)$	$O(E - V)$
GRIPP [Tris 07]	$O(V + E)$	$O(V + E)$	$O(E - V)$
PathTree [Jin 08]	$O(V +t'^2)$	$O(E + V \log V)$	$O(\log k^2)$
3-HOP [Jin 09]	$O(\sum_{i=1}^k V_{out}^i + \sum_{i=1}^k V_{in}^i + 2 V)$	$O(k V ^2 contour)$	$O(V)$ or $O(\log V + k)$
GRAIL [Yild 10]	$O(d V)$	$O(d(V + E))$	$O(d)/O(V + E)$

Example 2.5 *Let us still consider the graph in Figure 2.1 and their labels in Example 2.2, 2.3 and 2.4. Then we can determine that c can reach d by interval labels since $L(d) \in L(c)$. For graph with 2-HOP indexes, as $d \in L_{out}(c)$, then we say node c can reach node d . Respectively, for graph with dual indexes, the reachability test for node $c[6,7)$ and $d[4,5)$ returns true since there exists a link $6 \rightarrow [4,5)$ in the transitive link table (or the transitive link count entry $N(6, 4) > 0$)*

Comparisons on Index Size, Construction Time and Query Time. We compare the complexity of index size, construction time and query time, which is summarized in table 2.1, where $|E|$ is the number of edges, $|V|$ is the number of vertices, t is the number of non-tree edges, k is the number of paths, t' is the number of non-tree edges in path-graph in [Jin 08], and d is the number of iterations [Yild 10].

As shown in Table 2.1, INTERVAL indexing has linear construction time complexity, but may perform worse than 2-HOP and DUAL on index size and query time if graph is sparse. In addition, INTERVAL indexing could only work on trees and DAGs. When graph is extremely sparse, the performance of Dual index would be good, either in index size and query time. When graph is dense, the 2-HOP and HOPI would work well either in index size and query time. However, compared with other approaches, it takes longer time to finish the indexes construction for 2-HOP and HOPI.

The performance of more recent works such as path-tree [Jin 08], 3-HOP [Jin 09] and GRAIL [Yild 10] are quite efficient and scalable to some specific data. However, all of

CHAPTER 2. BACKGROUND KNOWLEDGE

---

their performance are highly dependant on a specific property of graphs. For instance, the performance of 3-HOP [Jin 09] can be affected a lot by the number of chains after chain decomposition. And it is possible that for some real data, the chain decomposition produces a large number of chains where each chain is quite short. In this case, the size of the Transitive closure contours can be large, which results in relatively worse performance.

Those index techniques mentioned above are all not so update-friendly. We need to reconstruct indexes for a large portion of data even if the update is tiny. Prime labeling [Wu 06] is relatively update-friendly, but the index size and query performance is worse than interval indexing. And it has the same problem as interval indexing, that is, it could only serve for trees and DAGs. Bramandia et al. [Bram 08] presented the node-separation property that leads to an efficient 2-HOP maintenance approach. After that, they illustrated inefficient cases in deletions of 2-HOP labeling and proposed a simple algorithm for processing them. Their experimental results showed that the incremental maintenance algorithms are efficient.

2.3 Review of Graph Pattern Query Indexing

Given a query graph, graph pattern query returns a set of answer graphs from a database such that the query graph is sub/graph isomorphism or pseudo isomorphism to each of the answer graphs. It is non-trivial due to the hardness of sub/graph isomorphism problem. Various graph indexing approaches have been developed to improve the efficiency of graph pattern query. As viewed from the type of data, existing research on graph indexing have been conducted on two types of graph databases. The first type consists of very large graphs, such as web graphs and social networks. Tian et al. [Tian 08b] proposed a neighborhood-based indexing approach, to facilitate approximate graph matching query. Zhang et al. [Zhan 09] devised a novel distance-based indexing approach to efficiently find matches of query graph in a large graph.

The second type of graph database is a database that consists of a large set of small graphs such as chemical compounds. This type of database is especially popular in scientific domains such as chemistry and bioinformatics. Various indexing approaches [Chen 09, Chen 07, He 06, He 08, Shas 02, Tian 07, Yan 04, Zhan 07, Zhao 07], have been proposed to improve the performance of subgraph queries and similarity queries.

CHAPTER 2. BACKGROUND KNOWLEDGE

From the point of view of representative approaches, there are three kinds of indexing approaches: feature-based indexing approach, closure-based indexing approach and encoding-based indexing approach. In the following, for each of the three approaches, we provide a detailed discussion on one representative work and briefly introduce some other existing works.

2.3.1 Feature-based Indexing Approach

A number of feature-based indexing approaches such as [Chen 07, Shas 02, Shan 08, Tian 07, Yan 04, Zhan 07, Zhao 07] have been proposed to process queries efficiently. All of the feature-based indexing approaches follow the “filtering and candidate verification” paradigm. First, filtering uses the feature-based index to eliminate part of the false results and to produce a candidate answer set. Specifically, feature-based indexing selects a set of features F , and then performs filtering by inclusion logic: given a query graph Q , for each graph $G \in D$ (D denotes a graph database), if $\exists f \in F$ such that $f \subseteq Q$ and $f \not\subseteq G$, then $Q \not\subseteq G$ and we filter out G . Then, candidate verification tests whether each candidate is indeed a supergraph of the query. Since the candidate answer set is in general much smaller than the entire graph database, query processing using the indexes is significantly more efficient than the sequential scan approach. The outline of “filtering and candidate verification” framework, is presented in Algorithm 2.2.

Algorithm 2.2: QueryProecssing query(q, I, D)

Input: q is a query graph; I is a graph index; D is a graph database

Output: R is a set of matched graphs

1: $F = \{f_i \mid f_i \subseteq q \wedge f_i \in I\};$

2: $C = \bigcap_{f_i \in F} f_i.\text{list};$

3: $R = \emptyset;$

4: **for** each $g \in C$ **do**

5:   **if** $q \subseteq g$ **then**

6:      $R = R \cup \{g\};$

7: **return** R

GraphGrep. Among them, Shasha et al. [Shas 02] identify a set of paths that are up to a limited length as features to index graph database. Their approaches can be summarized as follows:

CHAPTER 2. BACKGROUND KNOWLEDGE

Table 2.2: Index (Hash table of paths).

Key	G_1	G_2	G_3
$h(CA)$	1	0	1
$h(BA)$	2	2	1
$h(CAB)$	2	0	1
...			
$h(ABCB)$	2	2	0

Indexing Enumerate the set of all paths, of length up to L , of all graphs in the database and keep these paths in a hashtable.

Filtering Hash all paths, of length up to L , of a query Q , and then filter out graphs that do not contain any paths in Q .

Verification Test subgraph isomorphism between Q and each graph $G \in C$ where C is a candidate answer set.

Figure 2.7: A Graph Database.

Figure 2.8: A Query Graph.

Example 2.6 Consider an example shown in Figure 2.7. The graphgrep could build a path-based indexing offline, as shown in Table 2.2. Then when a query graph Q (as shown in Figure 2.8) comes, they perform the filtering as follows: first, the algorithm scan the hashtable, and find the set of graphs that contain the path CA , BA and CAB , which are $D_{CA}=\{G_1, G_3\}$, $D_{BA}=\{G_1, G_2, G_3\}$, and $D_{CAB}=\{G_1, G_3\}$. Then the candidate answer set $C=D_{CA} \cap D_{BA} \cap D_{CAB}=\{G_1, G_3\}$. Finally, the graphgrep conducts subgraph isomorphism test between each graph in C and the query graph Q and shows that both G_1 and G_3 are in the answer set to the query Q .

The frameworks of other feature-based methods are quite similar to Graphgrep approach except that they identify different structures as features. SAGA [Tian 07] enumerates fragments of graphs up to a size and answers are generated by assembling hits

CHAPTER 2. BACKGROUND KNOWLEDGE

Figure 2.9: Another Graph Database.

of the query fragments. TreePi [Zhan 07] and Shang [Shan 08] use frequent subtrees as index features. FG-index [Chen 07] uses frequent subgraphs as index features. Frequent graph queries are answered without verification and infrequent graph queries require only a small number of verifications. Some other approaches alternatively identify representative substructures as index features. GIndex [Yan 04] uses discriminative frequent fragments as index features to improve filtering rates and reduce index sizes. Zhao *et al.* [Zhao 07] show that frequent tree features plus a small number of discriminative graphs are better than frequent graph features.

2.3.2 Closure-based Indexing Approach

Figure 2.10: Examples of Graph Closures.

Closure-Tree [He 06] and FG-index [Chen 07] adopt the concept of graph closure, to build indexes. Graph closure is defined as a generalized graph that represents a number of graphs. More specifically, graph closure is a structural union of the graphs into a supergraph by some mapping, where common vertices/edges defined by vertex/edge closure. Figure 2.9 shows another example of graph database. The graph closure C_1 of

Figure 2.11: An Example of Closure Tree.

graphs G_1 and G_2 , the graph closure C_2 of graphs G_3 , G_4 and G_5 , and the graph closure C_3 of closure C_1 and C_2 are presented in Figure 2.10.

By using graph closures as bounding boxes, closure-based approaches introduce the Closure-tree structure, which organizes graphs hierarchically where each node summarizes its descendants by a graph closure. Specifically, in the tree structure, each node is a graph closure of its children, and has at least m children and at most M children, where $(M + 1)/2 \geq m$. As an example, Figure 2.11 shows the closure-tree structure of graphs (in Figure 2.9) and their closures (in Figure 2.10).

With the Closure-tree structure, given a query graph, in the search phase, the algorithm traverses the Closure-tree from the top to bottom, filters Closure-tree nodes and graphs by histograms, and then prunes by pseudo subgraph isomorphism tests. After the search phase, the algorithm outputs a candidate answer set, which will be verified further by real subgraph isomorphism in the verification phase.

In FG-index [Chen 07], a node in the FG-index tree represents a cluster of frequent subgraphs and can be regarded as a closure.

2.3.3 Encoding-based Indexing Approach

2.12.a: Line

2.12.b: Cycle

2.12.c: Star

Figure 2.12: Basic Structures in Chemical Compounds.

GString [Jian 07] introduces a novel sequencing method to capture the semantics of the underlying graph data. Specifically, first, they decide the basic structures using

Figure 2.13: An Example of the GString Approach.

domain knowledge. Consider chemical compounds, each of them consists of three basic patterns: *Line*, *Star*, and *Cycle*, as shown in Figure 2.12. Next, they transform the graph representation of a specific domain into a string by parsing the basic structures. For example, Figure 2.13 shows how a chemical compound is represented as a sequence of strings. First, the chemical compound is segmented into four substructures: a line structure with 2 nodes, two cycles with 6 nodes and one more line structure with 3 nodes. Then relationships among each segment are represented as strings.

For query graphs, the same transformation is also performed. Finally, the original graph search problem is converted into a string search problem. In order to facilitate string search, efficient indices are built such as a suffix tree.

Gcode [Zou 08] uses spectral graph coding to encode the structure of a graph into a numerical space. A query graph is encoded in the same way and match query by comparing graph codes.

2.3.4 Graph Pattern Query Indexing Comparison

In this section, we compare briefly each work's strength and limitation, as summarized in Table 2.3. However, there is no clear winner among those approaches. Performance of each approach is dependant on applications for different domains. In addition, each approach has its individual focus. For instance, GDIndex [Will 07] is proposed specifically for applications in Biological study. And GBLENDER [Jin 10a] provides a friendly user interface to facilitate graph query processing for non-experts.

Table 2.3: Graph Pattern Query Indexing Comparison

	feature-based	closure-based	verification-free	encoding-based	Fast Sub-Iso
Graphgrep [Shas 02]	Yes				
GIndex [Yan 04]	Yes				
C-tree [He 06]		Yes			Yes
FG-index [Chen 07]	Yes	Yes	Yes		
GString [Jian 07]				Yes	
TreePi [Zhan 07]	Yes				Yes
GDIndex [Will 07]			Yes		
Tree+ Δ [Zhao 07]	Yes				
Geoding [Zou 08]				Yes	
QuickSI [Shan 08]	Yes				Yes
GBLENDER [Jin 10a]	Yes				

2.4 Review of Graph Partition-based Indexing

Graph partitioning is a very common technique in proposing a good approach to those aforementioned tasks in graph data. In the processing stage, we can apply graph partitioning algorithms to build partition-based indexes, to facilitate query or mining tasks on graphs. In addition, graph partitioning problem itself has been one of the classical problems in combination optimization.

Mathematically, the graph partitioning problem could be defined as follows:

Definition 2.4 *Given a positive integer p and a vertex- and edge- weighted graph G , find p non-overlapping subsets of $V_1, \dots, V_p$ such that:*

1. $\bigcup_{i=1}^p V_i = V, V_i \cap V_j = \emptyset$ where $i \neq j$

CHAPTER 2. BACKGROUND KNOWLEDGE

---

2. *the cut size, i.e., the sum of weights of edges which cross between different partitions, is minimum*

$$3. \sum_{v \in V_i} w(v) \approx \sum_{v \in V} w(v)/p$$

A set $\{V_i \subseteq V, 1 \leq i \leq p\}$ could be called as a p -way partitioning of graph G if it satisfies the condition (i). A partition satisfies the condition (iii) could be called a balanced partitioning of G . In addition, condition (ii) could be replaced by other objective function for some specific problems.

Graph partitioning has been one of the classical problems in combinatorial optimization. The problem seeks to optimize a certain objective function. In general, this is an NP-complete problem. As graph partitioning has wide range of applications in various areas, such as sparse matrix-vector multiplication, solving PDEs, VLSI layout, telephone network design and sparse Gaussian elimination, a large number of graph partitioning schemes have been developed as desired. In this chapter, we mainly focus on the most popular scheme—recursive bisection scheme. As we know, when p equals to 2, the p -way graph partitioning problem is also named “graph bisection”. Hence, when we expand the number of partitions from 2 to p , a natural idea to solve the p -way graph partitioning problem is to do graph bisection recursively. That is, we first obtain a 2-way partition of V , and then we further subdivide each part using 2-way partition (bisection). After $\log p$ phases, graph G is partitioned into p parts. Even though this scheme does not necessarily lead to optimal partition, it is used extensively due to its simplicity.

Even in the recursive bisection graph partitioning scheme, various algorithms have been proposed to deal with different kinds of applications. In the following, we discuss five types of algorithms: geometry-based partitioning algorithms, spectral-based partitioning algorithms, modularity-based partitioning algorithms, graph growing partitioning algorithms and semidefinite-programming-based partitioning algorithms. The details of these graph partition algorithms, are further discussed in the following four subsections.

2.4.1 Geometry-based Partitioning Algorithms

We give a brief overview on two partitioning algorithms that utilize the geometrical properties of the graph: recursive coordinate bisection [Berg 87] and recursive inertial bisection [Fidu 88].

In the recursive coordinate bisection (RCB) [Berg 87], in order to obtain a bisection, they begin by selecting a coordinate axis (Usually, the coordinate axis for which the

CHAPTER 2. BACKGROUND KNOWLEDGE

2.14.a: Partition of the First Method. 2.14.b: Partition of the Second Method. 2.14.c: Partition in Another Different Coordinate System.

Figure 2.14: Problems of RCB Partition.

vertex coordinates have the largest spread in values is selected). Then, they find a plane, which is orthogonal to the selected axis and bisects the vertices of the graph into two equal-size subsets. This involves finding the median of a set of coordinate values.

Recursive application of this technique can be done in two ways: In the first method, they always choose the same axis will lead to thin strips and is usually undesirable since it leads to long boundaries and thus probably to a large cut-size. In the second method, they alternately bisect the x , y , and z coordinates. This leads to a better aspect-ratio and hopefully to a smaller cut-size.

One of the problems of coordinate bisection is that it is coordinate-dependent. Hence in another coordinate system, the same graph may be partitioned quite differently. The so-called inertial bisection described in the next section tries to remedy this problem by choosing its own coordinate system.

Example 2.7 We illustrate the problem of the coordinate bisection algorithm with three examples, shown in Figure 2.14.a, 2.14.b, and 2.14.c. Figure 2.14.a presents the partition of the first RCB method, while Figure 2.14.b shows the partition of the second RCB method. Figure 2.14.c shows another partition of the RCB method in a different coordinate system. The dotted lines denote the bisections and the dashed lines denote the inter-partition edges. As shown in figures, the first method produces the cut-size with 8 while the second method produces the cut-size with 4. Therefore, the second method has better aspect-ratio and smaller cut-size. However, if the coordinate system changes, the graph, which has the same structure as the graph in Figure 2.14.a and 2.14.b, produces a quite different partition result, as shown in Figure 2.14.c.

CHAPTER 2. BACKGROUND KNOWLEDGE

Instead of selecting a coordinate axis, inertial method (RIB) [Fidu 88] selects the axis of minimum angular momentum of the set of vertices. That is, instead of choosing a hyperplane orthogonal to some fixed coordinate axis, one chooses an axis that runs through “the middle” of the points. In three-dimensional space, this axis is equal to the eigenvector associated with the smallest eigenvalue of the matrix I

$$I = \begin{pmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{yx} & I_{yy} & I_{yz} \\ I_{zx} & I_{zy} & I_{zz} \end{pmatrix}$$

where

$$\begin{aligned} I_{xx} &= \sum_{v \in V} ((y(v) - y_c)^2 + (z(v) - z_c)^2) \\ I_{yy} &= \sum_{v \in V} ((x(v) - x_c)^2 + (z(v) - z_c)^2) \\ I_{zz} &= \sum_{v \in V} ((x(v) - x_c)^2 + (y(v) - y_c)^2) \\ I_{xy} &= I_{yx} = - \sum_{v \in V} (x(v) - x_c)(y(v) - y_c) \\ I_{xz} &= I_{zx} = - \sum_{v \in V} (x(v) - x_c)(z(v) - z_c) \\ I_{yz} &= I_{zy} = - \sum_{v \in V} (y(v) - y_c)(z(v) - z_c) \\ (x_c, y_c, z_c) &= \frac{\sum_{v \in V} (x(v), y(v), z(v))}{|V|} \end{aligned}$$

, and $x(v)$, $y(v)$ and $z(v)$ are the x , y , z coordinate of vertex v , respectively.

Then, we continue exactly as in the RCB method: That is, we find a plane, which is orthogonal to the axis of minimum angular momentum, and bisects the vertices of the graph into two equal-sized subsets.

Example 2.8 *We illustrate the inertial method in two-dimension case: Let $P_i = (x_i, y_i)$ be the position of the points, let $\bar{x}$, $\bar{y}$ be the mean of x_i and y_i , and let the line L be given by a point $\bar{p} = (\bar{x}, \bar{y})$ and a direction $u = (v, w)$ with $\|u\| = \sqrt{v^2 + w^2} = 1$ such that $L = \{\bar{p} + \alpha u | \alpha \in R\}$, as shown in Figure 2.15. Then since L is supposed to be the line that minimizes the sum of the squares of the distance from the points to the line, it is easy to see that the “center of mass” (i.e., the point that minimizes the sum of the squares of the distance from the points to a single point) lies on L . So we can simply choose the point $\bar{p} = (\bar{x}, \bar{y})$ as the minimum angular momentum.*

Figure 2.15: An Example for Inertial Partition in 2D.

2.4.2 Recursive Spectral Bisection

RCB and RIB algorithms, which require the geometric information of the graph, can be applied into a limited number of applications. In contrast, the spectral bisection is known to produce good partitions for a wide class of problems. In the spectral bisection, the spectral information is used to partition the graph [Gupt 97b, Poth 90]. Spectral graph theory is the study of eigenvalue and eigenvectors of graphs. These provide relevant information regarding the structure of a graph which is especially important when we are dealing with large graphs.

In the spectral graph theory, algebraic connectivity [Fied 73] and spectral radius are two important terminologies and are widely used to help graph partitioning. Specifically, consider the laplacian matrix $L = D - A$ of a graph G , where A is the matrix representation of the graph G , and matrix D is the diagonal such that d_{ii} is the sum of the row i of matrix A . Then the second smallest eigenvalue of L , is called algebraic connectivity. In addition, the magnitude value of algebraic connectivity is related to the vertex and edge connectivity of the graph G . Hence, Poth et al. [Poth 90] proposed a graph partitioning algorithm that utilizes the algebraic connectivity. This algorithm computes the eigenvector y corresponding to the algebraic connectivity. Next, given the vector y , the vertex set V is partitioned into two parts as follows: Let r be the i^{th} element of the y vector, then if the corresponding eigenvalue y_j of vertex j is larger than r , vertex j is in one partition; otherwise, vertex j is in another partition. Since we are interested in bisections, the value of r is chosen as the weighted median of the values of y_i .

Figure 2.16: A Graph and its Laplacian Matrix L

However, this method is very expensive since it requires the computation of the eigenvector corresponding to the second smallest eigenvalue. Execution time of the spectral methods can be reduced if the computation of finding second smallest eigenvalue is done by using a multilevel algorithm [Gupt 97b]. This multilevel spectral bisection algorithm usually manages to speed up the spectral partitioning methods by an order of magnitude without any loss in the quality of the cut-size.

The spectral radius of a graph, is related to the chromatic number, the independence number and the clique number of a graph. Mathematically, consider the adjacent matrix A of the graph G , its largest eigenvalue λ is called spectral radius of graph G . As the spectral radius is highly related to the density of a connected graph G , Andersen et al. [Ande 07] proposed another spectral graph partitioning algorithm based on spectral radius. In their algorithm, in each iteration, they extract a dense subgraph with large spectral radius and form a new partition.

Figure 2.17: Eigenvalues and Eigenvectors of the Matrix L

Example 2.9 Consider an example shown in Figure 2.16. Given a graph G , first we compute its laplacian matrix L , which is shown in the right part of Figure 2.16. Then we compute the eigenvalues Λ and eigenvectors X of the given laplacian matrix L , as given in Figure 2.17. With the eigenvectors X , we map each node to the corresponding components of the 2nd eigenvector. After that, we split at value zero, put x_1 , x_2 and x_3 into one partition with positive values and the remaining into another partition with negative values. The procedure is illustrated in Figure 2.18.

Figure 2.18: An Example of Spectral Partitioning.

2.4.3 Graph Growing and Greedy Algorithms

When thinking about bisecting the graphs, another idea is: Choosing one starting vertex in some way, and adding vertices to it until the partition is big enough. These algorithms can be classified as “graph growing”- type algorithms (GG). Specifically, we call these algorithms “greedy graph growing”-type algorithms (GGG), if the order of adding vertices is related to a specific objective function. The details are discussed as follows.

In graph growing partitioning algorithm, we start from a vertex and grow a region around it in a breath-first fashion, until half of the vertices have been included (or half of the total vertex weight). The quality of the graph growing partitioning algorithm is sensitive to the choice of a vertex from which to start growing the graph, and different starting vertices yield different cut-size. To partially solve the problem, usually we could randomly select k vertices and grow k different regions. The vertex selection with the smallest cut-size is selected as the partition.

Figure 2.19: Two Examples for Graph Growing Partitioning.

Example 2.10 Figure 2.19 shows two different partitionings based on graph growing algorithms. The first one starts growing from node 1, and put node 1’s neighbors 2, 3, and 5 into the same partition with node 1. The remaining nodes 4 and 6 forms another partition. However, if we start growing from another node, the result will be

CHAPTER 2. BACKGROUND KNOWLEDGE

---

quite different. Figure 2.19.b shows another partitioning which starts from node 2. In this scenario, nodes 1, 2 and 3 together comprise one partition.

In the graph growing algorithm, the region around the start vertex grows in a strict breadth-first fashion. However, in the greedy graph growing algorithms, for instance, three heuristic algorithms in [Kary 98], the next vertex added is the one that is “best” in some sense. For example, for each vertex v , we can define the gain in the cut-size obtained by inserting v into the growing region. Then, we can order the vertices of the graph’s frontier in ascending order according to their gain. Thus, in each step, the vertex with the smallest increase in the cut-size is inserted first. When a vertex is inserted into the growing partition, then the gains of its adjacent vertices already in the frontier are updated, and those not in the frontier are inserted. This greedy algorithm is also sensitive to the choice of the starting vertex, but less than the graph growing algorithm. As an example, in [Chak 04], they use the greedy graph growing partitioning algorithm, in which the vertices selected and added into the new partition with the maximum entropy per vertex. That is, the vertices put into the new partition are exactly the ones whose removal reduces the entropy per node in the original partition.

2.4.4 Modularity-based Graph Partitioning Algorithms

To combat the problem of community detection, a number of new algorithms have been proposed to partition a network into communities of densely connected nodes, with the nodes belonging to different communities being only sparsely connected. The quality of the partitions resulting from these methods, is often measured by the so-called modularity. The modularity of a partitioning is a scalar value between -1 and 1 that measures the density of links inside communities as compared to links between communities. Defined as [Blon 08],

$$Q = \frac{1}{2m} \sum_{i,j} [A_{ij} - \frac{k_i k_j}{2m}] \delta(c_i, c_j),$$

where A_{ij} represents the weight of the edge between vertex i and vertex j , $k_i = \sum_j A_{ij}$ is the sum of the weights of the edges attached to vertex i , c_i is the community to which vertex i is assigned, the δ -function $\delta(u, v)$ is 1 if $u = v$ and 0 otherwise and $m = \frac{1}{2} \sum_{i,j} A_{ij}$. As high values of the modularity correspond to good divisions of a network into communities, then people should be able to find such good partitions by searching through the possible candidates for ones with high modularity. Newman [Clau 04] proposed an

Figure 2.20: Examples of Modularity-based Partitioning.

algorithm that uses a greedy optimization in which, starting with each vertex being the sole member of a community of one, they repeatedly join together the two communities whose amalgamation produces the largest increase in Q . For a network of n vertices, after $n-1$ such joins users are left with a single community and the algorithm stops. The entire process can be represented as a tree whose leaves are the vertices of original network and whose internal nodes correspond to the joins.

Example 2.11 *Modularity is used to validate the quality of a partition, but also as a quantity to optimize in order to find the best partition: Given a network, what is the partitioning with the highest value of Q ? Figure 2.20 provides four different partitionings of the same graph, which results in different Q values. Obviously, the fourth partitioning which yields the highest value of Q is the best among those four.*

2.4.5 Graph Partitioning using Semidefinite Programming

Wolkowicz and Zhao [Wolk 99] present a new semidefinite programming relaxation for the general graph partitioning problem. Their relaxation is obtained through the dual of the Lagrangian dual of a quadratic model of graph partitioning. In particular, their approach adds a quadratic representation of the binary constraints to the quadratic model; and it exploits the structure of a gangster operator in the semidefinite programming relaxation to enforce zeros. These additional constraints provide strengthened bounds for this NP-hard problem.

CHAPTER 2. BACKGROUND KNOWLEDGE

A more recent work proposed by Krauthgamer et al. [Krau 09] formulates a semidefinite programming relaxation for k -balanced partitioning using two well-known ideas. The first one is ℓ_2^2 (a.k.a. negative-type) metrics, which were first used in approximation algorithms by [Aror 09], and the second one is spreading metrics (a.k.a spreading constraints), which were introduced by Even, Naor, Rao, and Schieber [Even 00] in the context of approximation algorithms that employ a divide-and-conquer approach. Finally, they design a rounding procedure that exploits these two types of constraints simultaneously. This bi-criteria approximation algorithm achieved an approximation of $O(\sqrt{\log n \log k})$.

2.4.6 Comparison on Different Partitioning Algorithms

As discussed in the above sections, various algorithms have been proposed to solve graph partitioning problem. And often, it is not easy to categorize these algorithms, and decide which one is better under what scenarios. Some of them depend on the geometrical properties of the graph and are thus only applicable for some specific problems [Berg 87], while others only need the graph itself and no additional information [Kary 98, Kern 70]. Some take a local view of the graph and only try to improve a given partition (*e.g.*, three heuristics in [Kary 98]), while others treat the problem globally [Poth 90]. Some are strictly deterministic, always giving the same result [Gupt 97b, Poth 90] while others use quite a lot of random decisions (*e.g.*, GG graph growing algorithm). Some apply graph theoretic methods [Kern 70] while others just treat the problem as a special instance of another problem [Poth 90]. We categorize these properties of some graph partitioning algorithms, which is shown in table 2.4.

2.5 Why New Indexing Approaches Are Needed?

Since there are multiple existing indexing approaches including graph reachability indexing [Bram 08, Chen 05b, Cohe 03, Diet 82, He 05, Jin 08, Jin 09, Tris 07, Wang 06, Wu 06, Wu 04, Yild 10], graph pattern query indexing [Chen 09, Chen 07, He 06, He 08, Shas 02, Tian 07, Yan 04, Zhan 07, Zhao 07] and graph partition-based indexing [Clau 04, Gupt 97a, Kary 98, Kern 70, Poth 90], the problem here is: why do we need new indexing approaches? We elaborate the reasons as follows.

Table 2.4: Characteristics of Graph Partition Algorithms.

Algorithm	random	need coor- dinates	local	global	problem modeling
RCB [Berg 87]	no	yes	no	yes	geometry
RIB [Fidu 88]	no	yes	no	yes	geometry
SB [Poth 90] [Ande 07]	no	no	no	yes	linear algebra
MSB [Gupt 97b]	no	no	yes	yes	linear algebra
KL [Kern 70]	no	no	yes	yes	graph, heuristic search
GG [Kary 98]	yes	no	yes	no	graph, heuristic search
GGG [Chak 04]	yes	no	yes	no	graph, heuristic search
modularity [Clau 04]	no	no	yes	yes	graph, modularity
SDP [Krau 09] [Wolk 99]	no	no	yes	yes	semidefinite program- ming

The Demand of Uniform Reachability Indexing. Consider an example from relational databases, from which the notion of “indexing” comes. One may build a B^+ tree on a *subset* of the attributes of a relation for range queries and a hash index on some other subsets of attributes for equi-joins. In comparison, to date, a reachability index (such as dual labeling [Wang 06]) is either applied to the entire graph, or not applied at all. We still need a new indexing approach that is generalizable to support multiple indexes in a single graph. To remedy this problem, we propose our new uniform indexing for reachability queries, see Chapter 3.

What’s Beyond Existing Graph Pattern Query Indexing? First, except the two works by [Tian 08b, Zhan 09], all of the above graph pattern query indexing techniques index on a set of relatively small graphs in a graph transaction database and even [Tian 08b, Zhan 09] are inefficient to handle graphs which are up to millions and even billions of nodes and edges. In the preprocessing stage of work [Zhan 09], given a data graph, they generated a set of intersecting subgraphs from pairs of nodes within distance “*Length*” and radius “*k*”. Clearly, the number of intersecting subgraphs would increase sharply when the size of data graph grows. [Tian 08b] used a maximum weighted bipartite graph matching algorithm during the stage of matching important nodes, which is known to be time-consuming if the bipartite graph is large. Obviously, the bipartite graph constructed by a small percentage of millions of nodes would be quite large.

CHAPTER 2. BACKGROUND KNOWLEDGE

Secondly, all of the above approaches are either structure-based indexes or attribute-based indexes. A strong link between structure and attribute has not been previously explored. On the contrary, our work focuses on indexing on a large graph and we index both the structure and attribution information of the graph.

Why New Algorithms for Partition-based Indexing? As introduced earlier in Section 2.4, lots of partitioning algorithms have been proposed, such as METIS [Kary 98], edge betweenness [Kern 70], spectral partitioning [Gupt 97a, Poth 90], modularity [Clau 04], and so on. However, though graph partitioning is a useful technique to facilitate tasks on graph data, there is no guarantee that partitioning resulted by any algorithm would surely optimize the performance of a specific task. Consider the set of tasks that we are interested in (see Section 1.2), if we want to adopt the graph partitioning to facilitate graph reachability query or graph match query, one shall design a suitable objective function and develop new partitioning algorithms accordingly. For instance, in order to facilitate graph reachability query, we propose a heuristic graph partitioning with a specific objective function that optimizes the estimated query cost, see Section 3.3.

Moreover, in an attributed graph, attribute information is also quite important in analysis. Unfortunately, utilizing attributes to perform graph partitioning hasn't attracted enough attention as utilizing structures. Furthermore, we believe that there is a strong link between structures and attributes, which hasn't been studied yet. To deal with these issues, in Chapter 5, we utilize both structures and homogeneity of nodes to perform graph partitioning.

Chapter 3

Uniform Framework for Ad-hoc Indexes to Satisfy Reachability Queries

3.1 Introduction

Recent interests on XML, biological databases, social network analysis, the Semantic Web, Web ontology and many other emerging applications have sparked renewed interests on graph-structured databases (or simply *graphs*) and related problems (e.g., query processing and optimization). In this part of work, we focus on querying large graphs. In particular, we are interested in a kind of fundamental queries from classical graph-structured databases – reachability query.

Reachability queries have many emerging applications in graph-structured databases. For example, in XML, the ancestor and descendant axes of XPATH can be implemented with reachability queries on the graph representation of XML. Reachability queries are also useful for building a query language [W3C] for the Semantic Web. As required by these applications, it is desirable to have efficient reachability query processing approaches.

Example 3.1 *Consider a directed graph that represents a social network of a set of researchers, where the vertices correspond to researchers and the edges correspond to the co-authorship relationship between two researchers, as shown in Figure 3.1. Social network analysis may often require reachability queries. For example, we may ask whether a researcher “Alice” has an Erdős number. A simple way to answer this query is to check whether “Alice” is reachable from “Paul Erdős”.*

Figure 3.1: An Example of a Real Graph.

As introduced in Section 2.2, much indexing technique has been proposed for optimizing reachability queries on trees [Diet 82, Wu 04], directed acyclic graphs (DAGs) [Agra 89, Chen 05b, Wu 06] and arbitrary graphs [Chen 08, Cohe 03, He 05, Jin 08, Tris 07, Wang 06]. These indexes have demonstrated some performance improvement on the graphs with certain structural characteristics.

Unlike relational data, graph-structured data may vary greatly in its structure; e.g., trees, sparse/dense DAGs and sparse/dense cyclic graphs. It is evident that the structure of the graphs has an impact on the performance of reachability indexes on graphs. For instance, dual labeling [Wang 06] works best for sparse graphs but performs suboptimally on dense graphs. Hence, a single reachability index is sometimes not ideal to graphs that have different structures. Given these, we raise the following issues in my research:

1. A notion of data granularity is missing in graph-structured databases. Let us consider an example from relational databases. One may build a B+ tree on a *subset* of the attributes of a relation for range queries and a hash index on some other subsets of attributes for equi-joins. In comparison, to date, a graph index (such as dual labeling [Wang 06]) is either applied to the entire graph, or not applied at all. Is there a general approach to seamlessly support multiple indexes? For example, one may apply dual labeling [Wang 06] to one subgraph and 2-hop [Chen 08] to another?
2. Structural differences between subgraphs have not been considered by state-of-the-art reachability indexes. Many real graphs such as Web graphs, telephone call graphs, and global social network graphs have different structures, both locally and globally. Real graphs, as a whole, are typically sparse with a constant average degree [Gibs 05]. However, there may be local dense subgraphs. Hence, we need

CHAPTER 3. UNIFORM FRAMEWORK FOR AD-HOC INDEXES TO SATISFY REACHABILITY QUERIES

Figure 3.2: Overview of Our Uniform Framework for Querying Large Graphs.

to address structural differences and determine suitable indexing techniques for subgraphs. Is it possible to detect different substructures from a graph and apply suitable indexes to these substructures?

3. Since there are more than 20 indexing techniques, including trees [Diet 82, Wu 04], directed acyclic graphs [Agra 89, Chen 05b, Wu 06], and arbitrary graphs [Chen 08, Cohe 03, He 05, Jin 08, Tris 07, Wang 06], the problem is: how does the user select and decide which index to be built such that will have maximal benefit in query performance and minimal overhead. Clearly, it is still non-trivial for a user (especially when the user is not an expert) to choose a better index from the multiple candidates.
4. Different reachability indexing techniques require different query evaluation algorithms. Is it possible to support multiple indexes and yet reuse existing query evaluation algorithms without modifying the indexes?

To address the above issues, we propose a uniform framework for indexing graphs. With the framework, we can flexibly use any existing index for reachability queries on subgraphs of a graph. An overview of our framework is shown in Figure 3.2. Our framework consists of two components: (1) Graph framework construction through graph partitioning and (2) reachability query evaluation on the graph framework with different indexes. As a proof of concept, our current prototype supports two state-of-the-art indexes for reachability queries, namely Interval [Agra 89] and HOPI [Chen 06].

In summary, we define a graph framework to represent a graph as a set of partitions and a graph skeleton. Each partition can use any existing reachability index. In

conjunction with the framework, we define a cost function and propose a heuristic algorithm for graph partitioning. We illustrate how existing query evaluation techniques can be extended to our graph framework. In particular, a reachability query is casted into inter-partition and intra-partition reachability queries on indexes. We present our experimental evaluation on our framework with both synthetic and real graphs.

The remainder of this chapter is organized as follows: In Section 3.2, we define notations used in this chapter, our graph framework and the evaluation of reachability queries using the graph framework. The graph framework construction is presented in Section 3.3. In Section 3.4, we present an experimental study to evaluate the effectiveness and efficiency of our approach. We summarize this chapter in Section 3.5.

3.2 Labeling Graph Framework for Reachability Query

In the following, we define our *graph framework* to represent reachability information of an arbitrary graph. In brief, the graph framework comprises information of strongly connected components, partitions of a graph and connections between partitions. We evaluate reachability queries efficiently by applying multiple indexing techniques to the graph framework. We defer the details of the construction of the graph framework to Section 3.3.

We describe the definitions used in this chapter in Section 3.2.1. In Section 3.2.2, we present the concept of our graph framework and its properties. In Section 3.2.3, we show how to apply multiple indexes to the graph framework and how to process reachability queries in the graph framework.

3.2.1 Preliminaries

Definition 3.1 (Condensed Graph) *The condensed graph of G is denoted as $G^*=(V^*, E^*, l, \Sigma)$, where a vertex $v_i^* \in V^*(G^*)$ represents a strongly connected component C_i in G and each edge $(v_i, v_j) \in E^*(G^*)$ iff there is at least one edge $(u, v) \in E(G)$ such that $u \in C_i$ and $v \in C_j$.*

The condensed graph can be computed efficiently using Tarjan’s algorithm with time complexity $O(|V(G)|+|E(G)|)$ [Tarj 72].

Example 3.2 *Consider the graph shown in Figure 3.3(a). We partition the graph on the left into three partitions $V_1=\{0, 1, 2, 3, 4, 5\}$, $V_2=\{6, 7, 8, 9\}$ and $V_3=\{10, 11, 12\}$. G_1 , G_2 and G_3 is a dense subgraph, a subtree and a sparse subgraph, respectively.*

Figure 3.3: A Partitioning of a Graph and Its Graph Skeleton.

Based on this partitioning, we define a *partition-level graph* as follows.

Definition 3.2 (Partition-level Graph) Given a partitioning $P(G)$, the partition-level graph $G_p(G)$ is (V_p, E_p) , where each vertex $v_i \in V_p$ represents a partition G_i in $P(G)$ and an edge $(v_i, v_j) \in E_p$ if there is an edge $(u, v) \in E(G)$ such that u and v are vertices in Partitions G_i and G_j , respectively.

Example 3.3 Consider the graph G and its partitioning $P(G)$ in Example 3.2. The partition-level graph $G_p(G)$ is shown in Figure 3.3(b). Vertices 1, 2 and 3 in G_p represent Partitions G_1 , G_2 and G_3 , respectively.

Next, let us consider the relationships between two partitions G_i and G_j that have not been captured by the partition-level graph. We define a *partition-level skeleton graph* $G_{ps}(G)$ to capture the connecting vertices of partitions of a graph G .

Definition 3.3 (Partition-level Skeleton Graph) Given a partition-level graph $G_p(G)$, the partition-level skeleton graph $G_{ps}(G)$ is (V_{ps}, E_{ps}) , where $V_{ps} \subseteq V$, $v \in V_{ps}$ is either a source or a target vertex of an inter-partition edge and E_{ps} is the set of inter-partition edges.

With the above, a graph skeleton is defined as follows:

Definition 3.4 (Graph Skeleton) Given a graph G and a partitioning of G , $P(G)$, the graph skeleton $S(G)$ is a 3-tuple relation $(G_p, G_{ps}, \mathcal{M})$, where $G_p = (V_p, E_p)$ is the partition-level graph, $G_{ps} = (V_{ps}, E_{ps})$ is the partition-skeleton graph and $\mathcal{M} : V_p \rightarrow 2^{V_{ps}}$ is a mapping function that takes a partition v_i as input and gives a subset of V_{ps} : $\{v \mid v \in V_{ps} \text{ and } v \in G_i\}$ as output.

To illustrate the mapping function $\mathcal{M}$, we give an example in Figure 3.3.

3.2.2 Graph Framework

Given the previous preliminary definitions, we propose our graph framework.

Definition 3.5 *A graph framework of a graph G , denoted as $H(G)$, consists of a strongly connected component index $C(V)$, a partitioning of the condensed graph $P(G^*)$ and a graph skeleton $S(G^*)$: $H(G) = (C(V), P(G^*), S(G^*))$.*

We remark that the proposed graph framework has the following properties:

- The graph framework supports multiple reachability indexing techniques. Different partitions may use different indexing techniques.
- The graph skeleton, which contains the inter-partition information, can be indexed in order to facilitate query processing. For example, we could apply the hash index to accelerate the search on the mapping $\mathcal{M}$ and any reachability indexes to the graph G_p and G_{ps} in $S(G^*)$.
- The graph framework consists of indexed subgraphs. Hence, the query evaluation on the graph framework can be transformed into the query evaluation on relevant indexed subgraphs.

3.2.3 Query Processing on Graph Framework with Reachability Indexing

In this subsection, we present an algorithm for evaluating a reachability query on our graph framework with multiple reachability indexes, as shown in Algorithm 3.1.

Algorithm `evaluate-query` returns true if vertex x is able to reach vertex y . It returns false otherwise. The first step is to obtain the strongly connected component for vertices x and y . If they are in the same strongly connected component, the query returns true (Lines 1-2). Next, we compute the partitions where x and y reside, i.e., G_i and G_j (Line 3). If the two input vertices are in the same partition, we use the index of the partition to answer the reachability query (Lines 4-5). Next, if we find that G_i cannot reach G_j by using the index of G_p in $S(G^*)$, then u is not able to reach v and the query returns false (Lines 6-7). Otherwise, we apply the mapping $\mathcal{M}$ to obtain the set of vertices in G_s related to partitions G_i and G_j , i.e., V_i and V_j (Line 8). We test whether there is a vertex $v_i \in V_i$ that is able to reach V_j . If so, we return true if x is able to reach v_i in G_i and v_j reaches y in G_j (Lines 9-12). Otherwise, we return false (Line 13). The correctness of this algorithm can be easily derived from the definition of the graph framework.

CHAPTER 3. UNIFORM FRAMEWORK FOR AD-HOC INDEXES TO SATISFY REACHABILITY QUERIES

Algorithm 3.1: A Query Evaluation Algorithm on the Graph Framework `evaluate-query`

Input: a graph framework $H(G): (C(V), P(G^*), S(G^*))$ with indexes,
two input vertices x and y , where $x, y \in V$

Output: true if x can reach y , false otherwise

```

1: if  $C(x) == C(y)$ 
2:   return true
3: denote  $G_i(G_j)$  to be the partition of  $x(y)$ 
4: if  $G_i = G_j$ 
5:   return true if  $x \rightsquigarrow y$  in  $G_i$ , false otherwise
6: if  $G_i \not\rightsquigarrow G_j$  with respect to  $G_p$  in  $S(G^*)$ 
7:   return false
8: else  $V_i = \mathcal{M}(i), V_j = \mathcal{M}(j)$ 
9:   for each vertex  $v_i \in V_i$ 
10:    for each vertex  $v_j \in V_j$ 
11:      if  $v_i \rightsquigarrow v_j$  with respect to  $G_s$  in  $S(G^*)$ 
12:        return true if  $x \rightsquigarrow v_i$  in  $G_i$  and  $v_j \rightsquigarrow y$  in  $G_j$ 
13: return false

```

Complexity. The time complexity of query processing on graph framework is index-dependant, i.e., it is determined by the reachability indexes applied to the graph framework. For example, assume that vertices x and y are not in the same partition and partition i , containing x , is indexed with interval labeling [Agra 89] with query time complexity $O(|V_i|)$ and partition j , containing y , is indexed with HOPI labeling [Chen 08] with query time complexity $O(|E_j|^{1/2})$. The partition-level graph G_p and partition-level graph G_s are indexed with dual labeling [Wang 06] with constant query time complexity. Hence, in this particular example, the overall complexity of `evaluate-query` is $O(|E_{ij}|(|V_i| + |E_j|^{1/2}))$, where E_{ij} denotes the set of inter-partition edges connecting the partition i and partition j .

3.3 Graph Framework Construction

Given an input graph G , we construct a graph framework as follows: (1) compute condensed graph G^* ; (2) compute a partitioning of G^* , $P(G^*)$, using a heuristic algorithm proposed in Section 3.3.1; (3) based on $P(G^*)$, compute the graph skeleton $S(G^*)$. The key in graph framework construction is to compute $P(G^*)$. Once $P(G)$ is computed, the graph skeleton $S(G)$ can simply be constructed as follows:

CHAPTER 3. UNIFORM FRAMEWORK FOR AD-HOC INDEXES TO SATISFY REACHABILITY QUERIES

---

1. compute the partition-level graph G_p ;
2. compute the set of inter-partition edges E_I ;
3. compute the subgraph G_s which is induced by the edge set E_I ; and
4. compute the mapping $\mathcal{M}$ between a partition and a set of vertices in G_s .

Hence, in this subsection, we focus on the details of determining $P(G^*)$ of a given graph G . First, we propose the objective function of our graph partitioning problem and the heuristic algorithmic strategy to solve the problem. Next, we present how to estimate the query cost of a graph and a graph framework, which is used in the heuristic graph partitioning.

3.3.1 Heuristics for Graph Partitioning

In this subsection, we present the objectivity of our graph partitioning problem. Next, we present the overall procedure for constructing the graph framework.

The graph partitioning problem considered in this work can be formulated as follows: Given a graph G , determine k non-overlapping subsets of $V_1, \dots, V_k$ such that:

1. $\cup_{i=1}^k V_i = V$, $V_i \cap V_j = \emptyset$ where $i \neq j$; and
2. the estimation of query time; i.e., the number of labels accessed during query processing, is minimal,

where k is determined during partitioning.

Overall Algorithm. Let $\Upsilon(G)$ and $\Upsilon(H(G))$ denote the query costs on a graph G and a graph framework $H(G)$, respectively. Our heuristic graph partition algorithm works as follows: Given a k -partitioning of a graph $P(G)$ and its corresponding graph framework $H(G)$, we create a new $(k+1)^{\text{th}}$ partition for a set of vertices $P'(G)$ if the resulting new graph framework $H'(G)$ reduces the query cost; i.e., $\Upsilon(H'(G)) \leq \Upsilon(H(G))$ (shown in Algorithm 3.2). Hence, in Algorithm 3.2, the main strategy is to determine whether further partitioning reduces the cost.

More specifically, we start with a partitioning of G that all the vertices are in the same partition (Line 1). Consider the current k -partition $P(G)$. We find the partition G_{max} whose cost is maximum among all other partitions in $P(G)$ (Line 3). Next, we

---

Algorithm 3.2: The Heuristic Graph Partition Algorithm $\text{partition}(G)$

Input: a digraph G
Output: a partitioning of G , $P(G)$
1: let $k=1$, $C_{\text{pre}}=\Upsilon(G)$, $C_{\text{decre}}=0$, $P(G)=\{G\}$
2: **do**
3:    $G_{\text{max}}=\arg\max_{1 \leq i \leq k} \{\Upsilon(G_i)\}$, $G_i \in P(G)$
4:    $V_{k+1} = \emptyset$
5:   **for** each vertex v of G_{max}
6:     if $\Upsilon(G_{\text{max}} \setminus v) < \Upsilon(G_{\text{max}})$
7:        $V_{k+1} = V_{k+1} \cup v$
8:    $k+1$ -partitioning $P'(G)=\{P(G) \setminus V_{k+1}, G_{k+1}\}$
   /* where G_{k+1} is the subgraph induced by V_{k+1} */
9:   call refinement procedure, $P(G)=\text{Refinement}(G, P'(G))$
10:    $C_{\text{decre}}=\Upsilon(P(G))-C_{\text{pre}}$, $C_{\text{pre}}=\Upsilon(P(G))$, $k = k + 1$
11: **while** $C_{\text{decre}} < 0$
12: **return** $P(G)$

---

assign a set of vertices in G' to a new partition G_{k+1} : for every vertex v in G_{max} , we place it into the new partition *if* its removal from G_{max} decreases its cost (Lines 5-7). This results in a $k+1$ -partitioning $P'(G)$ (Line 8). In order to optimize the quality of the $k+1$ partitions, we invoke the partition refinement procedure $\text{Refinement}(G, P'(G))$ (Line 9) to obtain $P_r(G)$. We proceed to the next partitioning iteration if the cost has been reduced in the current iteration. Otherwise, Algorithm 3.2 terminates and returns $P_r(G)$ (Lines 10-12).

Partition Refinement. Next, we present the details of the refinement procedure (Algorithm 3.3) used in Algorithm 3.2. In the refinement procedure, we improve the quality of a given k -partitioning. We apply a search technique to find a better k -partitioning with a lower cost. Initially, Algorithm 3.3 starts with a partitioning of graph $P(G)$ (Line 1). Then, for each vertex, we search for a good assignment that minimizes the cost: We find the best target partition pid for a vertex v such that the new k -partitioning produced by assigning v to Partition pid has the minimal cost among all other k -partitionings produced by other assignments (Lines 3-6). The partitioning algorithm terminates if the new partitioning found does not decrease the cost or the iteration number is up to a user input value m ; otherwise, Algorithm 3.2 search for other possible assignments (Lines 7-9).

CHAPTER 3. UNIFORM FRAMEWORK FOR AD-HOC INDEXES TO SATISFY REACHABILITY QUERIES

Algorithm 3.3: Partition Refinement $\text{refinement}(G, P(G))$

Input: a digraph G , initial k -partitioning $P(G)$, iteration number m

Output: a new k -partitioning of G , $P_r(G)$

1: $C_{\text{pre}} = E(P(G))$, $C_{\text{decre}} = 0$, $i = 0$

2: **do**

3:   **for** each vertex $v \in V$ of G

4:     let $P_j(G)$ denotes a new partitioning resulted by moving v into partition j in $P(G)$

5:      $\text{pid}(v) = \arg \min_{1 \leq j \leq k} \{\Upsilon(P_j(G)) - \Upsilon(P(G))\}$

6:      $P(G) = P_{\text{pid}}(G)$

7:      $C_{\text{decre}} = \Upsilon(P(G)) - C_{\text{pre}}$, $C_{\text{pre}} = \Upsilon(P(G))$, $i++$

8: **while** $C_{\text{decre}} < 0$ and $i < m$

9: **return** $P(G)$

Complexity. Let the number of iterations needed in the refinement procedure be m . The complexity of the whole partition procedure is $O(mk^2(|V| + |E|))$, where k is the number of partitions. In the worst case (every node forms a partition by itself), $m = n$ and $k = n$. In practice, however, m and k are both very small. The average values of m and k are 80. Thus, our partitioning algorithm is efficient in most cases, as also verified by our experiment.

3.3.2 Query Cost Estimation

In this subsection, we discuss how to model the costs, $\Upsilon(G)$ and $\Upsilon(H(G))$. One of the important properties of our framework is that it is able to support multiple reachability indexes, where any single specified reachability index is a special case of our framework. However, the accuracy of $\Upsilon(G)$ or $\Upsilon(H(G))$ is highly dependant on the cost model of reachability indexes involved. Hence, to compute the value of $\Upsilon(G)$ and $\Upsilon(H(G))$, the pre-condition is that involved reachability indexes have a reasonable cost model. As an illustration, we implement two state-of-the-art indexes, INTERVAL [Wu 06] and HOPI [Chen 06] in our prototype of the graph framework. It has been known that the time complexity for a reachability query on the HOPI and INTERVAL indexes are $O(|E|^{\frac{1}{2}})$ and $O(|V|)$, respectively. Therefore, the query time estimation for $\Upsilon(G)$ can be modeled by Equation 3.1.

$$\Upsilon(G) = \min(C_1|V|, C_2|E|^{\frac{1}{2}}), \quad (3.1)$$

where C_1 and C_2 are the unit cost in real measurements.

CHAPTER 3. UNIFORM FRAMEWORK FOR AD-HOC INDEXES TO SATISFY REACHABILITY QUERIES

Table 3.1: Statistics of Real Graphs.

Type	Name	$ V $	$ E $	$ V ^*$	$ E ^*$	Storage size (KB)
Tree-like graphs	kegg	14271	35170	3617	3908	433
	vchocyc	10694	14207	9491	10143	177
	mtbrv	10697	13922	9602	10245	174
	agrocyc	13969	17694	12684	13408	225
	anthra	13736	17307	12499	13104	220
	human	40051	43879	38811	39576	583
Other graphs	xmark	6483	7654	6080	7025	99
	nasa	5704	7942	5605	6537	100
	hpycyc	5565	8474	4771	5859	101
	eatRS	23219	325624	15466	19916	4033
	hep-th-new	27770	352807	20086	130469	2842
	days	13332	243447	13332	148038	2445

Based on $\Upsilon(G)$, the estimated query time on $H(G)$, which consists of a k -partitioning $P(G)$ and a graph skeleton $S(G)$, is defined as follows:

$$\Upsilon(H(G)) = \sum_{i=1}^k \Upsilon(G_i) + \Upsilon(G_p) + \Upsilon(G_s), \quad (3.2)$$

where k is the number of partitions.

3.4 Experimental Evaluations

In this subsection, we perform an experimental study to evaluate the effectiveness and efficiency of our proposed techniques. All experiments were run on a machine with a 3.4GHZ CPU. The run-times reported are the CPU times. We implemented all the proposed techniques in C++. Regarding graph indexes, we used the HOPI implementation from [Chen 06] and we implemented the Interval scheme for DAGs [Wu 06]. We also used the implementation of a recent path-tree approach from Jin *et al.* [Jin 08].

3.4.1 Index Size and Query Performance Evaluation on Real Data

We used a collection of real graphs in this experiment. We report the statistics of the real graphs in Table 3.1. Among them, “days”, “hep-th-new” and “eatRS” are obtained from a Web graph repository [Bata]; and the other real graphs are provided by Jin *et al.* [Jin 08].

CHAPTER 3. UNIFORM FRAMEWORK FOR AD-HOC INDEXES TO SATISFY REACHABILITY QUERIES

Table 3.2: Index Size Comparison on Real Graphs (Number of Integers).

Tree-like graphs	HOPI	Interval	Ptree	Our	Other graphs	HOPI	Interval	Ptree	Our
kegg	9488	10078	1703	2884	days	199788	227364	–	199826
vchocyc	33920	20196	830	1216	hep-th-new	268524	244748	–	204802
mtbrv	34312	20406	812	1204	eatRS	31032	67952	–	31034
agrocyc	43664	26728	962	1362	hpcyc	16576	11658	4224	2118
anthra	42888	26146	733	1160	nasa	49954	12852	5063	1644
human	84916	79058	965	1438	xmark	44112	14038	2356	1880

In the first set of experiments, we compared the index size of our graph framework with two popular approaches – Interval [Agra 89] and HOPI [Chen 06] and a recent path-tree approach [Jin 08] (PTree). The results are shown in Table 3.2. The reported index size is the number of integers in the indexes.

From the results presented in Table 3.2, we found that the index size of our approach is clearly smaller than that of the HOPI and Interval approaches. The reason is simple: for each subgraph in our framework, our approach chose a relatively better one between the two approaches. In addition, our approach is comparable to the Ptree approach. For graphs that are not “tree-like”, our approach achieved a much smaller index size than the Ptree approach (the bold numbers in Table 3.2). Although our approach is sometimes worse than the Ptree approach for “tree-like” graphs, our approach is more general than the Ptree approach since the Ptree approach could be a special case of our approach (where the Ptree approach is applied to all partitions). However, since an accurate cost model for the Ptree approach has not been available, we did not apply the Ptree approach to any partition (subgraph) in our framework.

Next, we investigate the time for index construction and query processing. The index construction time of our approach is the total time of framework construction and index construction for each subgraph and partition-level graph. Here we only compared our methods with HOPI and the Interval approach. This is because the CPU time measurer and the query evaluator of these three approaches are all implemented by us while the Ptree approach is provided by its authors. All three approaches run on MS Windows while the Ptree approach runs on Linux. Hence, a comparison of indexing and query time between our approach and the Ptree approach may be affected by many implementation issues. When comparing the index construction time and the query time, we used two

CHAPTER 3. UNIFORM FRAMEWORK FOR AD-HOC INDEXES TO SATISFY REACHABILITY QUERIES

Table 3.3: Construction Time Comparison on Real Graphs (millisec).

Tree-like graphs	HOPI	Interval	Our	S_L	S_H	Other graphs	HOPI	Interval	Our	S_L	S_H
kegg	473.7	3.93	4.2	0.94	113	days	9484	93.113	9096	0.01	1.04
vchocyc	1125	7.22	7.34	0.98	153	hep-th-new	56281	66.72	72	0.93	785
mtbrv	1140	5.47	8.269	0.66	138	eatRS	1718	33.1	1728	0.02	0.995
agrocyc	1500	7.19	9.66	0.74	155	hpycyc	609	4.53	5.76	0.79	106
anthra	1453	7	9.72	0.72	149	nasa	1203	3.47	4.40	0.79	273
human	4343	39	59.87	0.65	73	xmark	1000	3.81	4.835	0.79	207

Table 3.4: Total Query Time Comparison on Real Graphs (millisec).

Tree-like graphs	HOPI	Interval	Our	S_L	S_H	Other graphs	HOPI	Interval	Our	S_L	S_H
kegg	1315	1108	1097	1.008	1.2	days	7854	6646	7930	0.84	0.99
vchocyc	4137	2668	2674	0.997	1.5	hep-th-new	5576	7266	5641	0.99	1.3
mtbrv	4128	2672	2738	0.98	1.5	eatRS	3398	3514	3443	0.987	1.02
agrocyc	4287	2839	2797	1.015	1.5	hpycyc	3819	2600	2520	1.03	1.5
anthra	4255	2782	2767	1.005	1.5	nasa	5672	2740	2720	1.007	2.08
human	5003	3957	3901	1.014	1.28	xmark	5179	2678	2699	0.99	1.91

metrics, *i.e.*, S_L , and S_H to evaluate our approach, where $S_L = \frac{\min(\text{HOPI}, \text{Interval})}{\text{Our}}$, and $S_H = \frac{\max(\text{HOPI}, \text{Interval})}{\text{Our}}$. S_L measures the performance comparison between our method and the best of the two static methods. S_H measures the performance gain if an inefficient static method is chosen.

Table 3.3 illustrates that the Interval approach requires two traversals to construct the index and therefore always has the smallest indexing construction time. As known from previous work [Sche 04], the HOPI indexing time for large graphs can often be costly. Since a combination of HOPI and Interval is applied to our prototype implementation, the construction time of our approach is roughly between these two. In particular, the time depends on the percentage of the partitions using each of these indexing approaches. If most of the partitions are using HOPI, the construction time is closer to HOPI, such as the graph “days” and “eatRS”, as shown in Table 3.3.

To study query performance, we issue one million random reachability queries on the indexes constructed for the real graphs. We used an in-memory IO simulation to estimate

CHAPTER 3. UNIFORM FRAMEWORK FOR AD-HOC INDEXES TO SATISFY REACHABILITY QUERIES

Table 3.5: Labels Accessed During Query Processing on Real Graphs (Number of Integers).

Type	Name	HOPI	Interval	Our	S_L	S_H
Tree-like graphs	kegg	663830	415144	407876	1.02	1.63
	vchocyc	3251476	246004	242602	1.02	13.4
	mtbrv	3285414	248322	245036	1.01	13.4
	agrocyc	3292830	241414	237890	1.01	13.8
	anthra	3298963	205832	206216	0.99	16
	human	2160820	78618	80188	0.98	27
Other graphs	xmark	6838694	583146	583262	1.0	11.72
	nasa	8993648	655730	655716	1.0	13.72
	hpycyc	3017708	787830	770372	1.02	3.92
	eatRS	1333936	3181514	1333963	1.0	2.39
	hep-th-new	8869499	47682220	8749328	1.01	5.45
	days	14986610	32996064	14957887	1.0	2.2

the IO cost. The IO simulation performs the following: Reachability labels (i.e., indexes) are stored in pages with the size 4KB. During query processing, we maintain a buffer with the size 4MB. When we check whether two vertices are reachable, we first obtain the ID of pages where the labels of two vertices are kept. Then we access the buffer to read the labels from required pages. If those pages are in buffer, we read the labels from pages directly. Otherwise, before reading labels from those pages, we insert each page into buffer or replace an old page in buffer using LRU replacement policy. Finally, we report the total query time and the number of labels accessed during query processing in Table 3.4 and 3.5, respectively. The average values of S_L in the total query time and the number of labels accessed are 0.99 and 1.006, respectively. These indicate that our approach is comparable to (or slightly better than) the best of Interval and HOPI approaches in the total query time and the number of labels accessed. Moreover, the average values of S_H in the total query time and the number of labels accessed are 1.44 and 10.38, respectively. This supports that our approach avoids the cost of selecting an inefficient static method. In all, our approach is both IO efficient and time efficient.

3.4.2 Index Size and Query Performance Evaluation on Synthetic Data

In Section 3.4.1, we compared our approach with Interval, HOPI and the Ptree approach on real graphs. In order to have a full control over graph structures, we implemented our own graph generator which controls the percentage of tree-like components and graph-like components. The generator works as follows: First, we generate a set of tree-like

Figure 3.4: Index Size and Construction Time Comparison on Synthetic Data.

DAGs and a set of dense DAGs with the maximum fan-out of spanning tree $F=6$, the branch depth of spanning tree $D=6$. Then, we generate a large graph with n vertices by connecting p ($\times 100\%$) dense graphs to $1-p$ ($\times 100\%$) of tree-like graphs, where n and p are the two input parameters.

We generate a set of random graphs with n around 30k, and varying p from 0 to 0.9. The indexing size and index construction time comparison of Interval, HOPI, and our approach is shown in Figure 3.4¹. Similar to the previous experiment, the index size of our approach is significantly better than HOPI and Interval. Regarding the construction time, we find that the construction time of our approach is comparable to the Interval approach when the graph is sparse and slightly worse than the Interval approach when the graph is dense. However, the construction time of our approach is much smaller than that of the HOPI approach.

Figure 3.5 shows the total query time and the number of labels accessed of one million random queries. It is clear that our approach has a better query performance and is more IO efficient than the best of HOPI and Interval.

3.5 Summary

In this chapter, we proposed a uniform framework for efficiently processing reachability query on large graphs. Specifically, a graph is represented by a set of partitions and inter-partition connections. Subsequently, (possibly different) graph indexes can be applied

¹In this experiment, we did not compare the index size of our approach with Ptree [Jin 08], as our files storing the random graphs cannot be recognized by the Ptree implementation in Linux.

CHAPTER 3. UNIFORM FRAMEWORK FOR AD-HOC INDEXES TO SATISFY REACHABILITY QUERIES

Figure 3.5: Total Query Time and Number of Labels Accessed Comparison on Synthetic Data.

to each partition. This facilitates a seamless application of the state-of-the-art of graph indexing on subgraphs represented in the graph framework. Our experimental study verified the effectiveness and efficiency of our framework. In our experiment with a large variety of synthetic graphs and real graphs, our framework consistently produced relatively small indexes when compared to the best index of a non-partition approach. In addition, our experiment showed that the framework improves the query processing performance over the non-partitioning methods.

Chapter 4

Adaptive System for Graph Reachability Index Selection

4.1 Introduction

A graph reachability query answers the following question: For a given graph, does there exist a directed path between a given source vertex and a destination vertex in the graph? Graph reachability queries have recently attracted much attention due to its wide applications in diverse areas: XML documents, biological databases, and social network analysis. In biological databases, reachability queries are generally applied to check if two genes interact. Reachability queries are used in social network analysis to determine whether there is a direct channel of communication from one peer to another.

To efficiently answer queries, indexing techniques are generally used to optimize query performance. There are more than 20 indexing techniques for trees [Diet 82, Wu 04], directed acyclic graphs [Agra 89, Chen 05b, Jin 08, Wu 06], and arbitrary graphs [Cohe 03, He 05, Tris 07, Wang 06], with each indexing technique applicable to more than one type of graph structures. In this chapter, instead of proposing a better novel indexing system, as has been done in aforementioned work, we explore an orthogonal research problem: We want to determine the best reachability index from a set of potential indexes for a specific graph. We refer to this problem as the *Graph Reachability Index Selection* (GRIS) problem.

Problem 4.1 *Given a set of graphs $\mathcal{G}$ and a set of reachability indexes $\mathcal{I}$ where for each graph $G_i \in \mathcal{G}$, the best reachability index I_{G_i} is known, we want to (1) find a function f that maps each $G_i \in \mathcal{G}$ to $I_{G_i} \in \mathcal{I}$; and (2) for a new graph $G \notin \mathcal{G}$ and any $\epsilon > 0$, the probability that the predicted label $f(G)$ (i.e., the predicted best reachability index) is the same as the optimal reachability index for G is at least $1 - \epsilon$.*

CHAPTER 4. ADAPTIVE SYSTEM FOR GRAPH REACHABILITY INDEX SELECTION

We elaborate the needs to study such a problem as follows.

- Consider an example in relational database; one may build a B+ tree on a subset of the attributes of a relation for range queries and a hash index on some other subsets of attributes for equi-joins. Given a data graph, similarly, we may also apply one index to a subgraph and another to other subgraphs. However, the problem here is that for a specific type of sub/graphs, there is no clear winner (best index) among a number of candidates. This motivates us to study the GRIS problem.
- In addition, the borders of graph types are not so distinct in any real application. For instance, it is not easy for a non-expert to clearly tell whether a given graph is a sparse graph, especially when the graph is neither too sparse nor too dense. Hence, optimal reachability index selection for a given sub/graph is still a challenge and worthy to be studied.
- Finally, it is known that the use of an index access structure incurs overheads: it takes up additional storage space and needs additional computational time. Given that there are many candidate indexes to support query processing, the problem is to select an index that improves query performance and incurs minimal overheads. Clearly, it is non-trivial for a user (especially when the user is not an expert) to choose the best index from multiple candidates.

In order to solve the GRIS problem, we adopt the graph classification/clustering approach. One conventional solution to classify graphs is first to compute the graph similarity using various measures such as graph-edit distance [Bunk 97], maximum common subgraph [Gome 00] and canonical sequence similarity [Baba 83], and then decide whether they belong to the same cluster or not based on a pre-defined similarity threshold. Another solution is that one finds subgraph patterns as features [Jin 10b, Ranu 09, Saig 08, Yan 08] and then represents each graph as a feature vector. In this way, the graph classification problem is converted into the classification of traditional high dimensional data points. We refer the above two approaches as the isomorphism-based approach. The isomorphism-based approaches generally assume a hypothesis to verify the similarity of two graphs: If two graphs are similar in size/shape or they share a set of subgraph patterns, then they belong to the same cluster. As a result, graphs within the same cluster only differ from one another by a small number of nodes or edges. This hypothesis guarantees high accuracy for classification tasks for some particular domains,

CHAPTER 4. ADAPTIVE SYSTEM FOR GRAPH REACHABILITY INDEX SELECTION

---

such as in the chemical data where size/shape differences of chemical component are small.

Unfortunately, the above hypothesis is too restrictive to obtain significant outputs for GRIS problem. Consider two trees with huge differences in sizes, they are quite “different” from each other in terms of either graph edit distance or graph sub-patterns. Hence, traditional graph classifiers will assign them with different clusters and different best indexes. However, they do share some implicit graph properties such as sparsity and acyclicity. As a result of having similar implicit graph properties, these two trees also share the same best reachability index.

In lieu of the above issue, in this chapter, we study the feasibility of performing graph reachability index selection by utilizing graph classification according to a general hypothesis: if two graphs are similar in a set of diverse properties, then they share the same best reachability index (i.e., belong to the same class). We call it “property-based graph classification”. More specifically, we define the preliminary concept of property-based graph classification as follows.

Problem 4.2 (Property-based Graph Classification) *Given a graph G , we assume that its properties could be measured by a set of graph theoretical-based metrics (More details about graph metrics will be given in Section 4.2.1). Then the property-based graph classification problem is to induce a mapping $f(x): \chi \rightarrow \{-1, 0, 1, \dots, \gamma\}$, from giving training samples $T = \{ \langle x_i, y_i \rangle \}_{i=1}^L$, where $x_i \in \chi$ represents properties of a graph G and $y_i \in \{-1, 0, 1, \dots, \gamma\}$ is a class label associated with training data, and L is the size of training samples. If $y_i \in \{-1, 1\}$, we denote it as a binary property-based graph classification problem.*

The objectives of this chapter consist of three parts: (1) Investigating a set of graph theoretical metrics to facilitate graph classification; (2) Selecting a subset of features that are specifically designed for graph reachability index selection; and (3) Proposing an efficient and effective classification method for the property-based graph classification.

There are some challenges and issues in this work. First, in order to solve the GRIS problem, what we need is not only simply to enumerate graph metrics, but also to find suitable metrics specifically for classifying graphs with the same best reachability index into a cluster. Another issue lies in designing a good classifier for this specific graph classification problem. Could we just adopt any existing classifiers? Finally, as our approach is more like an expert system, both complexity and accuracy are important design issues.

We address the challenges as follows:

CHAPTER 4. ADAPTIVE SYSTEM FOR GRAPH REACHABILITY INDEX SELECTION

- We give an extensive survey about the most important and diverse graph metrics that have been proposed and used in a wide range of studies in multiple disciplines. We also propose five novel metrics, non-tree density, cut vertex density, connectivity, vertex compression ratio and edge compression ratio to further discover implicit properties.
- Based on domain knowledge, we present an objective of feature selection combining with a greedy algorithm to perform discriminative feature subset selection.
- We explore an application of graph classification using graph metrics. We design two types of classifiers to verify the feasibility of solving graph reachability index selection using graph classification and graph theoretical metrics. The first one is based on searching k -nearest neighbors and reverse k -nearest neighbors; while the second one is built upon spectral graph partitioning.

The organization of this chapter is as follows: In Section 4.2, we enumerate a number of graph theoretical metrics and use them to transform graphs into feature vectors. We discuss how to build up graph classifiers in Section 4.3. In Section 4.4, we perform experiments to verify the effectiveness of our approach. Finally, we summarize this chapter in Section 4.5.

4.2 Representing Graphs with Theoretical Metrics

4.2.1 Graph-Theoretical Metrics

In the following definitions of metrics for graph properties, the definitions apply both to undirected and directed graph unless otherwise specified. In addition, we also provide the related references if they are not first proposed by this work. For notations and descriptions, please refer to Table 1 at the beginning of this thesis.

Metric 1 (Global Clustering Coefficient [Luce 49]) *The global clustering coefficient is defined as: $GCC(G) = \frac{3\Delta}{|(u,v,w):u,v,w \in V, u,v,w \text{ is connected}|}$, where Δ is number of triangles.*

Global clustering coefficient is based on triplets of nodes. A triplet is three nodes that are connected by edges. A triangle consists of three closed triplets, one centrad on each of the nodes. The global clustering coefficient is $3 \times$ triangles over the total number of triplets.

CHAPTER 4. ADAPTIVE SYSTEM FOR GRAPH REACHABILITY INDEX SELECTION

Metric 2 (Characteristic Path Length [Coff 04]) *Characteristic path length is defined as*

$$CPL(G) = \frac{\sum_{u,v} \sigma(u,v)}{|V| * (|V|-1)}$$

While characteristic path length is a measure of the average distance between any two nodes in the graph, global efficiency is a measure of the rate at which nodes can be reached from each other in the graph. It is approximately inversely related to characteristic path length.

Metric 3 (Global Efficiency [Thom 04]) *Global efficiency of a graph is defined as* $GE(G) = \frac{\sum_{u,v} \frac{1}{\sigma(u,v)}}{|V| * (|V|-1)}$.

Metric 4 (Diameter) *Diameter of a graph is defined as* $D(G) = \max_{u,v} \sigma(u,v)$.

Diameter of a graph is the longest shortest paths among any pair of nodes of a graph. In other words, a graph's diameter is the largest number of vertices which must be traversed in order to travel from one vertex to another when paths which backtrack, detour, or loop are excluded from consideration.

Metric 5 (Degree Distribution [Falo 06]) *The degree distribution of a graph is a function* $P: [0, \dots, k_{max}] \rightarrow [0, 1]$, *where* $P(k)$ *is the fraction of the vertices in* G *that have degree* k *for* $0 < k < k_{max}$, *and* k_{max} *is the largest degree in* G .

Metric 6 (Coreness [Seid 83]) *The* k -*core of* G *is the largest subgraph* G_k *of* G *such that every vertex in* G_k *has at least* k *neighbors in* G_k , *i.e.,* $\forall v \in V(G_k), |N(v, G_k)| \geq k$.

Metric 7 (Density) *Given a graph* G , *we define its graph density as:*

$$d_g = \begin{cases} |E|/|V|(|V|-1) & \text{if } G \text{ is directed,} \\ 2|E|/|V|(|V|-1) & \text{if } G \text{ is undirected.} \end{cases}$$

In addition, we propose two more density evaluation metrics, non-tree density and cut vertex density. Non-tree density d_{nt} , is defined as $|T|/|E|$, where T is the set of non-tree edges, i.e., remaining edges after removing a spanning tree of a graph. Cut vertex density $d_c = |V_c|/|V|$ (where V_c is the set of cut vertices; i.e., articulation points) is the proportion of cut vertices.

In the following metrics are restricted to directed graphs only.

CHAPTER 4. ADAPTIVE SYSTEM FOR GRAPH REACHABILITY INDEX SELECTION

Metric 8 (Girth [Dies 05]) *The girth of a graph is the length of the shortest cycle (of length > 3) in the graph.*

Metric 9 (Circumference [Dies 05]) *The circumference of a graph is the length of the longest cycle (of length > 3) in the graph.*

Metric 10 (Connectivity) *Connectivity $|C|$ is the number of strongly connected component(s).*

Metric 11 (Vertex Compression Ratio) *Vertex compression ratio $r_v = |V^*|/|V|$ (where $|V^*|$ is number of vertices in condensed graph given in Definition 3.1 in Chapter 3.) is the proportion of condensed graph vertices.*

Metric 12 (Edge Compression Ratio) *Similar to Def. 11, edge compression ratio $r_e = |E^*|/|E|$ (where $|E^*|$ is number of edges in the condensed graph) is the proportion of condensed graph edges.*

Metric 13 (Acyclicity) $d_e = |E_c|/|E|$ (E_c is the set of distinct edges that appear in a cycle) is a measure of the number of edges participating in one or more cycles. If there is no cycle, then $|E_c| = 0$.

4.2.2 Feature Selection and Graph Transformation

A graph can be viewed as a collection of features on the basis of above metrics. We may simply normalize the values of each features into the range $[0, 1]$ and transform each graph into a feature vector. In the above setting, we just assume that each metric is equally important. In reality, however, it can not be true especially when considering applications in a specific domain. For example, in social network analysis, a small-world network is mainly characterized by two metrics, namely, a shorter characteristic path length and a higher clustering coefficient when compared to random graphs. Then a problem comes: given those aforementioned metrics, how can one select a subset of them to reduce the computation cost and achieve good classification performance? We refer it as a discriminative feature selection problem.

In the following, we adopt similar technology from GSSC [Kong 10], to formulate the feature selection as an optimization problem. Next we discuss possible selection criteria and propose a greedy algorithm to solve the optimization problem. For simplicity, both term “feature” and “metric” are used to mention any of the aforementioned metrics. In

CHAPTER 4. ADAPTIVE SYSTEM FOR GRAPH REACHABILITY INDEX SELECTION

addition, we use f to denote a single feature and $\mathcal{F}$ to denote a feature set. The numeric value of any feature over a graph G is denoted as $f(G)$, and the combination of values of a set of features $\mathcal{F}$ over a graph G is denoted as a vector $\overrightarrow{\mathcal{F}(G)}$.

Mathematically, discriminative feature selection problem could be formulated as:

$$\mathcal{F}_d = \arg \max_{\mathcal{T} \subseteq \mathcal{F}} \{R(\mathcal{T})\} \quad (4.1)$$

where $R(\mathcal{F})$ is a measurement of discriminative over a set of features $\mathcal{F}$.

We investigate two principles that optimal features should have: representativeness and separability. This guides us to design a suitable discriminative measurement R . Separability means that graphs should be able to be separated from each other based on values of features; while representativeness is important to avoid selecting two highly correlated features. Both separability and representability are desirable in feature selection. Hence, we define R as a multi-criteria measurement:

$$R(\mathcal{F}) = \omega_1 \rho(\mathcal{F}) - \omega_2 \frac{1}{|\mathcal{F}|} \sum_{f_i, f_j \in \mathcal{F}} \text{corr}(f_i, f_j) \quad (4.2)$$

where $\rho(\mathcal{F})$ denotes separability ratio of features set $\mathcal{F}$, ω_1 and ω_2 are smoothing weights, and $\text{corr}(f_i, f_j)$ denotes correlation of two features f_i and f_j .

Unfortunately, both separability ratio and feature correlation can be highly domain-dependant. As an example, we provide one possible design for measurements $\rho(\mathcal{F})$ and $\text{corr}(f_i, f_j)$ based on information of labeled background graphs. More specifically, let clustering $\mathcal{C} = \{C_1, \dots, C_\gamma\}$ denote the information of labeled background graphs, i.e., a cluster $C_i = \{G_1, \dots, G_n\}$ is a set of graphs that share the same class label $y_i \in \{-1, 0, 1, \dots, \gamma\}$. Then given clustering $\mathcal{C}$, the separability ratio of a set of features $\mathcal{F}$, is defined as

$$\rho(\mathcal{F} \mid \mathcal{C}) = \max_{\mathcal{C} \in \mathcal{C}} \left\{ \log \frac{\sum_{G_i \in \mathcal{C}} \|\overrightarrow{\mathcal{F}(G_i)}\|}{\sum_{G_j \notin \mathcal{C}} \|\overrightarrow{\mathcal{F}(G_j)}\|} \right\} \quad (4.3)$$

It is clear that the larger the value of $\rho(\mathcal{F} \mid \mathcal{C})$, the better “worst-case” class separable capability feature set $\mathcal{F}$ is.

Similarly, correlation of two features f_i and f_j given clustering $\mathcal{C}$, is defined as

$$\text{corr}(f_i, f_j \mid \mathcal{C}) = \frac{|X - E(X)| |Y - E(Y)|}{\sigma_X \sigma_Y} \quad (4.4)$$

where the values of feature f_i and f_j over graphs in $\mathcal{C}$ are denoted as X and Y respectively.

Algorithm 4.1: Discriminative Feature Selection.

Input: a clustering $\mathcal{C}$ of labeled graphs, a feature set $\mathcal{F}$
Output: a subset of features $\mathcal{F}_d$
1: $\mathcal{T} = \emptyset$, **flag**=true;
2: **while** **flag**
3:    $f_k = \arg \max_{f \in (\mathcal{F} \setminus \mathcal{T})} \{R(\mathcal{T} \cup \{f\} \mid \mathcal{C}) - R(\mathcal{T} \mid \mathcal{C})\};$
4:   **if** $R(\mathcal{T} \cup \{f_k\} \mid \mathcal{C}) - R(\mathcal{T} \mid \mathcal{C}) > 0$
5:      $\mathcal{T} = \mathcal{T} \cup \{f_k\};$
6:   **else**
7:     **flag**=false;
8: **return** $\mathcal{T};$

With respect to Equ. 4.1, we propose a greedy algorithm to solve it, as shown in Algorithm 4.1, we start with an empty set $\mathcal{T} = \emptyset$, and in step k , iteratively add a feature f_k such that it maximizes the margin gain

$$f_k = \arg \max_{f \in (\mathcal{F} \setminus \mathcal{T})} \{R(\mathcal{T} \cup \{f\} \mid \mathcal{C}) - R(\mathcal{T} \mid \mathcal{C})\} \quad (4.5)$$

Complexity. The time complexity of Algorithm 4.1 is $O(n|\mathcal{C}||\mathcal{F}_d|)$, where n is the number of iterations, $|\mathcal{C}|$ is the total number of labeled graphs, and $|\mathcal{F}_d|$ is the number of discriminate features. As the iteration number n is bounded by $O(\log |\mathcal{F}|)$, it is efficient to perform feature selection using Algorithm 4.1.

Selected Features. For the specific task of “graph reachability index selection”, the set of discriminative features that returned by Algorithm 4.1 are:

1. Non-tree density
2. Degree distribution
3. Global efficiency
4. Diameter
5. Vertex compression ratio
6. Coreness
7. Global clustering coefficient

Figure 4.1: An example of KNN and RKNN.

8. Graph density

Now given a graph G in a specific domain, we represent it as a feature vector using discriminative features. We also use the Euclidean distance to measure the dissimilarity of two graphs: $\text{dist}(G_i, G_j) = \sqrt{\|\mathcal{F}(G_i) - \mathcal{F}(G_j)\|^2}$. **For simplicity, we use both x_i and $\mathcal{F}(G_i)$ to denote the feature vector representation of a graph G_i .**

4.3 Graph Classification

In the previous section, we have discussed the features used to describe and represent graphs. In this section, with graph representation using theoretical metrics, we explore the potential of graph classification in the domain of graph reachability index selection. Once graphs are converted into feature vector space, one can use traditional classifier to classify graphs. Therefore, in the following, we introduce three classic classifiers that are widely used in various applications.

4.3.1 k -NN Classifier and RKNN Classifier

As a start, we build two classifiers which are similar to distance weighted KNN classifier. Before we present the details of two classifiers, first let us introduce the notation of “KNN” and “RKNN”.

For query graph G and the set of background graphs $\mathcal{G}$, the set of k -nearest neighbors of G is the smallest set $\text{KNN}(G)$ such that $\text{KNN}(G) \subseteq \mathcal{G}$, $|\text{KNN}(G)| \geq k$, and $\forall G_i \subseteq \text{KNN}(G)$, $\forall G_j \subseteq (\mathcal{G} \setminus \text{KNN}(G))$: $\text{dist}(G, G_i) < \text{dist}(G, G_j)$. Similarly, we define the set of reverse k -nearest neighbors (RKNN [Tao 04]) of a graph G as $\text{RKNN}(G) = \{G_i \subseteq \mathcal{G} \mid G \subseteq \text{KNN}(G_i)\}$.

Note that k -nearest neighbors and reverse k -nearest neighbors are related but not exactly the same concept. We illustrate more using the following examples.

CHAPTER 4. ADAPTIVE SYSTEM FOR GRAPH REACHABILITY INDEX SELECTION

Algorithm 4.2: Classifier

Input: a set of tuples $\mathcal{T} = \langle y, \{x\} \rangle$ and a vector x' representing the test graph
Output: a class label y' representing classification of x'

- 1: **if** KNN classifier
- 2: $\mathcal{T}_{nn} = \text{KNN}(x', \mathcal{T})$;
- 3: **else** $\mathcal{T}_{nn} = \text{RKNN}(x', \mathcal{T})$;
- 4: initializing an array “vote” with length equal to number of classes γ ;
- 5: **for** each tuple $t \in \mathcal{T}_{nn}$
- 6: $\text{vote}[t[1]] += \frac{1}{\text{dist}(\vec{x}', t[2]) + \xi}$
- /* we use $t[1]$ to denote the first entry of tuple t (i.e., the class label)
and $t[2]$ to denote the second entry of tuple t (i.e., a feature vector) */
- 7: **return** $y' = \arg \max_{i=1 \text{ to } \gamma} \text{vote}[i]$

Example 4.1 Let us consider an example shown in Figure 4.1. Figure 4.1(a) shows 4 graphs $G_1, G_2, \dots, G_4$, where each graph is assigned with a circle covering its two nearest neighbors. For example, $2\text{NN}(G_1) = \{G_2, G_3\}$ and both G_2 and G_3 are enclosed in a circle with center G_1 . Note that $G_i \in \text{KNN}(G_j)$ does not necessarily imply $G_i \in \text{RKNN}(G_j)$: $G_1 \in 2\text{NN}(G_4) = \{G_1, G_3\}$, but $G_1 \notin \text{R2NN}(G_4) = \{G_3\}$. In addition, for any given query G with $k \geq 1$, $\text{KNN}(G) \neq \emptyset$; while the set of reverse k -nearest neighbors $\text{RKNN}(G)$ might be empty. As shown in Figure 4.1(b), $2\text{NN}(G_9) = \{G_1, G_7\}$, but $\text{R2NN}(G_9) = \emptyset$.

From an abstract point of view, given a test graph, in the KNN classifier, the algorithm finds the k most similar graphs in the training graph set. Then the weighted-based majority vote by the k most similar graphs, decides the classification. A RKNN classifier is built in a similar way except that the weighted-based majority vote is done by the reverse k nearest neighbors.

The details are shown in Algorithm 4.2. The feature vector representation of the query graph x' and sets of training feature vectors from each class y are fed to the classifier. Next, for x' , we perform either KNN or RKNN search to find a set of closest feature vectors and their class labels in the training set, denoted as $\mathcal{T}_{nn}$ (Lines 1–3). After that, a distance weighted score is calculated for classification (Lines 4–7). In Line 6, we use a very small number ξ as a smoothing factor to avoid unbalance when the distance $\text{dist}(x', t[2])$ is equal to 0. Note that for RKNN classifier, $\mathcal{T}_{nn}$ may be empty (Line 3). In this sense, the classification performance is the same with a random guess.

Complexity. The most time consuming part of Algorithm 4.2 is the KNN and RKNN search procedure. In our KNN classifier, we just use the naive linear search with the

CHAPTER 4. ADAPTIVE SYSTEM FOR GRAPH REACHABILITY INDEX SELECTION

time complexity $O(|\mathcal{T}||\mathcal{F}_d|)$ [Webe 98]. The complexity of RKNN search in the best case is $\Omega(r|\mathcal{T}||\mathcal{F}_d|)$, where r is the average number of results of the RKNN search; in the worst case, the time complexity of RKNN search is $O(|\mathcal{T}|^2|\mathcal{F}_d|^2)$ [Lin 08].

Error Stability for KNN. As the amount of data approaches infinity, the algorithm is guaranteed to yield an error rate no worse than twice the Bayes error rate (the minimum achievable error rate given the distribution of the data) [Cove 67]. In addition, the KNN classifier is guaranteed to approach the Bayes error rate, for some large value of k .

Error Stability for RKNN. In the following, we present a theoretical analysis to show that the error rate of our approach is close to that of Bayes error. Our argument is structured as follows. We assume that x' belongs to cluster C^* with label y^* , we bound the probability that $y' \neq y^*$ where x' is assigned to C with label y' by RKNN search.

Lemma 4.1 *Given a true cluster C^* for graph G represented as x , the probability that RKNN returns a cluster assignment C for G with $C \neq C^*$ is bounded by $\exp\{-\frac{2\delta^2}{L|\mathcal{F}_d|}\}$, where $|\mathcal{F}_d|$ is the dimension of discriminative features, and $0 < \delta < 1$.*

Proof (Sketch): Let us consider the following condition: $\forall q \in C^*, \text{dist}(q, x) \geq \text{nndist}_k(q)$, where $\text{nndist}_k(q)$ denotes the distance from q to its k -th nearest neighbor. Under this condition, $\forall q' \in \text{RKNN}(x, \mathcal{G}), q' \notin C^*$. Hence, x must be assigned to a cluster that is different from C^* . This indicates that the probability that RKNN returns a cluster assignment C for G with $C \neq C^*$ is bounded by $p(\text{dist}(q, x) \geq \text{nndist}_k(q) \mid q \in C^*)$, which is computed as follows. Consider the hoeffding's inequality that provides an upper bound on the probability for the sum of random variables to deviate from its expected value: Let $X_1, \dots, X_n$ be independent random variables. Assume that the X_i are almost surely bounded; that is, assume for $1 \leq i \leq n$ that $P(X_i \in [a_i, b_i]) = 1$, then we have the hoeffding's inequality $P(\bar{X} - E[\bar{X}] \geq \delta) \leq \exp\{-\frac{2\delta^2 n^2}{\sum_{i=1}^n (b_i - a_i)^2}\}$. Since q is randomly selected from C^* , here we define X as the sequence of $\text{dist}(q, x)$ whose absolute value is bounded by $\sqrt{|\mathcal{F}_d|}$ and expected value is $\text{nndist}_k(q)$. Therefore, by using hoeffding's inequality, $p(\text{dist}(q, x) - \text{nndist}_k(q) > \delta \mid q \in C^*) \leq \exp\{-\frac{2\delta^2}{L|\mathcal{F}_d|}\}$. Hence, Lemma 4.1 holds. $\square$.

Theorem 4.1 *Let ϵ be Bayes error, then the prediction error of the RKNN classifier is bounded by $\epsilon(1 - \exp\{-\frac{2\delta^2}{L|\mathcal{F}_d|}\}) + \exp\{-\frac{2\delta^2}{L|\mathcal{F}_d|}\}$, where $0 < \delta < 1$.*

Proof (Sketch): The proof to Theorem 4.1 can be easily derived on the basis of Lemma 4.1. Let p_2 be the probability that RKNN returns an incorrect cluster, of which the bound is given in Lemma 4.1. Then the predict error of the RKNN classifier, is $1 - (1 - \epsilon)(1 - p_2) = \epsilon(1 - p_2) + p_2$. The proof completes by replacing p_2 using Lemma 4.1. $\square$

4.3.2 Classification using Spectral Graph Partitioning/Clustering

Though the techniques behind KNN and RKNN classifiers are quite simple, they do achieve good performance when sufficient training graphs are available. However, unlike index selection in relational database which has been a mature technique, the graph reachability index selection problem is relatively new and hence only a few number of labeled data can be used for training. Therefore, we also design another classifier based on spectral graph partitioning, which has relatively better performance when the size of training data is inefficient.

Recall in Chapter 2.4.2, we have briefly introduced the concept of graph spectral partitioning. Then in the following, given a set of training graphs $\mathcal{G}$ which are represented as $T = \{ \langle x_i, y_i \rangle \}_{i=1}^L$, we consider the methods that make use of graph spectral partitioning to predict the label (i.e, the best reachability index) of the test graphs $\mathcal{G}'$:

1. Construct an abstract graph G_a , where each node v_a represents a graph $G \in (\mathcal{G} \cup \mathcal{G}')$, and edges are given by a symmetric weight matrix W which denotes the feature space distances among graphs. The i^{th} node in the graph has a label y_i .
2. Partition the set of nodes V_a into clusters $P_1, P_2, \dots, P_k$ using graph spectral partitioning.
3. Label all nodes in each cluster with the majority label for that cluster.

Complexity. Clearly, the complexity of this classifier is dominated by the procedure of spectral partition, which is bounded by $O(kL^2)$, where k is the number of partitions in spectral partitioning of graph G_a . The setting of value k is 6 in our experiments.

Error Stability. The error rate ϵ of the spectral bisection classification under the perturbation satisfies that $\epsilon \leq E ||\tilde{v}_2 - v_2||^2$ [Huan 08], where $\tilde{v}_2$ and v_2 are the unit-length second eigenvectors of the Laplacian Matric of graph G_a and the perturbation Laplacian Matric.

4.4 Experimental Evaluations

4.4.1 Dataset and Evaluation Metrics

In this subsection, we present an experimental study to verify the efficiency and effectiveness of our proposed methods. In our experiment, we chose Interval approach [Agra 89],

CHAPTER 4. ADAPTIVE SYSTEM FOR GRAPH REACHABILITY INDEX SELECTION

Table 4.1: Statistics of Graph Data.

Class	# of graphs	# of real graphs	average $ V $	average $ E $	average d_g
DUAL	139	22	11973	48481	0.138874
INTERVAL	84	10	19996	106001	0.009191
HOPI	77	8	4886	16575	0.155612

HOPI [Chen 06], and Dual labeling [Wang 06] as the set of candidate reachability indexing.

We used both of real data and synthetic data in our experiments. For the real data, we used a set of real graphs obtained from a Web graph repository¹ and the experiment sections of papers [Jin 08, Univ, Wang 06]. In addition, we generated a set of synthetic graphs with two different graph generators. The first generator² controls the graph properties with four input parameters: the maximum fan-out of a spanning tree, the depth of a spanning tree, the percent of non-tree edges and number of cycles. The second generator³ generates a scale-free power-law graph, whose degree distribution follows a power-law function. The total number of graphs used is 300. For each graph, we manually label it with the best reachability index as follows: we pre-build each index on the graph and compare the total query time of 1 million random queries. The best index is the one such that the total cost of running the queries is minimal. The statistics of graphs we used are summarized in Table 4.1.

To measure the performance of our framework, we used the prediction accuracy A and micro precision P to measure the performance of our framework. The prediction accuracy A , is defined as $A=T_i/n$, where T_i is number of samples with right classification, and n is number of samples in total. precision P is defined as $(\sum_{i=1}^{\gamma} \frac{TP_i}{TP_i+FP_i})/\gamma$, where γ is number of class labels, TP is the true positive, and FP is the false positive.

4.4.2 Results

Table 4.2: Running Time and Memory Usage of Feature Selection.

Running time (seconds)		Memory usage (MB)	
Metric computation	Feature selection	Metric computation	feature selection
50.373	24.035	89.83	3.8

¹<http://vlado.fmf.uni-lj.si/pub/networks/data/>

²<http://www.cais.ntu.edu.sg/~linhong/gen.rar>

³<http://www.cais.ntu.edu.sg/~linhong/GTgraph.zip>

CHAPTER 4. ADAPTIVE SYSTEM FOR GRAPH REACHABILITY INDEX SELECTION

Figure 4.2: Performance Comparison with Equally-splitting between Training and Testing Data.

First, we report the total running time and memory usage of two stages in discriminative feature selection: I) computing graph metrics (See Sec 4.2.1), and II) discriminative feature selection (See Algorithm 4.1). The results are shown in Table 4.2. It is observed that the memory usage is dominated by Stage I, i.e, the storage consumption to keep the whole graph in the memory. Interestingly, running time is also mainly taken up by the first stage. The reason is that we need to compute 15 metrics for each graph in the first stage and each metric computation needs to be done at least in linear time to the graph size.

Next, we conduct classification performance comparison over different classifiers. First, we compare the efficiency of different classifiers. We conduct a set of experiments with equally-splitting between training and testing data, but vary the total sample used from 40 to 200.

The total running time and memory usage of spectral-clustering, KNN and RKNN, are presented in Figure 4.2. The result shows that KNN is faster than RKNN and much faster than spectral-clustering. In addition, though KNN approach is faster than RKNN, the margin is quite small. Figure 4.2.b reports the memory usage (excluding the memory consumption to keep graphs) comparison of three approaches. It indicates that each of the three classifiers is comparable to others in the memory consumption.

Figure 4.3 gives the prediction accuracy and precision when the total number of graphs is varied. The result shows that when there are sufficient training data, the prediction accuracy and precision of spectral clustering $\gg$ RKNN $\gg$ KNN where $\gg$ denotes significantly better with p -value < 0.05 . Moreover, based on Figure 4.3, we can draw a

CHAPTER 4. ADAPTIVE SYSTEM FOR GRAPH REACHABILITY INDEX SELECTION

Figure 4.3: Prediction Accuracy and Precision Comparison with Equally-splitting between Training and Testing Data.

Figure 4.4: Prediction Accuracy and Precision Comparison varying Number of Training Graphs.

conclusion that graph classification using theoretical metric is feasible since the accuracy of all the three classifiers is much higher than the baseline which labels all the graph with class label “DUAL” (46.3%).

Another factor which might influence the overall performance is the size of the training data (i.e., the sufficiency of training data). Therefore, we also present corresponding results of accuracy comparison when varying the number of training documents. Specifically, we use r_t to denote the ratio between the number of training graphs and the total number of graphs and vary it from 10% to 50%. If $r_t=50\%$, it is equal to the above setting with equally-splitting between training and testing data. When $r_t < 50\%$, we are studying the effect of insufficient training data over different approaches. The results are reported in Figure 4.4. From those numbers, in average, spectral clustering obtains substantial improvement than KNN and RKNN no matter whether training graphs are suf-

CHAPTER 4. ADAPTIVE SYSTEM FOR GRAPH REACHABILITY INDEX SELECTION

---

ficient or not. When training graphs are noticeably sufficient, RKNN classifier could get classification of good quality; when training graphs are relatively insufficient, the performance of RKNN drops off sharply. The reason is that the probability of the RKNN search returns an empty set is higher when the training graphs is insufficient. In addition, the weighted major vote on an empty set would trigger a random guess (see Algorithm 4.2), which deteriorates the performance of RKNN. We attain similar results for precision of these three classifiers.

4.5 Summary

In this chapter, we formalized the problem of selecting the best graph reachability index for a particular application as a clustering/classification problem, where each class label is a type of indexing technique. Upon this formalization, we studied the feasibility of graph classification using a number of graph theatrical metrics. We also proposed a greedy algorithm to select a set of discriminative metrics. Experimental results showed that our framework works good in terms of both accuracy and precision, which verified the possibility of graph reachability index selection using graph metrics.

Chapter 5

Structure and Attribution Index-based Approximate Graph Matching over Large Graphs

5.1 Introduction

Data and their relationship in various domains such as the Semantic Web, images, videos, social networks, bioinformatics, and so on, exhibit graph-like structures. In recent years, the accelerated growth of such data and their applications have sparked renewed interests in developing approximate (inexact) graph matching techniques in large graphs. Approximate matching is an important extension to the exact graph matching, as to allow more flexibility and cater for the need of many applications where exact information is often not available. Approximate graph matching has a wide range of applications. For example, in biology, approximate graph query is used to search for protein complex¹ in a large protein-protein interaction network [Tian 07]. Another example is terrorist modus operandi detection [Marc 02], which is a form of searching for a query graph (a threat pattern) in a graph that stores a large volume of observed threat activities data.

We give an example of approximate graph matching as follows:

Example 5.1 *Figure 5.1 shows a sample data graph and query graph. The data graph shows the relationship among scholars from different research areas. Suppose that we want to find Alice in machine learning and her connection to other scholars in different research areas, we can create an abstract query graph as shown in Figure 5.1.a. Clearly,*

---

¹Usually a protein complex is a dense subgraph whose nodes (proteins) perform similar biological functions.

CHAPTER 5. STRUCTURE AND ATTRIBUTION INDEX-BASED APPROXIMATE GRAPH MATCHING OVER LARGE GRAPHS

Figure 5.1: An Illustrative Example.

exact matching returns no answer to this query (indeed, Alice may not be connected to scholars in every area). On the contrary, with approximate matching, the induced subgraph by $\{2, 6, 7, 8\}$ is a good answer.

There are several challenges in tackling the approximate graph matching problem. First, a query graph may not contain the details of every node, since users may not know all the details (often that is why the query is asked). For example, in Figure 5.1.a, only node a carries both label information (i.e., the name “Alice”) and attribution (i.e., the area “machine learning”), while other query nodes contain only attribution information. Second, an edge in the query may not necessarily have a match in the data graph. For example, the edge (a, c) in Figure 5.1.a does not match any edge in Figure 5.1.b. In this case, we want to find the best path that connects the corresponding nodes of a and c in the data graph (e.g., the path 6-7-2 in Figure 5.1.b). Third, we cannot simply find the best matching edge/path for an individual edge in the query graph, but need to consider the overall query graph that is to be matched, which is a difficult optimization problem.

In this chapter, we model the approximate graph matching problem as an optimization problem. We prove that the problem is NP-hard and propose an index-based greedy algorithm that are both effective and efficient. Our proposed method consists of three phases:

Indexing In the first phase, we develop an index that is both structure-aware and attribution-aware, namely *SA-Index*. At the first level of *SA-Index*, we index on the structure similarity and attribute similarity of the data graph, which partitions the

---

CHAPTER 5. STRUCTURE AND ATTRIBUTION INDEX-BASED APPROXIMATE GRAPH MATCHING OVER LARGE GRAPHS

---

data graph into clusters of nodes that are close to one another both in structure and in attribute values. Then at the second level, we further encode the connectivity among the clusters in a hash table.

Best path computation In the second phase, we define the set of best paths² in the data graph for matching each edge in the query graph. Then, we utilize SA-Index to efficiently compute this set of best paths.

Partial answer connection discovery In the third phase, we propose two greedy algorithms to compute the best matching answer graph, by selecting the best paths computed from the second phase.

We evaluate the performance of our method using a set of large real datasets. Our experimental results show that SA-Index is efficient to construct, and query performance using the index is efficient and scalable to different sizes of data graphs and query graphs. Our method is over an order of magnitude faster than G-ray[Tong 07] and be more specific faster than the memory-based implementation of TALE [Tian 08b]³. We also use various quality measures to show that the answer graphs obtained by our method are of high quality by case studies.

Chapter Organization. The rest of this chapter is organized as follows: We give a formal definition of approximate graph matching problem in Section 5.2. Sections 5.3, 5.4.1, and 5.4.2 describe the three phases of our solution, respectively. We report experimental results in Section 5.5. Finally, we conclude our work in Section 5.6.

5.2 Problem Definition and Computational Hardness

5.2.1 Problem Definition

In this subsection, we present our problem formulation of approximate graph matching query as follows.

Definition 5.1 (Approximate Graph Matching) *Let G and Q be the data graph and query graph, respectively. An approximate matching from Q to G , denoted as $G \approx Q$, is defined as a bijection $f: \hat{V}(Q) \subseteq V(Q) \leftrightarrow \hat{V}(G) \subseteq V(G)$ s.t $\forall u \in \hat{V}(Q)$, $f(u) \in \hat{V}(G)$, $A^Q(u) = A^G(f(u))$, and $(l^Q(u) = l^G(f(u))$ or $l^Q(u) = \lambda)$, and $\forall u, v \in \hat{V}(Q)$, where $(u, v) \in E(Q)$, there is a path from $f(u)$ to $f(v)$ in G .*

CHAPTER 5. STRUCTURE AND ATTRIBUTION INDEX-BASED APPROXIMATE GRAPH MATCHING OVER LARGE GRAPHS

Figure 5.2: Examples of Approximate Graph Matching

Example 5.2 In the examples shown in Figure 5.2, the query graph is matched to two subgraph G_1 and G_2 of data graph G in Figure 5.1.b. The dashed lines indicate the matched nodes in query graph and data graph. Note that not all nodes are required to be mapped, for instance, as shown in Figure 5.2.b, node c in query Q has no mapping in G_2 . Also note that an edge in the query graph could be mapped to a path in the data graph, e.g., in Figure 5.2.a, the edge $a - c$ in the query Q is matched to the path $6 - 7 - 2$ in G_1 .

We now define the problem of processing an approximate graph query in an attributed graph as follows.

Problem 5.1 (Approximate Graph Matching Problem)

Given a query Q and a graph G , the problem of Approximate Graph Matching (AGM) is to find the best matching answer graph G_a as defined by the following equation:

$$G_a = \arg \max_{G'} \text{sim}(Q, G'), \quad (5.1)$$

where $G' \subseteq G$ and $G' \approx Q$, and $\text{sim}(Q, G')$ is the degree of matching from Q to G' . If $\text{sim}(Q, G')=1$, then Q is graph isomorphic to G' .

The degree of matching between two graphs can be measured by various metrics. For example, Montes-y-Gómez *et al.* [Gome 00] defined graph similarity as the number

²Note that an edge can also be regarded as a path.

³We changed the disk-based implementation of TALE (i.e., going through PostgreSQL) provided by authors to memory-based implementation for fair comparison.

CHAPTER 5. STRUCTURE AND ATTRIBUTION INDEX-BASED APPROXIMATE GRAPH MATCHING OVER LARGE GRAPHS

of common edges shared by two graphs. The term “graph edit distance” covers a class of metrics that measure the degree of matching between two graphs. Its variants have been studied theoretically [Bunk 97] as well as applied in domains as diverse as image retrieval [Shap 81] and querying graph database [He 06]. The choice of different metrics for the degree of matching is orthogonal to this work. In the following sections, we propose an approximate graph query algorithm that takes a general metric as input. Furthermore, we would discuss briefly how our work supports the two metrics “maximum common edges” and “graph edit distance” in Appendix A.

5.2.2 Hardness of AGM Problem

We analyze the hardness of simplified version of our AGM problem, i.e., the decision problem of AGM. A simple reduction from the γ -quasi clique problem established the intractability of that problem.

Theorem 5.1 (NP-Hardness) *Given a query Q , a graph G , and a value $0 < \delta \leq 1$, the decision problem of AGM, i.e., to determine whether there is a subgraph G_a of G such that $\text{sim}(Q, G_a) \geq \delta$ is NP-hard.*

Table 5.1: Examples of Degree of Matching Metrics and Relationship between γ and δ .

metric	common edge [Gome 00]	edit distance [He 06]	our experiment
relation	$\gamma = \delta$	$\gamma = \delta$	$\gamma = \min(\frac{1}{\alpha}\delta, 1)$

Figure 5.3: An Example of Reducing From γ -Quasi Clique to Decision Problem of Approximate Graph Matching.

CHAPTER 5. STRUCTURE AND ATTRIBUTION INDEX-BASED APPROXIMATE GRAPH MATCHING OVER LARGE GRAPHS

Proof (Sketch): We show the NP-hardness of the decision problem of AGM by reducing the problem of γ -quasi clique to our problem. The problem of γ -quasi clique is defined as follows: Given a graph G , an integer K , and $0 < \gamma \leq 1$, is there a subgraph $\tilde{G}=(V(\tilde{G}), E(\tilde{G}))$ of G with K or more nodes such that $\frac{2|E(\tilde{G})|}{|V(\tilde{G})|(|V(\tilde{G})|-1)} \geq \gamma$ (if $\gamma=1$, then $\tilde{G}$ is a clique). The problem of γ -quasi clique with $0 < \gamma \leq 0.5$ and $\gamma=1$ is shown to be NP-complete [Gare 79, Mats 99]. Although it is not proved whether the γ -quasi clique problem of $0.5 < \gamma < 1$ is NP-complete or not, [Mats 99] infer the γ -quasi clique problem is NP-complete. Now, given G , K and γ , build a complete graph Q such that Q has K nodes. Clearly G has a subgraph G_a such that $\text{sim}(Q, G_a) \geq \delta$ if and only if G has a γ -quasi clique. The relationship between γ and δ is determined by the type of degree of matching metric used. In Table 5.1, we give some examples of different types of degree of matching metric and the corresponding relationship between γ and δ . Thus, the reduction consists of constructing a complete graph with K nodes, which can clearly be done by using logarithmic auxiliary space.

In conclusion, the decision problem of AGM is NP-hard. $\square$

Example 5.3 Consider the graph G shown in Figure 5.3. Suppose that $\gamma=\frac{2}{3}$ and $K=4$. Now we build a complete graph Q with 4 nodes. Assume that the input degree of matching metric uses number of common edges [Gome 00]. Then the subgraph G' induced by the node set $\{2, 3, 4, 5\}$ approximately matches the query Q with $\text{sim}(Q, G')=\frac{2}{3}$ (Number of common edges is 4 and the total number of query edges is 6). Hence, G' is a “yes” instance to approximate graph matching with $\delta=\frac{2}{3}$. This indicates the subgraph G' induced by the node set $\{2, 3, 4, 5\}$ is also a “yes” instance to γ -quasi clique with $\gamma=\frac{2}{3}$.

Since the decision problem of AGM is NP-hard, its optimization problem—our AGM problem is also NP-hard. Thus, for finding an optimal approximate graph matching to a given query, we need to develop some heuristic approaches in the following sections.

5.3 SA-Index

In this subsection, first we propose to use a novel *SA-Index* (Structure- and Attribute-aware Index) to summarize the connectivity, structural similarity and attribute similarity information of a data graph. SA-Index consists of two-levels: partition-based index and connectivity-based index. Next, in Section 5.3.2, we analysis the advantages of SA-Index. Finally, we discuss the efficient computation of SA-Index, see Section 5.3.3.

5.3.1 The Notion of SA-Index

Partition-based Index. Intuitively, at the first level of SA-Index, we partition the data graph into subgraphs based on its structure similarity (i.e., nodes that are highly connected with one another form a group) and attribute similarity (i.e., nodes that are closely related to one another in attribute values are condensed together), denoted as *partition-based index*.

In order to utilize the structure similarity, we introduce a notion of “conductance”. In graph theory the conductance of a graph G measures how “well-knit” the graph is: it controls how fast a random walk on G converges to a uniform distribution [Boll 98]. In this chapter, we follow the definition given by Leskovec *et al.* [Lesk 08]. Its generalized version would form the core of our first level of SA-Index.

Definition 5.2 (Conductance)[Lesk 08] *Given a graph G represented as an n -by- n matrix M , the conductance of a set of nodes S is defined as*

$$\phi(S) = \frac{\sum_{u \in S, v \notin S} M_{uv}}{\sum_{u \in S} d(u)} \quad (5.2)$$

where $d(u)$ is the degree of the node u in G .

It is commonly noticed that conductance captures the “gestalt” notion of a community. A group of nodes S is a good community if it is separated from the remaining of the graph by a low-conductance cut. In particular, the use of conductance as an objective function returns a good partitioning that facilitates connectivity discovery in on-line query since optimizing conductance function also maximizes intra-partition connections and minimizes inter-partition connections.

However, utilizing a single metric such as conductance has a drawback: only structure information is adopted to measure the goodness of a group of nodes S ; other information such as attribution information is ignored. In parallel to structure information, attribution information has recently been rediscovered in graph clustering/partitioning: people may seek to find a good community based on attribute similarity. Attributes also make a more targeted search in a large data graph.

To address the aforementioned issues, now we introduce a concept of *homogeneity*, to capture the intuition of what it means for a set of nodes to be attribute similar. Essentially, given a graph, homogeneity of a node v measures how many nodes in the neighborhood of v share the same attribute values. As a simple example, a group of

CHAPTER 5. STRUCTURE AND ATTRIBUTION INDEX-BASED APPROXIMATE GRAPH MATCHING OVER LARGE GRAPHS

5.4.a: Structure-based Partition. 5.4.b: Attribute-based Partition. 5.4.c: SA Partition.

Figure 5.4: A Running Example.

nodes that all have the value “database” for their “research area” attribute would be considered very homogeneous; while a group of nodes each with a different value for the “research area” attribute would be considered very heterogeneous.

Mathematically, given a set of nodes $S \subseteq V(G)$, the homogeneity of a node $v \in S$, $h(v, S)$, is defined as:

$$h(v, S) = - \sum_{i \in A_v^S} p(i | v) \log p(i | v) \quad (5.3)$$

where $A_v^S = \cup_{u \in S} M_{uv} A^G(u)$ is the set of attribute values in the neighborhood of v , $p(i | v) = \frac{\sum_{u \in S} M_{uv} \mathcal{I}(A^G(u)=i)}{d(v)}$ denotes the fraction of nodes in the neighborhood of v that have attribute value i , and $\mathcal{I}(\cdot)$ is an indicate function which returns one if the expression is true, and zero otherwise. Clearly, a node v with higher value of $h(v, S)$ is quite heterogeneous.

In the following, we explore a strong link between conductance and homogeneity to collapse nodes which are similar in terms of both structure and attribute into a group. Consider a running example in Figure 5.4 which shows several possible partitionings. Figure 5.4.a shows one partitioning according to Equation 5.2: $\{1, 2, 3, 4, 5\}$ and $\{6, 7, 8, 9, 10, 11\}$. People in one group are highly connected with one another, which leads to a small conductance. However, people inside the same partition may have diverse research interests: in the group $\{6, 7, 8, 9, 10, 11\}$, research interests of people distribute equally over “data mining”, “information retrieval” and “machine learning”. According to Equation 5.3, another attribute-based partitioning is shown in Figure 5.4.b. People work in the same research area are grouped together such as $\{1, 2, 3, 4, 5\}$, $\{7, 9\}$, $\{8, 10\}$ and $\{6, 11\}$. Unfortunately, using such a way to partition make nodes in the same group quite isolated and nodes in different groups highly connected. Therefore, users may expect a partitioning that weighs both research interests and interactivities, as shown in Figure 5.4.c.

CHAPTER 5. STRUCTURE AND ATTRIBUTION INDEX-BASED APPROXIMATE GRAPH MATCHING OVER LARGE GRAPHS

As a start, a naive approach is to perform a hierarchical partitioning. That is, one may perform partitioning based on one dimension (i.e., a single conductance or homogeneity metric), and then refine the partitioning based on another dimension of information. This approach raises such a problem: Shall we perform structure-based partitioning first and then attribute-based partitioning? Does the order of hierarchical partitioning affect the final results? If the answer is yes, how to determine the order becomes another challenging problem. Besides, an assumption behind this approach is that structure information is independent with attribute information, which is not the truth in most cases.

Hence, instead of using a hierarchical partitioning, we make use of both structure and attribute together to perform partitioning. A nature idea of our approach is that a set of nodes S is a good community if splitting S into a group results in lower conductance and higher homogeneity (lower values of $h(v, S)$ for $v \in S$). Therefore, we embed the homogeneity property into conductance, and propose a generalized conductance $\varphi(S)$ for a set of nodes S as follows.

$$\varphi(S) = \frac{\sum_{u \in S, v \notin S} A_{uv} h(u, S)}{\sum_{u \in S} d(u) h(u, V(G))} \quad (5.4)$$

where the nominator indicates the conductance and homogeneity after condensing S into a group; and the denominator indicates the conductance and homogeneity before separating S from a graph G . A set of nodes S is a good community if S has a small value of $\varphi(S)$.

With Equation 5.4, the first level of SA-Index of a data graph G , partition-based index $P^*(G)$, is defined as: $P^*(G) = \arg \min_{P(G)} \{\sum_{V_k \in P(G)} \varphi(V_k)\}$, where $P(G) = \{V_1, V_2, \dots, V_k\}$ denotes any partitioning over G such that $(\bigcup_{i=1}^k V_i) = V(G)$ and $V_i \cap V_j = \emptyset$, where $i \neq j$ and $1 \leq i, j \leq k$.

Connectivity-based Index. So far we have formally introduced the first level of SA-Index. Now, given the data graph G and $P^*(G)$, we further present the details of the second level of SA-Index, which provides a path-view over connectivity among sub-partitions in $P^*(G)$, denoted as *connectivity-based index*. More specifically, we introduce a summary graph on the basis of a fundamental terminology “boundary nodes”, to keep the inter-partition information as a complementary to $P^*(G)$.

Definition 5.3 (Boundary Nodes and Summary Graph) *Given a group of nodes V_i , let $\chi(V_i)$ denote the set of nodes in V_i that are connected to the nodes outside V_i . Then, $\chi(V_i)$ is called boundary nodes set of V_i . Given a partitioning $P(G)$ over the data graph G , its boundary node set, is defined as: $\chi(P(G)) = \bigcup_{V_i \in P(G)} \{\chi(V_i)\}$. Then the summary graph G_s is a subgraph of G induced by the node set $\chi(P(G))$.*

CHAPTER 5. STRUCTURE AND ATTRIBUTION INDEX-BASED APPROXIMATE GRAPH MATCHING OVER LARGE GRAPHS

Figure 5.5: An Example of the SA-Index.

The summary graph G_s , provides a lossless information of direct connections via edges among boundary nodes. In addition, G_s is unconnected if there exists a pair of boundary nodes that are in the same partition and are not connected via edges. However, using G_s only may still suffer low speed-up since we need to scan G_s multiple times to find those connections among boundary nodes in the on-line query phase. To deal with it, we pre-compute some partial path information to prune the search space in on-line query. Specifically, we utilize the fundamental data structure *hash table* to provide a path view over inter-partition connections. Thus, both the summary graph G_s and the hash table HT together comprise the connectivity-based index. In the hash table HT , the hash key of each tuple is a pair of partition *IDs* of boundary nodes and the hash value consists of the set of nodes (attribute values are also included) in the shortest path connecting two partitions where the pair of boundary nodes are located.

In all, the SA-Index proposed in this work is a three-tuple relation $\langle P^*(G), G_s, HT \rangle$, where $P^*(G)$ is a partition-based index, G_s is a summary graph, and HT is a hash table that stores partial path information among sub-partitions. We use the following example to further illustrate the concept of SA-Index.

Example 5.4 As shown in Figure 5.5, based on the structure information and homogeneous property, we partition the data graph in Figure 5.1.b into three parts: $P(G) = \{\{1, 2, 3, 4, 5\}, \{6, 7, 9, 11\}, \{8, 10\}\}$. Next, we compute the boundary node set $\chi(G)$. In this example, $\chi(G) = \{2, 4, 6, 7, 8, 10\}$. The summary graph, induced by the boundary node set, is shown in Figure 5.5(b). Finally, we encode the connectivity information among partitions in a hash table, which is shown in Figure 5.5(c).

5.3.2 Why SA-Index?

Why Partition-Based Index At the first level, we partition a data graph based on its structure and attribute. As we know, the partitioning of a given data graph is not unique. Hence, a problem need to address is: why we need a partitioning $P^*(G)$ rather than other partitionings?

Lots of partitioning algorithms have been proposed such as METIS [Kary 98], edge betweenness [Kern 70], Spectral Clustering [Gupt 97a, Poth 90], conductance [Lesk 08], attribution similarity [Tian 08a], and modularity [Clau 04]. In the following, we illustrate the reason of using $P^*(G)$ by identifying restrictions of other approaches.

First, all of the above partitioning works group nodes either based on structure only or based on attribute only. Consider corresponding examples from other areas, one often makes use of both structure and attribute: In the area of image analysis and compute vision, one may adopt the joint learning of object parts and attributes to perform object detection; in the area of data clustering, SAclustering [Zhou 09], also proposes a structure-based and attribute-based approach to improve the quality of clustering. However, in graph partitioning, a strong link between structure and attribute has not been previously explored. Furthermore, during query processing, both attribute equality and structure similarity are of great importance. Therefore, a good partitioning also needs to take both structure and attribute into consideration.

Next, though in SAclustering [Zhou 09], nodes in one cluster are close to one another either in structure or in attribute, the inter-cluster information, may become too complex to be encoded concisely. This may incur additional processing time since in query processing we also need to access inter-partition nodes/edges to discover connectivity among partial answers in different partitions.

As an alternative, with the $P^*(G)$, we could quickly identify the set of partitions (subgraphs) that contain candidate nodes for answer graphs on the basis of attribution equality and structure similarity. In addition, partition-based index prunes the search space for partial answers from the entire data graph to a set of subgraphs. Finally, as $P^*(G)$ takes the generalized conductance as the objective function, it can also produce a low cut among different partitions and reduce the encoding cost of the second-level index.

Why Need G_s and HT It is straightforward that the connectivity among nodes in different partitions is important for query processing. One may simply choose the node

---

CHAPTER 5. STRUCTURE AND ATTRIBUTION INDEX-BASED APPROXIMATE GRAPH MATCHING OVER LARGE GRAPHS

---

or edge as unit of connectivity-index, i.e., index the set of inter-partition nodes/edges. This method has the benefit that the index size grows linearly with the number of nodes in the data graph, but suffers from low speed-up for query processing since one may still be required to find connections among inter-partition edges/nodes in on-line query processing.

In response to the above issues, we use the *graph-theoretic distance* to capture the connectivity among inter-partitions and encode a set of paths into a hash table (denoted as HT). One can identify connections among partitions quickly with the use of HT . In addition, as the number of boundary nodes is not large in reality, the size of connectivity-based index ($|HT|$ and $|G_s|$), is still affordable even if storage cost is considered as an important issue. Last but not the least, HT can be used to prune search space during the procedure of discovering connections among candidate answer nodes.

5.3.3 Efficient Index Construction

In this subsection, we study efficient approaches to compute the SA-Index. We first introduce a heuristic algorithm to fast compute a partitioning of a data graph with respect to the definition of $P^*(G)$. Next, we present the procedure of summary graph and hash table computation.

Our heuristic partitioning algorithm follows the traditional “extended local search” paradigm: a node could be swapped between two partitions again and again to correct previous bad assignments. In another word, in each iteration, we search for a better assignment that reduces the value of Equation 5.4. The details are outlined in Algorithm 5.1. Initially, we just randomly bisect graph G into two partitions (Line 1). Next, $\text{partition}(G)$ iteratively searches a locally best assignment for each node. In other words, in each step, for each node v , we enumerate all possible new partitionings by assigning v from its original partition i to another existing partition j or a new empty partition (Lines 3–7). Note that in Line 5 we insert an empty node set $V_\emptyset$ into the current partition $P(G)$, so that at each iteration we also allow v to form a partition by itself.

Then in Line 8, among those new partitionings, we find the best target partition τ , such that the new partitioning produced by assigning v to partition τ obtains the minimal size of generalized conductance among all other partitionings produced by other assignments. If the value of generalized conductance of the new partitioning $P^\tau(G)$ (as defined by Equation 5.4) is smaller than that of the current partitioning $P(G)$, we replace $P(G)$ by $P^\tau(G)$ and repeat the process (Lines 9–11). If $P(G)$ is ever replaced in

CHAPTER 5. STRUCTURE AND ATTRIBUTION INDEX-BASED APPROXIMATE GRAPH MATCHING OVER LARGE GRAPHS

Algorithm 5.1: Partition-Based Index Construction `partition(G)`

Input: A graph G
Output: a partitioning $P(G)$

```

1:  $P(G) = \{V_1, V_2\}$ ;
2:  $flag = \text{false}$ ;
3: for each  $V_i \in P(G)$ 
4:   for each node  $v \in V_i$ 
5:      $P(G) = P(G) \cup \{V_\emptyset\}$ , where  $V_\emptyset$  is an empty set;
6:     for each  $V_j \in P(G)$ , where  $j \neq i$ 
7:       Get  $P^j(G) = \{V_1^j, \dots, V_k^j\}$  by moving  $v$  to  $V_j$ ;
8:        $\tau = \arg \max_j \{ \sum_{V_k \in P(G)} \varphi(V_k) - \sum_{V_k^j \in P^j(G)} \varphi(V_k^j) \}$ ;
9:       if  $\sum_{V_k \in P(G)} \varphi(V_k) > \sum_{V_k^\tau \in P^\tau(G)} \varphi(V_k^\tau)$ 
10:         $P(G) = P^\tau(G)$ ;
11:         $flag = \text{true}$ ;
12: if  $flag == \text{true}$ 
13:   repeat Lines 2-11;
14: else
15:   return  $P(G)$ ;
```

---

the previous iteration, $flag$ is set to `true` in Line 11 and we start a new iteration of Lines 2–11. Otherwise, $P(G)$ has the local minimal value of generalized conductance and is returned (Lines 14–15).

Complexity Analysis. Assume i is the number of iterations, then the overall time complexity of our partitioning algorithm is $O(ik(n+m))$, where n and m are the number of nodes and edges in G , and k is the number of partitions in $P(G)$.

After investigating how to partition the data graph G based on its structure and attribute similarity, now we discuss the details of computing the connectivity-based index, which is a concise representation of inter-partition connections.

Given the partition-based index $P^*(G)$ and the data graph G , Algorithm 5.2 first computes the summary graph G_s (Line 1). As we are studying on undirected graphs, the set of shortest paths from u to v is equal to the set of shortest paths from v to u . Hence, instead of computing all pairs of shortest paths, we compute shortest paths of pairs of node (u, v) when $u \prec v$ (Lines 4–5). Next, we hash the set of paths into HT where each hash value is computed according to a set of path features (Lines 7–11).

Complexity Analysis. The time complexity of Algorithm 5.2 is dominated by line 5: the computation of shortest paths set. As we have known, a number of shortest paths that have the same source can be computed in $O(n \log n + m)$ [Corm 01], where n and

CHAPTER 5. STRUCTURE AND ATTRIBUTION INDEX-BASED APPROXIMATE GRAPH MATCHING OVER LARGE GRAPHS

Algorithm 5.2: Connectivity-Based Index Construction**Input:** A graph G and its partitioning $P^*(G)$ **Output:** a connectivity-based index $\langle G_s, HT \rangle$

- 1: compute G_s based on $P^*(G)$ and G ;
- 2: order the node set $V(G_s)$ as $v_1 \prec v_2 \dots \prec v_n$;
- 3: $H(G_s) = \emptyset$;
- 4: **for each** $(u, v) \in V(G_s) \times V(G_s)$, $u \prec v$
- 5:   compute the set of shortest path P_{uv} connecting u and v in G ;
- 6:   **for each** $p \in P_{uv}$
- 7:   compute the set of features of path p including path length, weighted path length, and attribute values of nodes;
- 8:   compute $\text{value}(p)$ based on the path features;
- 9: $\text{key}(p) = \text{make\_pair}(i, j)$, where $u \in V_i$ and $v \in V_j$
- 10: $T_h = \text{make\_pair}(\text{key}(p), \text{value}(p))$;
- 11:   hash T_h into HT ;
- 12: **return** HT and G_s ;

m are the number of nodes and edges in a given graph. Hence, in the worst case, the complexity of Algorithm 5.2 is $O(|V(G_s)| |V(G)| \log |V(G)| + |V(G_s)| |E(G)|)$.

5.4 On-line Query Processing

In this section, we propose an algorithm to process approximate graph matching queries using the SA-Index. In Section 5.4.1, we describe a method to find the set of best paths in the data graph for matching the set of query edges. Then in Section 5.4.2, we present two greedy algorithms to construct the best query answer from the set of best paths.

5.4.1 Computation of the Best Paths

In Definition 5.1, we allow the matching of an edge in the query graph with a path in the data graph, in the case when the query edge does not match any edge in the data graph. Although a path does not directly connect two nodes as does an edge, a good path can provide good insight of how the two nodes are connected. This is particularly useful when there is no edge in the data graph that matches a query edge. In fact, the user who specifies a query edge may not know the existence of such an edge (otherwise he probably will not ask the query), and the user may want to know the connection between the two nodes rather than a *strict* direct connection as by an edge.

CHAPTER 5. STRUCTURE AND ATTRIBUTION INDEX-BASED APPROXIMATE GRAPH MATCHING OVER LARGE GRAPHS

We further explain the concept by the following example.

Example 5.5 Consider the edge (a, c) in Figure 5.1.a which indicates the connection from a person Alice in machine learning area to another person in database area. However, in the corresponding data graph in Figure 5.1.b, there is no direct connection (i.e., an edge) from Alice to any DB person. Instead, Alice is connected to a DB person via a set of paths such as 6-7-2, 6-8-7-2, 6-7-2-4, and so on. These paths reveal to us how Alice is connected to a DB person.

As shown in Example 5.5, a given query edge may have a set of paths that can match it. Therefore, different edge-to-path matching may affect the quality of the answer graph returned. In the following, we discuss how we select the set of best paths for a given query edge.

Given a query edge e and two nodes x and y in a data graph G , where $A^G(x) = A^Q(u_e)$ and $A^G(y) = A^Q(v_e)$, the “best path” $p^* \triangleright G$ that matches e and connects x and y is defined as:

$$\begin{aligned} p^*(x, y, e) &= \arg \max_{p \triangleright G} \text{sim}(e, p) \\ &\text{subject to } u_p = x \text{ and } v_p = y, \end{aligned} \quad (5.5)$$

where sim^4 is the same input metric to measure the degree of matching from a query graph to a data graph.

The set of best paths matching query edges, is further defined as

$$P = \bigcup_{e \in E(Q)} \bigcup_{\substack{x, y \in V(G) \\ A^G(x) = A^Q(u_e) \\ A^G(y) = A^Q(v_e)}} p^*(x, y, e) \quad (5.6)$$

Now we discuss the details of computing P as given in Equation 5.6. In Algorithm 5.3, given a query edge (u, v) , the first step is to compute the set of nodes in data graph whose values of attributes are compatible to u and v (Lines 2–3). Next, we compute the partitions where x and y reside, i.e., G_i and G_j (Line 4). No matter whether x and y are in the same partition or not, it is required to compute the best path p^* that matches e by examining both inter- and intra-partition concepts. In order to compute those inter-partition connections, if G_s is connected, we only need to scan G_s for the best connections

---

⁴Here we consider a path or an edge as a graph.

CHAPTER 5. STRUCTURE AND ATTRIBUTION INDEX-BASED APPROXIMATE GRAPH MATCHING OVER LARGE GRAPHS

Algorithm 5.3: Finding the Best Paths $\text{edgetopath}(P^*(G), G_s, HT, Q, \text{sim})$

Input: Query Q , index $\langle P^*(G), G_s, HT \rangle$, and metric sim

Output: A path set, P

```

1:  $P = \emptyset$ ;
2: for each edge  $(u, v) \in E(Q)$ 
3:   for each  $x, y \in V(G)$ , where  $A^G(x) = A^Q(u)$  and  $A^G(y) = A^Q(v)$ 
4:     Let  $G_i$  ( $G_j$ ) be the partition that  $x$  ( $y$ ) belongs to;
     /* compute  $p^*$  where  $p^*$  is the solution to Equation 5.5 */
5:     Let  $S$  denote the search space;
6:     if  $G_s$  is connected
7:        $S = G_s$ ;
8:     else
9:        $S = G_s \cup HT$ ;
10:     $S = S \cup G_i \cup G_j$ ;
11:     $p^* = \arg \max_{p \in S} \{\text{sim}(e, p)\}$ ;
12:     $P = P \cup p^*$ ;
13: return  $P$ ;
```

---

since G_s has encoded connections via edges among boundary nodes; otherwise, we can augment the search space to $G_s \cup HT$ for best inter-partition connections (Lines 6–9). Finally, we scan all the possible combinations of both intra- (i.e., G_i and G_j) and inter- (G_s and HT) connections for the best path (Lines 10–11).

5.4.2 Partial Answer Connection Discovery

In this section, we compute the answer graph that matches the input query graph by greedily selecting the best paths that are computed by Algorithm 5.3. We design two types of greedy algorithms as follows.

The first greedy algorithm starts from an empty answer graph G_a^1 , and iteratively in step k , inserts into G_a^1 the path p_k that matches an unmatched query edge (u, v) , and that has common nodes with G_a^1 , and that maximizes the following equation.

$$\begin{aligned}
 p_k = \arg \max_{\substack{p \in P \\ p \not\in G_a^1}} \{ \text{sim}(Q, G_a^1 \cup \{p\}) - \text{sim}(Q, G_a^1) \} \\
 \text{subject to } (G_a^1 \cup \{p\}) \text{ is connected.}
 \end{aligned} \tag{5.7}$$

The algorithm stops when all the edges in the query graph are matched.

CHAPTER 5. STRUCTURE AND ATTRIBUTION INDEX-BASED APPROXIMATE GRAPH MATCHING OVER LARGE GRAPHS

Algorithm 5.4: Greedily select path $\text{greedy}(Q, P, \text{sim})$

Input: query Q , metric sim , a candidate partial answer set P

Output: An answer graph G_a

```

1: type=1,  $G_a^1 = \emptyset$ ;
2: while not all the query edges are processed
3:    $p_0 = \arg \max_{\substack{p \in P \\ p \not\in G_a^1}} \{\text{sim}(Q, G_a^1 \cup \{p\}) - \text{sim}(Q, G_a^1)\}, G_a^1 \cup \{p\} \text{ connected};$ 
4:    $G_a^1 = G_a^1 \cup \{p_0\}$ , mark an edge mapped processed;
5:    $\forall p \in P$ , if  $p$  and  $p_0$  match to the same query edge
6:      $P = P \setminus \{p\}$ ;
7: type=2,  $G_a^2 = \emptyset$ ;
8: while number of processed query edges  $< |E_q| - 1$ 
9:    $p_0 = \arg \max_{\substack{p \in P \\ p \not\in G_a^2}} \{\text{sim}(Q, G_a^2 \cup \{p\}) - \text{sim}(Q, G_a^2)\};$ 
10:   $G_a^2 = G_a^2 \cup \{p_0\}$ , mark an edge mapped processed;
11:   $\forall p \in P$ , if  $p$  and  $p_0$  match to the same query edge
12:     $P = P \setminus \{p\}$ ;
13:  $p_0 = \arg \max_{\substack{p \in P \\ p \not\in G_a^2}} \{\text{sim}(Q, G_a^2 \cup \{p\}) - \text{sim}(Q, G_a^2)\}, G_a^2 \cup \{p\} \text{ connected};$ 
14: if  $p_0$  not empty
15:    $G_a^2 = G_a^2 \cup \{p_0\}$ ;
16: else let  $G_a^2$  be the maximum connected part of  $G_a^2$ ;
17: return  $\arg \max\{\text{sim}(Q, G_a^1), \text{sim}(Q, G_a^2)\}$ ;

```

The second greedy algorithm is similar to the first one, except that in each non-final step k , it adds path p_k without taking the connectivity constraint into consideration (connectivity constraint means that p_k should share some common node with G_a^2). In other words, it selects a path p_k that matches an unmatched query edge (u, v) and maximizes

$$p_k = \arg \max_{\substack{p \in P \\ p \not\in G_a^2}} \{\text{sim}(Q, G_a^2 \cup \{p\}) - \text{sim}(Q, G_a^2)\}. \quad (5.8)$$

The second greedy algorithm only takes care of the connectivity constraint in the final step. For the last query edge, we would select the path p by solving Equation 5.7. If the path p exists, we just insert it into the answer graph G_a^2 ; otherwise, we compute the maximal connected component of G_a^2 .

Even though both of the two greedy algorithms may not perform well in some situation, there is at least one of them which is not too far away from optimum. Hence,

---

CHAPTER 5. STRUCTURE AND ATTRIBUTION INDEX-BASED APPROXIMATE GRAPH MATCHING OVER LARGE GRAPHS

---

by selecting the one with higher similarity score as answer graph, our algorithm has a decent answer quality, as verified by the experimental studies (See Section 5.5).

To sum up, our graph matching algorithm is summarized as follows: First, we use Algorithm 5.3 to find the set of best paths; Second, we compute the answer graph by Algorithm 5.4.

Complexity Analysis We now analyze the time complexity of the approximate graph matching algorithm. Assume the time complexity to compute equation 5.5 is $O(\Upsilon(|V(G)|, |E(G)|))$.⁵ In Algorithm 5.3, for each node x , the worst case is to find the set of best paths from node x to other nodes in different partitions, which needs $O(\Upsilon(|V(G_i \cup G_j \cup G_s)|, |E(G_i \cup G_j \cup G_s)|) + |HT|)$. Hence, the overall complexity of Algorithm 5.3 is $O(\Upsilon(|V(G)|, |E(G)|)|E(Q)|d)$, since for each query edge, we need run lines 4–11 for d times in the worst case, where d is the maximum number of nodes in data graph that are attribute compatible to a query node. The time complexity of Algorithm 5.4 is $O(|E(Q)||P|)$ where P is the best path set computed by Algorithm 5.3. Unfortunately, it is non-trivial to give a tight bound for the size of the best path set P , though it is loosely bounded by the number of simple paths between $|E(Q)|$ pairs of nodes, i.e., $O(|E(Q)|2^{|\Sigma|})$. An empirical study shows that in average $|P|$ is linear or even much smaller than $|\Sigma|$. For all the real graphs that we have tested, the ratio between the size of P and $|\Sigma|$ is in the range $[0.0003, 0.86]$.

5.5 Experimental Study

In this section, we conduct a set of experiments on real large graphs to verify the effectiveness and efficiency of our proposed method. We compare our greedy algorithm with G-ray [Tong 07] and memory-based implementation of TALE [Tian 08b]. For fair comparison, we change the original disk-based implementation of TALE to memory-based implementation. In addition, to analyze the benefit of our index technique on overall query processing performance, we implement two algorithms SA-greedy and B-greedy. SA-greedy is a combination of our SA-Index proposed in Section 5.3 with our

---

⁵The computation time complexity is related to the input metric of degree of matching. For instance, in our experiment, the algorithm to compute equation 5.5 is similar to the Dijkstra’s algorithm [Corm 01], which requires $O(|V(G)| \log |V(G)| + |E(G)|)$ time complexity. Hence, in this scenario, $\Upsilon(|V(G)|, |E(G)|) = |V(G)| \log |V(G)| + |E(G)|$.

CHAPTER 5. STRUCTURE AND ATTRIBUTION INDEX-BASED APPROXIMATE GRAPH MATCHING OVER LARGE GRAPHS

greedy algorithm for query processing. B-greedy combines a modularity-based partitioning [Blon 08]⁶ with our second-level indexing as an alternative to SA-Index (namely, B-Index) and feeds the output B-Index to our greedy algorithm. All algorithms proposed in this chapter are implemented in standard C++. The original implementation of G-ray uses MATLAB. We convert the MATLAB code into a C++ shared library and run under the same machine configuration as ours and TALE. All experiments are run on a 64-bit machine with a Intel Xeon(R) 2.67G HZ CPU and 24GB RAM. We use the default settings of G-ray in our experiments; that is, the value of fly-out probability $c=0.1$ and the number of iterations is 10. Regarding to those parameter settings in TALE, we set approximation ratio ρ to be 25% and percentage of important nodes to be 10%.

Datasets. In our experiments, we use the following datasets: protein (PR), amazon (AZ), DBLP, roadnet (RN), patents (PT) and livejournal (LJ). Protein is a human protein interaction network from the Human Protein Database (www.hprd.org), in which nodes are proteins and edges are protein-protein interactions. DBLP is a co-authorship graph extracted from the DBLP Computer Science Bibliography (www.informatik.uni-trier.de/~ley/db/). Amazon, roadnet, patents and livejournal are from Stanford large network dataset collection (snap.stanford.edu/data/). We convert those graphs into undirected ones and generate synthetic attribute information for graphs without attribute information. We give the details of each dataset (number of nodes and edges, physical storage size, whether attribute information is synthetic, number of distinct values of attributes and number of distinct values of labels) in Table 5.2.

Table 5.2: Statistics of Datasets.

	PR	AZ	DBLP	RN	PT	LJ
$ V(G) $	19428	262111	797787	1965206	3774768	4847571
$ E(G) $	36631	899792	1301298	2765348	16518943	42851237
size	584KB	14.6MB	28.6MB	67.6MB	299MB	672MB
synthetic	false	true	false	true	false	true
$ A $	200	5000	3752	5000	987	5000
$ \Sigma $	19428	262111	696360	1965206	3774768	4847571

Evaluation Metrics. We evaluate the proposed methods with the following metrics:

- Scalability with increasing sizes of query graphs and data graphs.

⁶<http://sites.google.com/site/findcommunities/>

CHAPTER 5. STRUCTURE AND ATTRIBUTION INDEX-BASED APPROXIMATE GRAPH MATCHING OVER LARGE GRAPHS

- Node ratio $|V_m|/|V(Q)|$ and edge ratio $|E_m|/|E(Q)|$, where $|V_m|$ is the number of query nodes matched by answer graphs and $|E_m|$ is the number of query edges matched by answer graphs. This metric is originated from the work [Tong 06].
- Approximate quality ratio $\text{sim}(Q, G_a)/\widehat{\text{sim}}_{\text{opt}}$, where G_a denotes the answer graph, sim_{opt} is the degree of matching between the query graph and the optimal answer graph, and $\widehat{\text{sim}}_{\text{opt}}$ is an estimated upper bound for sim_{opt} . We compute $\widehat{\text{sim}}_{\text{opt}}$ by Theorem B.1 which we put in Appendix B.

The first metric evaluates the efficiency of the different approaches, while the remaining metrics evaluate the effectiveness of the different approaches.

Metric of Degree of Matching. In our experiments, we use the following metric to evaluate the degree of matching from a query graph to a data graph. Suppose we are given a query graph Q , a data graph G , and a mapping function $f: \widehat{V}_q \leftrightarrow \widehat{V}$. We define the degree of matching between an edge e in query graph and its matching path p in data graph as

$$\text{sim}(e, p|f) = \alpha \text{sim}_1(e, p|f) + \beta \text{sim}_2(e, p|f), \quad (5.9)$$

where $\text{sim}_1(e, p|f)$, which is the structure similarity between e and p , is defined as $1 - \frac{|w(e) - \text{len}(p)|}{\max(w(e), \text{len}(p))}$; $\text{len}(p)$ denotes the weighted length of a path p ; $w(e)$ denotes the weight of an edge e ; $\text{sim}_2(e, p|f)$, which is the semantic similarity between an edge e and a path p , is defined as

$$\frac{\text{sim}(u_e, f(u_e)) + \text{sim}(v_e, f(v_e))}{2} \times \frac{\sum_{v_0 \in p} \text{sim}(f^{-1}(v_0), v_0)}{|p|},$$

where $|p|$ denotes the number of nodes on p ; $\text{sim}(u, v) = 1$ if $A^Q(u) = A^G(v)$ and ($l^Q(u) = l^G(v)$ or $l^Q(u) = \lambda$), and 0 otherwise; If u (v) is a mismatched node, then $f(u)$ ($f^{-1}(v)$) is empty; α and β are weighting factors that allow users to adjust the importance of structure similarity and semantic similarity (we consider them equally important here and set $\alpha = \beta = 0.5$). Note that although here the similarity measure is defined for weighted graphs, our method works for both weighted and un-weighted graphs.

Now we can define the degree of matching between a query graph Q and a data graph G . Given a query graph Q and a data graph G , the graph matching measure $\text{sim}(Q, G)$ is defined as

$$\text{sim}(Q, G) = (\max_f \sum_{e \in E(Q)} \max_p \text{sim}(e, p|f)) / |E(Q)|, \quad (5.10)$$

where $|E(Q)|$ is the number of edges in the query graph. We use $|E(Q)|$ to normalize $\text{sim}(Q, G)$.

CHAPTER 5. STRUCTURE AND ATTRIBUTION INDEX-BASED APPROXIMATE GRAPH MATCHING OVER LARGE GRAPHS

Figure 5.6: Index Construction Performance Comparison over Real Data.

Figure 5.7: Index Size Comparison over Real Data.

5.5.1 Performance Evaluation on Real Data

Indexing Performance Evaluation. We first evaluate the performance of our SA-Index.

Figure 5.6.a and Figure 5.6.b plot the running time and peak memory consumption during index construction. The results show that our SA-Index takes less time to construct and uses fewer memory than B-Index. Note that we can not obtain results for B-Index on LJ data since the implementation of modularity-based partitioning [Blon 08] runs out of memory when it is tested on LJ data.

Figure 5.7.a shows a comparison on the number of boundary nodes produced by SA-Index and B-Index. In all cases, we can see that SA-Index results in smaller size of boundary nodes than B-Index. One may notice there is a sharp decrease in SA-Index for graph “RN”. This is because “RN” has a very small average degree, which

CHAPTER 5. STRUCTURE AND ATTRIBUTION INDEX-BASED APPROXIMATE GRAPH MATCHING OVER LARGE GRAPHS

indicates that the graph itself has a relatively low conductance and high homogeneity. Therefore, SA-Index returns a partitioning of “RN” with a small partition number and also a small number of boundary nodes. However, B-Index identifies a large number of small communities and hence produces a large number of boundary nodes. The index storage size of SA-Index and B-Index is shown in Figure 5.7.b. One may expect similar results for index storage size, since both the first-level index (i.e., partition-based index) and the second-level index (connectivity-based index) are highly related to the number of boundary nodes. This phenomena is verified in Figure 5.7.b. The storage size of the SA-Index is consistently smaller than that of the B-Index. In addition, the curve variation of Figure 5.7.b is also analogous to that of Figure 5.7.a.

Figure 5.8: Effect of Data Graph Size.

Efficiency Analysis. In this set of experiments, we use different sizes of query graphs and data graphs to assess the query performance of our proposed methods. The results reported are averaged over ten runs.

First, we report the query performance over six data graphs with number of nodes varying from 20k to 6M. For each graph, we randomly generate a set of queries with fixed node size 30. Figure 5.8.a shows that both SA-greedy and B-greedy are up to over an order of magnitude faster than G-ray and in the worst case, our greedy algorithms are still around 4 times faster than TALE. To keep this experiment manageable, we kill any G-ray query if it runs over 10 hours. Hence, we do not report the query performance of G-ray on PT and LJ data since running a small query using G-ray on PT takes around 20 hours. In addition, compared with B-Index, the improvement of using SA-Index could achieve 70% speed up. Furthermore, B-Index could not run on large graphs such as LJ.

CHAPTER 5. STRUCTURE AND ATTRIBUTION INDEX-BASED APPROXIMATE GRAPH MATCHING OVER LARGE GRAPHS

Figure 5.9: Effect of Query Graph Size.

Figure 5.10: Effect of Using Index.

The query memory consumption of SA-greedy, B-greedy, G-ray and TALE is reported in Figure 5.8.b. Except B-greedy, the memory usages of the other approaches are quite scalable to the data graph size. But for B-greedy, we notice a sharp increase on RN data since it produces a large number of boundary nodes for RN data (maintaining connections among boundary nodes takes up a large volume of memory). Naturally, when the data graph size increases, the query memory consumption of B-greedy may soon grow out of memory as memory size is always limited. In addition, among the three scalable approaches, we notice that our SA-greedy achieves the smallest memory consumption.

Next, we vary the number of query nodes from 5 to 30, and report the total query time of four approaches on protein graph in Figure 5.9. We do not report the query memory consumption since it is dominated by the size of data graphs other than the size of query graphs.

The above experiments demonstrate the efficiency of our greedy algorithm against G-ray and TALE. To further compare the efficiency introduced by using SA-Index against B-Index, we introduce a metric, “ratio of visited nodes”, i.e., number of nodes visited using SA/B-Index against number of nodes visited without index using greedy algorithm, to examine the filtering power of these two indexing approaches. The results are shown in Figure 5.10. Clearly, the average number of nodes accessed by using SA-greedy, is much smaller than that by using B-greedy. This also serve as an indicator for the query performance using the two indexes, as we verify in Figures 5.8.a and 5.9.

Effectiveness Analysis. We present the approximate quality ratio, node ratio and edge ratio of answer graphs returned by our greedy algorithm, G-ray and TALE in Figure 5.11. According to node ratio and edge ratio, as shown in Figures 5.11.b and 5.11.c, the

CHAPTER 5. STRUCTURE AND ATTRIBUTION INDEX-BASED APPROXIMATE GRAPH MATCHING OVER LARGE GRAPHS

Figure 5.11: Effectiveness of Matching Algorithm.

answer graphs computed by our method capture in general more important nodes and edges than those returned by G-ray and TALE. According to the approximate quality ratio comparison shown in Figure 5.11.a, our answer graphs have a higher approximate quality ratio than G-ray’s and TALE’s answer graphs. Specifically, our significant test shows that (Greedy $\gg$ G-ray $\gg$ TALE), where $\gg$ means significantly better with p -value < 0.05 .

Query Examples. In Figures 5.8 to 5.11 we conduct a systematic comparison on the efficiency and the quality of our method with G-ray and TALE. Here, we give a set of three typical example queries to provide the readers a more direct view on the answer graphs returned by Greedy and G-ray (We do not show the answer graphs returned by TALE as G-ray $\gg$ TALE on approximate quality ratio.)

Figure 5.12: Answer Graphs of Line Query.

First, let us consider an example of *line query* shown in Figure 5.12. Both G-ray and our approach find an exact answer graph for the given query with respect to the attributes. In this case, our approach and G-ray have comparable quality of answer graphs. However, one may notice that in the query “Jiawei Han” is associated with “SIGMOD”, but in the answer of G-ray “Michael Stonebraker” is associated with “SIGMOD” instead of “Jiawei Han”. This is because G-ray matches a query only by the attributes of the query nodes. Therefore, our method supports a query type that is more expressive than that supported by G-ray.

CHAPTER 5. STRUCTURE AND ATTRIBUTION INDEX-BASED APPROXIMATE GRAPH MATCHING OVER LARGE GRAPHS

Figure 5.13: Answer Graphs of Star Query.

Next, let us look at an example of *star query*. As shown in Figure 5.13, there is no exact match for the given star query. Hence, both G-ray and our approach return an approximate answer graph for the given query. However, for the G-ray, it fails to find those connections from scholars who publish in “SIGMOD” to those who publish in “ICDT”. In contrast, our approach returns a short path which reflects a connection from scholars who publish in “SIGMOD” to those who publish in “ICDT”.

Figure 5.14: Answer Graphs of Loop Query.

Lastly, we show a *loop query*. The result is shown in Figure 5.14. Although both answer graphs show some good connections between nodes, our method in particular finds the global connection among the given query nodes, which therefore provides more useful information than G-ray.

5.6 Summary

In this chapter, we study the problem of approximate graph matching query over large attributed graphs. We propose a SA-Index so that query processing can be effectively

CHAPTER 5. STRUCTURE AND ATTRIBUTION INDEX-BASED APPROXIMATE GRAPH MATCHING OVER LARGE GRAPHS

---

directed to the much smaller but promising search space. Using the SA-Index, we efficiently compute a set of best paths (or edges) matching the set of query edges. Then, two fast greedy algorithms are proposed to construct the best matching answer graph from the set of best paths. Our extensive experiments on real datasets verify that our index is efficient to construct, and query processing using the index is over an order of magnitude faster than existing work. We also show, by various quality measures, that our method obtains high-quality query answers.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

Recent interests on XML, biological databases, social network analysis, the Semantic Web, Web ontology and many other emerging applications have sparked renewed interests on graph-structured databases (or simply *graphs*) and related problems (e.g., query processing and optimization). Among many graph-based applications, my thesis research studied three types of applications: the reachability query, graph abnormal pattern detection, and approximate graph matching query.

The first part of my thesis research proposed a uniform framework to represent a graph as a set of partitions and a graph skeleton. Each partition could use any existing reachability index. In conjunction with the framework, my work defined a cost function and propose a heuristic algorithm for graph partitioning. With the framework, a reachability query was cast into inter-partition and intra-partition reachability queries on indexes. We presented part of work in the 14th international conference on database systems for advanced application [Zhu 09].

One related problem raised in the first part of thesis study is about selecting suitable reachability indexing for each partition. This motivated us to study the graph reachability index selection problem and formulate it as the property-based graph classification problem. Instead of mining useful subgraph patterns as features, our work enumerated various graph theoretical metrics, and chose those discriminative ones as features. Finally, two efficient and effective classifiers were computed for classifying graph into different groups (where graphs with the same reachability index forms a group). Part of this work originally appeared in [Zhu 11b] and its extension is still under review.

Another key finding in the first part of my research is that when the given graph is very large, often preprocessing such as indexing technique or summary technique is

CHAPTER 6. CONCLUSIONS AND FUTURE WORK

---

needed in order to facilitate further analysis (i.e., query, mining, graph visualization and so on). As a proof of concept, we studied the problem of approximate graph matching over a large attributed graph. Given a query graph, this work computed a subgraph of the large graph that best matches the query graph. The coauthors and I proposed a novel structural- and attribute- aware index to process approximate graph matching over a large attributed graph. We used the index to find a set of best matching paths. From these best matching paths, we computed the best matching answer graph using a greedy algorithm. This work has been published in [Zhu 11a].

6.2 Future Plan

6.2.1 Disk-Resident Graphs

The first part of my future work will primarily focus on disk-resident massive graphs (networks) processing for specific types of query or mining. In many application domains such as the web and social networks, the underlying graphs are so large that they cannot be stored in main memory, but may need to be stored onto disk. In such cases, it is necessary to design efficient and effective methods for indexing, querying and mining. As a start, I would focus on large real-world graphs, whose degree distribution follows a power law, or called scale-free networks. Currently, I have been involved in a work “Finding Maximal Cliques in Massive Networks by H^* Graph”, which has been published in ACM SIGMOD 2010 [Chen 10]. My plan for the extension work are listed as follows:

1. Developing a set of semi-streaming algorithms to detect link farms (i.e., very dense subgraphs) in a large graph representing blogosphere. Those link farms would be treated as an input to further spam blog detection modular.
2. Investigating an efficient and effective recommendation system when the data itself is so massive that its storage and retrieval would be non-trivial.

6.2.2 Data Value Chain Analysis

There are many fields of science devoted to furthering our understanding of humankind, such as sociology, medicine, economics and political science. In all of these fields, scientific inquiry generally proceeds by making observations about the complex behavior of the people that live within it. Nowadays, with the advent of Web service, a new source of data about people behavior has become available. For example, in the field of sociology,

CHAPTER 6. CONCLUSIONS AND FUTURE WORK

---

to-date, users may share photo albums with friends by “Snapfish” services provided by HP; users may rate the movies in “Yahoo! Movies”; users may take part in an online game and recommend it to their friends in facebook. Taken as a whole, across billions of users, these actions constitute a measurement of the world and humanity through time. However, particular challenges exist in the area of user activity analysis. Some of these are as follows:

- Modeling the complex relation among users, actions and time is non-trivial. It is not just a simple two- tuple relation between user and action or similarity of users/actions.
- Even the problem computations are conceptually straightforward, the input data, the intermediate data and even the output data is usually large and the computations have to be distributed across a number of machines.

This research proposes to:

1. To explore a data model which could facilitate the reading, storing and sending of long-term logs and as well as the output.
2. To design an effective approach to extract users’ temporal trails through logs of actions performed by users, dig out social relation among users on the basis of shared interest and predict users’ next actions.
3. To develop an efficient algorithm to identify the propagation of influence (i.e., leaders) for a specific action.

Next, we present some design issues of the proposed problems and our approaches to address them:

Log Table Storage. Given the large volume of long-term log data, using any existing relational DB such as SQL, MYSQL and PostgreSQL would ease the workload of implementation, but may not be amendable to achieve scalability and high performance regarding to high IO cost. Therefore, the proposed solution would introduce a new data model (such as that used in Google’s Big Table [Chan 06]) that store data as sparse matrices. These are different from relational data models which store data in tabular format. Flat files have shown promise by providing efficient access to large data volumes (e.g. Google’s online applications such as Google Earth).

Privacy Risks of User Log Retention. As pointed in the work [Coop 08], while logs—which generally consist of the user’s IP address, a time stamp, the user’s browser and operating system information, the user’s cookie ID, and the result clicked—may be extremely useful as a research or marketing tool, they also present serious privacy risks to those doing the actions. Hence, the proposed solution would embed the privacy-preserving mining algorithms into the whole framework, to prevent disclosure to third parties, government and misuse of user profiles.

Multi-Relation Analysis. It might not be as simple as a single mutual reinforce relation between users and actions. Instead, it is a complex data model which contains user-user interaction, action-action interest share, and user-action relation. Therefore, the proposed approach would use the combination of relation hyper-graph and high-dimension tensor to model the data.

Propagation Influence Identification. Existing work on propagation influence identification either use information theory or network flow to solve the problem. Our solution would propose a good objective function and formulate the propagation problem as a Linear Programming problem. Next, we would transform the problem into min-cost flow problem and apply algorithms with bounded approximate ratio to solve it. This approach would be quite efficient and effective, as both of its time complexity and approximate ratio is bounded.

6.2.3 Graph Abnormal Node Detection

Besides the thesis study on efficient graph querying and mining algorithms, we have also worked on a spam blog detection project. Its purpose was to investigate how to detect spam blogs based on search results when modeling data and their relationships as graphs. Some techniques of this project have been published in [Zhu 08, Zhu 11c]. This study motivated us to focus on another interesting problem: abnormal node detection in dynamic directed graphs.

Traditional normality scoring function assumes that the normality score between two nodes is related to their neighborhood connectivity. That is, two nodes are similar to each other if they are referenced by similar entities or if they reference similar entities. However, normality computation is really domain-specific, and multiple aspects can be exploited for normality scoring function.

CHAPTER 6. CONCLUSIONS AND FUTURE WORK

Another challenge is introduced by the unique characteristics of dynamic graphs. In static graphs, given a query node u , the normality score of another node v to u is related to their neighborhood connectivity. However, in dynamic graphs, the connectivity between each two nodes changes over time. Therefore, the normality score between two nodes at future may depend not only on its current time, but also on its past.

Last but not the least, efficiency is always an important issue in order to make the proposed approach practical in real applications. Optimization techniques are desirable to facilitate normality score computation.

In order to address the above issues, my future research will focus on the following aspects:

- Proposing a novel and effective normality scoring function. We plan to combine the traditional normality scoring function with structure similarity (i.e., whether the neighborhood graph of a node can approximately match that of another node), to define and formulate a new normality scoring function. In addition, we would develop a fixed-point iterative algorithm toward effectively computing normality scores.
- Developing two different formulations, i.e., memoryless and with memory, for normality scoring function. The former one assumes that the normality score at future (i.e., time $t + 1$) depends only on its current time t ; while the latter one assumes that the normality score at future may depend not only on its current time, but also on its past.
- Investigating how reachability query can be used to optimize normality score computation. Clearly, if two nodes can not reach each other, then their normality score must be low or even equal to zero. Therefore, possibly reachability query test can help prune the computation space: the normality score between two vertices far apart to be zero, and only those vertex-pairs reachable within a radius of r from each other are considered in normality computation.

References

- [Abra 03] A. Abraham. “Business Intelligence from Web Usage Mining”. *Journal of Information and Knowledge Management*, Vol. 2, No. 4, pp. 375–390, 2003.
- [Agra 89] R. Agrawal, A. Borgida, and H. V. Jagadish. “Efficient Management of Transitive Relationships in Large Data and Knowledge Bases”. In: *Proceedings of the 1989 ACM SIGMOD International Conference on Management of data*, pp. 253–262, ACM, New York, NY, USA, 1989.
- [Ande 07] R. Andersen and S. M. Cioaba. “Spectral Densest Subgraph and Independence Number of a Graph”. *Journal of Universal Computer Science*, Vol. 13, No. 11, pp. 1501–1513, 2007.
- [Aror 09] S. Arora, S. Rao, and U. Vazirani. “Expander Flows, Geometric Embeddings and Graph Partitioning”. *Journal of ACM*, Vol. 56, pp. 5:1–5:37, April 2009.
- [Baba 83] L. Babai and E. M. Luks. “Canonical labeling of graphs”. In: *Proceedings of the 15th annual ACM Symposium on Theory of Computing*, pp. 171–183, ACM, New York, NY, USA, 1983.
- [Bar 02] Z. Bar-Yossef and S. Rajagopalan. “Template Detection via Data Mining and its Applications”. In: *Proceedings of the 11th International Conference on World Wide Web*, pp. 580–591, ACM, New York, NY, USA, 2002.
- [Bata] V. Batagelj and A. Mrvar. “Pajek Datasets”. webpage. <http://vlado.fmf.uni-lj.si/pub/networks/data/>.
- [Becc] L. Becchetti, C. Castillo, D. Donato, S. Leonardi, and R. Baeza-Yates. “Link-Based Characterization and Detection of Web Spam”. In: *Proceedings of the 2nd International Workshop on Adversarial Information Retrieval on the Web*, ACM, New York, NY, USA.

REFERENCES

- [Berg 87] M. J. Berger and S. H. Bokhari. “A Partitioning Strategy for Nonuniform Problems on Multiprocessors”. *IEEE Transactions on Computers*, Vol. 36, No. 5, pp. 570–580, 1987.
- [Blon 08] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre. “Fast Unfolding of Communities in Large Networks”. *Journal of Statistical Mechanics: Theory and Experiment*, p. P10008, 2008.
- [Boll 98] B. Bollobas. *Modern Graph Theory*. Graduate Texts in Mathematics, Vol. 184, Springer Verlag, 1998.
- [Bram 08] R. Bramandia, B. Choi, and W. K. Ng. “On Incremental Maintenance of 2-hop Labeling of Graphs”. In: *Proceeding of the 17th International Conference on World Wide Web*, pp. 845–854, ACM, New York, NY, USA, 2008.
- [Brin 98] S. Brin and L. Page. “The Anatomy of A Large-scale Hypertextual Web Search Engine”. *Computer Networks and ISDN Systems*, Vol. 30, No. 1–7, pp. 107–117, 1998.
- [Bunk 97] H. Bunke. “On A Relation between Graph Edit Distance and Maximum Common Subgraph”. *Pattern Recognition Letters*, Vol. 18, No. 9, pp. 689–694, 1997.
- [Cai 04] D. Cai, S. Yu, J.-R. Wen, and W.-Y. Ma. “Block-based Web Search”. In: *Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 456–463, ACM, New York, NY, USA, 2004.
- [Cai 10] J. Cai and C. K. Poon. “Path-hop: Efficiently Indexing Large Graphs for Reachability Queries”. In: *Proceedings of the 19th ACM International Conference on Information and Knowledge Management*, pp. 119–128, ACM, New York, NY, USA, 2010.
- [Cao 08] H. Cao, D. Jiang, J. Pei, Q. He, Z. Liao, E. Chen, and H. Li. “Context-aware Query Suggestion by Mining Click-through and Session Data”. In: *Proceeding of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 875–883, ACM, New York, NY, USA, 2008.

REFERENCES

- [Chak 04] D. Chakrabarti. “AutoPart: Parameter-free Graph Partitioning and Outlier Detection”. In: *Proceedings of the 8th European Conference on Principles and Practice of Knowledge Discovery in Databases*, pp. 112–124, Springer-Verlag New York, Inc., New York, NY, USA, 2004.
- [Chan 06] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. “Bigtable: A Distributed Storage System for Structured Data”. In: *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation*, pp. 15–15, USENIX Association, Berkeley, CA, USA, 2006.
- [Chen 03] Q. Chen, A. Lim, and K. W. Ong. “D(k)-index: An Adaptive Structural Summary for Graph-structured Data”. In: *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, pp. 134–144, ACM, New York, NY, USA, 2003.
- [Chen 05a] L. Chen, A. Gupta, and M. E. Kurul. “Efficient Algorithms for Pattern Matching on Directed Acyclic Graphs”. In: *Proceedings of the 21st International Conference on Data Engineering*, pp. 384–385, IEEE Computer Society, Washington, DC, USA, 2005.
- [Chen 05b] L. Chen, A. Gupta, and M. E. Kurul. “Stack-based Algorithms for Pattern Matching on DAGs”. In: *Proceedings of the 31st International Conference on Very Large Data Bases*, pp. 493–504, Very Large Data Bases Endowment, 2005.
- [Chen 06] J. Cheng, J. X. Yu, X. Lin, H. Wang, and P. S. Yu. “Fast Computation of Reachability Labeling for Large Graphs”. In: *Proceedings of the 10th International Conference on Extending Database Technology*, pp. 961–979, 2006.
- [Chen 07] J. Cheng, Y. Ke, W. Ng, and A. Lu. “Fg-index: Towards Verification-free Query Processing on Graph Databases”. In: *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data*, pp. 857–872, ACM, New York, NY, USA, 2007.

REFERENCES

- [Chen 08] J. Cheng, J. X. Yu, X. Lin, H. Wang, and P. S. Yu. “Fast Computing Reachability Labelings for Large Graphs with High Compression Rate”. In: *Proceedings of the 11th International Conference on Extending Database Technology*, pp. 193–204, ACM, New York, NY, USA, 2008.
- [Chen 09] C. Chen, C. X. Lin, M. Fredrikson, M. Christodorescu, X. Yan, and J. Han. “Mining Graph Patterns Efficiently via Randomized Summaries”. *The Proceedings of the Very Large Data Bases Endowment*, Vol. 2, No. 1, pp. 742–753, 2009.
- [Chen 10] J. Cheng, Y. Ke, A. W.-C. Fu, J. X. Yu, and L. Zhu. “Finding Maximal Cliques in Massive Networks by H*-graph”. In: *Proceedings of the 2010 International Conference on Management of Data*, pp. 447–458, ACM, New York, NY, USA, 2010.
- [Cho 02] Y. Cho. “A Personalized Recommender System based on Web Usage Mining and Decision Tree Induction”. *Expert Systems with Applications*, Vol. 23, No. 3, pp. 329–342, 2002.
- [Clau 04] A. Clauset, M. E. J. Newman, and C. Moore. “Finding Community Structure in Very Large Networks”. *Physical Review E*, Vol. 70, p. 066111, 2004.
- [Coff 04] T. R. Coffman and S. E. Marcus. “Dynamic Classification of Groups through Social Network Analysis and HMMs”. In: *IEEE Aerospace Conference*, pp. 3197–3205, 2004.
- [Cohe 03] E. Cohen, E. Halperin, H. Kaplan, and U. Zwick. “Reachability and Distance Queries via 2-Hop Labels”. *SIAM Journal on Computing*, Vol. 32, No. 5, pp. 1338–1355, 2003.
- [Cook 06] D. J. Cook and L. B. Holder. *Mining Graph Data*. Wiley-IEEE, 2006.
- [Cool 97] R. Cooley, B. Mobasher, and J. Srivastava. “Web Mining: Information and Pattern Discovery on the World Wide Web”. In: *Proceedings of the 9th International Conference on Tools with Artificial Intelligence*, pp. 558–567, IEEE Computer Society, Washington, DC, USA, 1997.
- [Coop 08] A. Cooper. “A Survey of Query Log Privacy-enhancing Techniques from a Policy Perspective”. *ACM Transaction on Web*, Vol. 2, No. 4, pp. 1–27, 2008.

REFERENCES

- [Corm 01] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, 2nd Ed., 2001.
- [Cove 67] T. Cover and P. Hart. “Nearest neighbor pattern classification”. *IEEE Transactions on Information Theory*, Vol. 13, No. 1, pp. 21–27, January 1967.
- [Desh 05] M. Deshpande, M. Kuramochi, N. Wale, and G. Karypis. “Frequent Substructure-Based Approaches for Classifying Chemical Compounds”. *IEEE Transaction on Knowledge and Data Engineering*, Vol. 17, No. 8, pp. 1036–1050, 2005.
- [Dies 05] R. Diestel. *Graph Theory*. Vol. 173, Springer-Verlag, Heidelberg, third Ed., 2005.
- [Diet 82] P. F. Dietz. “Maintaining Order in A Linked List”. In: *Proceedings of the 14th annual ACM Symposium on Theory of Computing*, pp. 122–127, ACM, New York, NY, USA, 1982.
- [Eiri 03] M. Eirinaki and M. Vazirgiannis. “Web Mining for Web Personalization”. *ACM Transaction on Internet Technology*, Vol. 3, pp. 1–27, February 2003.
- [Euro] European Bioinformatics Institute. “QuickGO: GO Browser”. Web interface. <http://www.ebi.ac.uk/ego/>.
- [Even 00] G. Even, J. S. Naor, S. Rao, and B. Schieber. “Divide-and-conquer Approximation Algorithms via Spreading Metrics”. *Journal of ACM*, Vol. 47, pp. 585–616, July 2000.
- [Falo 06] M. Faloutsos, Q. Yang, G. Sigamos, and S. Lonardi. “Evolution versus Intelligent Design: Comparing the Topology of Protein-protein Interaction Networks to the Internet”. In: *Proceedings of the LSS Computational Systems Bioinformatics Conference*, pp. 299–310, Stanford, CA, 2006.
- [Fidu 88] C. M. Fiduccia and R. M. Mattheyses. “A Linear-time Heuristic for Improving Network Partitions”. In: *25 years of DAC: Papers on Twenty-five years of electronic design automation*, pp. 241–247, 1988.

REFERENCES

- [Fied 73] M. Fiedler. “Algebraic Connectivity of Graphs”. *Czechoslovak Mathematical Journal*, Vol. 23, No. 2, pp. 298–305, 1973.
- [Gare 79] M. R. Garey and D. S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979.
- [Gent 03] R. Gentilini, C. Piazza, and A. Policriti. “From Bisimulation to Simulation: Coarsest Partition Problems”. *Journal of Automated Reasoning*, Vol. 31, No. 1, pp. 73–103, 2003.
- [Gibs 05] D. Gibson, R. Kumar, and A. Tomkins. “Discovering Large Dense Subgraphs in Massive Graphs”. In: *Proceedings of the 31st International Conference on Very Large Data Bases*, pp. 721–732, Very Large Data Bases Endowment, 2005.
- [Gome 00] M. M. y Gómez, A. López-López, and A. F. Gelbukh. “Information Retrieval with Conceptual Graph Matching”. In: *Proceedings of the 11th International Conference on Database and Expert Systems Applications*, pp. 312–321, Springer-Verlag, London, UK, 2000.
- [Gupt 97a] A. Gupta. “Fast and Effective Algorithms for Graph Partitioning and Sparse-Matrix Ordering”. *IBM Journal of Research and Development*, Vol. 41, No. 1-2, pp. 171–183, 1997.
- [Gupt 97b] A. Gupta, G. Karypis, and V. Kumar. “Highly Scalable Parallel Algorithms for Sparse Matrix Factorization”. *IEEE Transactions on Parallel and Distributed Systems*, Vol. 8, No. 5, pp. 502–520, 1997.
- [He 05] H. He, H. Wang, J. Yang, and P. S. Yu. “Compact Reachability Labeling for Graph-structured Data”. In: *Proceedings of the 14th ACM International Conference on Information and Knowledge Management*, pp. 594–601, ACM, New York, NY, USA, 2005.
- [He 06] H. He and A. K. Singh. “Closure-Tree: An Index Structure for Graph Queries”. In: *Proceedings of the 22nd International Conference on Data Engineering*, p. 38, IEEE Computer Society, Washington, DC, USA, 2006.

REFERENCES

- [He 08] H. He and A. K. Singh. “Graphs-at-a-time: Query Language and Access Methods for Graph Databases”. In: *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, pp. 405–418, ACM, New York, NY, USA, 2008.
- [Henz 95] M. R. Henzinger, T. A. Henzinger, and P. W. Kopke. “Computing Simulations on Finite and Infinite Graphs”. In: *Proceedings of the 36th Annual Symposium on Foundations of Computer Science*, p. 453, IEEE Computer Society, Washington, DC, USA, 1995.
- [Huan 04] J. Huan, W. Wang, J. Prins, and J. Yang. “SPIN: Mining Maximal Frequent Subgraphs from Graph Databases”. In: *Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 581–586, ACM, New York, NY, USA, 2004.
- [Huan 08] L. Huang, D. Yan, M. I. Jordan, and N. Taft. “Spectral Clustering with Perturbed Data”. In: *Proceedings of the Twenty-Second Annual Conference on Neural Information Processing Systems*, pp. 705–712, MIT Press, Cambridge, MA, USA, 2008.
- [Jian 07] H. Jiang, H. Wang, P. S. Yu, and S. Zhou. “GString: A Novel Approach for Efficient Search in Graph Databases”. In: *Proceedings of the 23rd International Conference on Data Engineering*, pp. 566 – 575, IEEE Computer Society, Washington, DC, USA, 2007.
- [Jin 08] R. Jin, Y. Xiang, N. Ruan, and H. Wang. “Efficiently Answering Reachability Queries on Very Large Directed Graphs”. In: *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, pp. 595–608, ACM, New York, NY, USA, 2008.
- [Jin 09] R. Jin, Y. Xiang, N. Ruan, and D. Fuhry. “3-HOP: a High-compression Indexing Scheme for Reachability Query”. In: *Proceedings of the 35th SIGMOD International Conference on Management of Data*, pp. 813–826, ACM, New York, NY, USA, 2009.
- [Jin 10a] C. Jin, S. S. Bhowmick, X. Xiao, J. Cheng, and B. Choi. “GBLENDER: Towards Blending Visual Query Formulation and Query Processing in Graph Databases”. In: *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, pp. 111–122, ACM, New York, NY, USA, 2010.

REFERENCES

- [Jin 10b] N. Jin, C. Young, and W. Wang. “GAIA: Graph Classification using Evolutionary Computation”. In: *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, pp. 879–890, ACM, New York, NY, USA, 2010.
- [Kary 98] G. Karypis and V. Kumar. “A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs”. *SIAM Journal on Scientific Computing*, Vol. 20, No. 1, pp. 359–392, 1998.
- [Kaus 02] R. Kaushik, P. Shenoy, P. Bohannon, and E. Gudes. “Exploiting Local Similarity for Indexing Paths in Graph-Structured Data”. In: *Proceedings of the 18th International Conference on Data Engineering*, p. 129, IEEE Computer Society, Washington, DC, USA, 2002.
- [Kern 70] B. W. Kernighan and S. Lin. “An Efficient Heuristic Procedure for Partitioning Graphs”. *The Bell System Technical Journal*, Vol. 49, No. 1, pp. 291–307, 1970.
- [Klei 99] J. M. Kleinberg. “Authoritative Sources in a Hyperlinked Environment”. *Journal of ACM*, Vol. 46, No. 5, pp. 604–632, 1999.
- [Kong 10] X. Kong and P. S. Yu. “Semi-supervised Feature Selection for Graph Classification”. In: *Proceedings of the 16th ACM International Conference on Knowledge Discovery and Data Mining*, pp. 793–802, ACM, New York, NY, USA, 2010.
- [Kram 01] S. Kramer, L. De Raedt, and C. Helma. “Molecular Feature Mining in HIV Data”. In: *Proceedings of the 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 136–143, ACM, New York, NY, USA, 2001.
- [Krau 09] R. Krauthgamer, J. S. Naor, and R. Schwartz. “Partitioning Graphs into Balanced Components”. In: *Proceedings of the twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 942–949, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2009.

REFERENCES

- [Krit 06] A. Kritikopoulos, M. Sideri, and I. Varlamis. “BlogRank: Ranking Weblogs Based on Connectivity and Similarity Features”. In: *Proceedings of the 2nd International Workshop on Advanced Architectures and Algorithms for Internet Delivery and Applications*, p. 8, ACM, New York, NY, USA, 2006.
- [Kura 01] M. Kuramochi and G. Karypis. “Frequent Subgraph Discovery”. In: *Proceedings of the 2001 IEEE International Conference on Data Mining*, pp. 313–320, IEEE Computer Society, Washington, DC, USA, 2001.
- [Lesk 08] J. Leskovec, K. J. Lang, A. Dasgupta, and M. W. Mahoney. “Community Structure in Large Networks: Natural Cluster Sizes and the Absence of Large Well-Defined Clusters”. *Computing Research Repository*, Vol. abs/0810.1355, 2008.
- [Lin 08] J. Lin, D. Etter, and D. DeBarr. “Exact and Approximate Reverse Nearest Neighbor Search for Multimedia Data”. In: *Proceedings of the SIAM International Conference on Data Mining*, pp. 656–667, SIAM, Philadelphia, PA, USA, 2008.
- [Luce 49] R. Luce and A. Perry. “A Method of Matrix Analysis of Group Structure”. *Psychometrika*, Vol. 14, No. 2, pp. 95–116, 1949.
- [Marc 02] S. Marcus and T. R. Coffman. “Terrorist Modus Operandi Discovery System 1.0: Functionality, Examples, and Value.”. In: *21st Century Technologies Internal Publication*, Austin, TX, 2002.
- [Mats 99] H. Matsuda, T. Ishihara, and A. Hashimoto. “Classifying Molecular Sequences Using a Linkage Graph with Their Pairwise Similarities”. *Theoretical Computer Science*, Vol. 210, No. 2, pp. 305–325, 1999.
- [Mend 96] A. Mendelzon, G. Mihaila, and T. Milo. “Querying the World Wide Web”. *Parallel and Distributed Information Systems, International Conference*, Vol. 0, p. 0080, 1996.
- [Milo 99] T. Milo and D. Suciu. “Index Structures for Path Expressions”. In: *Proceedings of the 7th International Conference on Database Theory*, pp. 277–295, Springer-Verlag, London, UK, 1999.

REFERENCES

- [Ntou 06] A. Ntoulas, M. Najork, M. Manasse, and D. Fetterly. “Detecting Spam Web Pages through Content Analysis”. In: *Proceedings of the 15th International Conference on World Wide Web*, pp. 83–92, ACM, New York, NY, USA, 2006.
- [Poth 90] A. Pothén, H. D. Simon, and K.-P. Liou. “Partitioning Sparse Matrices with Eigenvectors of Graphs”. *SIAM Journal on Matrix Analysis and Applications*, Vol. 11, No. 3, pp. 430–452, 1990.
- [Ranu 09] S. Ranu and A. K. Singh. “GraphSig: A Scalable Approach to Mining Significant Subgraphs in Large Graph Databases”. In: *Proceedings of the 2009 IEEE International Conference on Data Engineering*, pp. 844–855, IEEE Computer Society, Washington, DC, USA, 2009.
- [Robb 02] S. W. A. L. J. L. C. D. b. B. W. O. P. J. Robbie Loewith, Estela Jacinto and M. N. Hall1. “Two TOR Complexes, Only One of which Is Rapamycin Sensitive, Have Distinct Roles in Cell Growth Control”. *Molecular Cell*, Vol. 10, pp. 457–468, 2002.
- [S Ab 95] R. H. S. Abiteboul and V. Vianu. *Foundations of Databases: The Logical Level*. Addison-Wesley Longman Publishing Co., Inc, Boston, MA, USA, 1995.
- [Saig 08] H. Saigo, N. Krämer, and K. Tsuda. “Partial Least Squares Regression for Graph Mining”. In: *Proceeding of the 14th ACM International Conference on Knowledge Discovery and Data Mining*, pp. 578–586, ACM, New York, NY, USA, 2008.
- [Sche 04] R. Schenkel, A. Theobald, and G. Weikum. “HOPI: An Efficient Connection Index for Complex XML Document Collections”. In: *Proceedings of the 9th International Conference on Extending Database Technology*, pp. 237–255, ACM, New York, NY, USA, 2004.
- [Seid 83] S. B. Seidman. “Network Structure and Minimum Degree”. *Social Networks*, Vol. 5, pp. 269–287, 1983.
- [Shan 08] H. Shang, Y. Zhang, X. Lin, and J. X. Yu. “Taming Verification Hardness: An Efficient Algorithm for Testing Subgraph Isomorphism”. *The Proceedings of the Very Large Data Bases Endowment*, Vol. 1, No. 1, pp. 364–375, 2008.

REFERENCES

- [Shap 81] L. Shapiro and R. Haralick. “Structural Descriptions and Inexact Matching”. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 3, pp. 504–519, 1981.
- [Shas 02] D. Shasha, J. T. L. Wang, and R. Giugno. “Algorithmics and Applications of Tree and Graph Searching”. In: *Proceedings of the 21st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pp. 39–52, ACM, New York, NY, USA, 2002.
- [Shen 06] G. Shen, B. Gao, T.-Y. Liu, G. Feng, S. Song, and H. Li. “Detecting Link Spam Using Temporal Information”. In: *Proceedings of the 6th International Conference on Data Mining*, pp. 1049–1053, IEEE Computer Society, Washington, DC, USA, 2006.
- [Sriv 00] J. Srivastava, R. Cooley, M. Deshpande, and P.-N. Tan. “Web Usage Mining: Discovery and Applications of Usage Patterns from Web Data”. *SIGKDD Explorations Newsletter*, Vol. 1, pp. 12–23, January 2000.
- [Svor 07] K. M. Svore, Q. Wu, C. J. C. Burges, and A. Raman. “Improving Web Spam Classification Using Rank-time Features”. In: *Proceedings of the 3rd International Workshop on Adversarial information Retrieval on the Web*, pp. 9–16, ACM, Banff, Alberta, Canada, 2007.
- [Tan 06] B. Tan, X. Shen, and C. Zhai. “Mining Long-term Search History to Improve Search Accuracy”. In: *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 718–723, ACM, New York, NY, USA, 2006.
- [Tao 04] Y. Tao, D. Papadias, and X. Lian. “Reverse kNN Search in Arbitrary Dimensionality”. In: *Proceedings of the 30th International Conference on Very Large Data Bases*, pp. 744–755, Very Large Data Bases Endowment, 2004.
- [Tarj 72] R. E. Tarjan. “Depth-First Search and Linear Graph Algorithms”. *SIAM Journal on Computing*, Vol. 1, No. 2, pp. 146–160, 1972.
- [Thom 04] B. E. Thomason, T. R. Coffman, and S. E. Marcus. “Sensitivity of Social Network Analysis Metrics to Observation Noise”. In: *IEEE Aerospace Conference*, pp. 3206–3216, 2004.

REFERENCES

- [Thom 06] L. T. Thomas, S. R. Valluri, and K. Karlapalem. “MARGIN: Maximal Frequent Subgraph Mining”. In: *Proceedings of the 6th International Conference on Data Mining*, pp. 1097–1101, IEEE Computer Society, Washington, DC, USA, 2006.
- [Tian 07] Y. Tian, R. C. McEachin, C. Santos, D. J. States, and J. M. Patel. “SAGA: A Subgraph Matching Tool for Biological Graphs”. *Bioinformatics Journal*, Vol. 23, No. 2, pp. 232–239, 2007.
- [Tian 08a] Y. Tian, R. A. Hankins, and J. M. Patel. “Efficient Aggregation for Graph Summarization”. In: *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, pp. 567–580, ACM, New York, NY, USA, 2008.
- [Tian 08b] Y. Tian and J. M. Patel. “TALE: A Tool for Approximate Large Graph Matching”. In: *Proceedings of the 24th International Conference on Data Engineering*, pp. 963–972, IEEE Computer Society, Washington, DC, USA, 2008.
- [Tong 06] H. Tong and C. Faloutsos. “Center-piece Subgraphs: Problem Definition and Fast Solutions”. In: *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 404–413, ACM, New York, NY, USA, 2006.
- [Tong 07] H. Tong, C. Faloutsos, B. Gallagher, and T. Eliassi-Rad. “Fast Best-effort Pattern Matching in Large Attributed Graphs”. In: *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 737–746, ACM, New York, NY, USA, 2007.
- [Tris 07] S. Trißl and U. Leser. “Fast and Practical Indexing and Querying of Very Large Graphs”. In: *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data*, pp. 845–856, ACM, New York, NY, USA, 2007.
- [TSAI 79] W. H. TSAI and K. FU. “Error-correcting Isomorphisms of Attributed Relational Graphs for Pattern Recognition”. *IEEE Transaction on Systems, Man, and Cybernetics*, Vol. 9, No. 12, pp. 757–768, 1979.

REFERENCES

- [TSAI 83] W. H. TSAI and K. FU. “Subgraph Error-Correcting Isomorphisms of Syntactic Pattern Recognition”. *IEEE Transaction on Systems, Man, and Cybernetics*, Vol. 13, No. 1, pp. 48–62, 1983.
- [Tyag 10] N. K. Tyagi, A. K. Solanki, and M. Wadhwa. “Analysis of Server Log by Web Usage Mining for Website Improvement”. *International Journal of Computer Science Issues*, Vol. 7, No. 4, pp. 17–21, 2010.
- [Ullm 76] J. R. Ullmann. “An Algorithm for Subgraph Isomorphism”. *The Journal of the ACM*, Vol. 23, No. 1, pp. 31–42, 1976.
- [Univ] University of Michigan. “The Origin of Power-Laws in Internet Topologies Revisited”. Web page. <http://topology.eecs.umich.edu/data.html>.
- [Vane 02] N. Vanetik, E. Gudes, and S. E. Shimony. “Computing Frequent Graph Patterns from Semistructured Data”. In: *Proceedings of the 2002 IEEE International Conference on Data Mining*, p. 458, IEEE Computer Society, Washington, DC, USA, 2002.
- [W3C] W3C. “OWL Web Ontology Language Overview”. <http://www.w3.org/TR/owl-features>.
- [Wang 02] J. Wang, Z. Chen, L. Tao, W.-Y. Ma, and L. Wenyin. “Ranking User’s Relevance to A Topic through Link Analysis on Web Logs”. In: *Proceedings of the 4th International Workshop on Web Information and Data Management*, pp. 49–54, ACM, New York, NY, USA, 2002.
- [Wang 06] H. Wang, H. He, J. Yang, P. S. Yu, and J. X. Yu. “Dual Labeling: Answering Graph Reachability Queries in Constant Time”. In: *Proceedings of the 22nd International Conference on Data Engineering*, p. 75, IEEE Computer Society, Washington, DC, USA, 2006.
- [Webe 98] R. Weber, H.-J. Schek, and S. Blott. “A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces”. In: *Proceedings of the 24rd International Conference on Very Large Data Bases*, pp. 194–205, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1998.

REFERENCES

- [Will 07] D. Williams, J. Huan, and W. Wang. “Graph Database Indexing Using Structured Graph Decomposition”. In: *Proceedings of the 23rd International Conference on Data Engineering*, pp. 976–985, IEEE Computer Society, Washington, DC, USA, 2007.
- [Wolk 99] H. Wolkowicz and Q. Zhao. “Semidefinite Programming Relaxations for the Graph Partitioning Problem”. *Discrete Applied Mathematics*, Vol. 96–97, pp. 461–479, 1999.
- [Wu 04] X. Wu, M. L. Lee, and W. Hsu. “A Prime Number Labeling Scheme for Dynamic Ordered XML Trees”. In: *Proceedings of the 20th International Conference on Data Engineering*, p. 66, IEEE Computer Society, Washington, DC, USA, 2004.
- [Wu 05] B. Wu and B. D. Davison. “Identifying Link Farm Spam Pages”. In: *Special interest tracks and posters of the 14th International Conference on World Wide Web*, pp. 820–829, ACM, New York, NY, USA, 2005.
- [Wu 06] G. Wu, K. Zhang, C. Liu, and J.-Z. Li. “Adapting Prime Number Labeling Scheme for Directed Acyclic Graphs”. In: *Proceedings of the 11th International Conference on Database Systems for Advanced Applications*, pp. 787–796, Springer-Verlag, Berlin, Heidelberg, 2006.
- [Yan 03] X. Yan and J. Han. “CloseGraph: Mining Closed Frequent Graph Patterns”. In: *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 286–295, ACM, New York, NY, USA, 2003.
- [Yan 04] X. Yan, P. S. Yu, and J. Han. “Graph Indexing: A Frequent Structure-based Approach”. In: *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, pp. 335–346, ACM, New York, NY, USA, 2004.
- [Yan 08] X. Yan, H. Cheng, J. Han, and P. S. Yu. “Mining Significant Graph Patterns by Leap Search”. In: *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, pp. 433–444, ACM, New York, NY, USA, 2008.

REFERENCES

- [Yild 10] H. Yildirim, V. Chaoji, and M. J. Zaki. “GRAIL: Scalable Reachability Index for Large Graphs”. *Proceedings of Very Large Data Base Endowment*, Vol. 3, pp. 276–284, September 2010.
- [Zain 95] O. R. Zaïne and J. Han. “Resource and Knowledge Discovery in Global Information Systems: A Preliminary Design and Experiment”. In: *Proceedings of the 1st International Conference on Knowledge Discovery and Data Mining*, pp. 331–336, ACM, New York, NY, USA, 1995.
- [Zhan 01] C. Zhang, J. Naughton, D. DeWitt, Q. Luo, and G. Lohman. “On Supporting Containment Queries in Relational Database Management Systems”. In: *Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data*, pp. 425–436, ACM, New York, NY, USA, 2001.
- [Zhan 07] S. Zhang, M. Hu, and J. Yang. “TreePi: A Novel Graph Indexing Method”. In: *Proceedings of the 23rd International Conference on Data Engineering*, pp. 966–975, IEEE Computer Society, Washington, DC, USA, 2007.
- [Zhan 09] S. Zhang, S. Li, and J. Yang. “GADDI: Distance Index Based Subgraph Matching in Biological Networks”. In: *Proceedings of the 12th International Conference on Extending Database Technology*, pp. 192–203, ACM, New York, NY, USA, 2009.
- [Zhao 07] P. Zhao, J. X. Yu, and P. S. Yu. “Graph Indexing: Tree + Delta $\leq$ Graph”. In: *Proceedings of the 33rd International Conference on Very Large Data Bases*, pp. 938–949, Very Large Data Bases Endowment, 2007.
- [Zhou 09] Y. Zhou, H. Cheng, and J. X. Yu. “Graph Clustering Based on Structural/Attribute Similarities”. *The Proceedings of the Very Large Data Bases Endowment*, Vol. 2, No. 1, pp. 718–729, 2009.
- [Zhu 08] L. Zhu, A. Sun, and B. Choi. “Online Spam-Blog Detection through Blog Search”. In: *Proceeding of the 17th ACM Conference on Information and Knowledge Management*, pp. 1347–1348, ACM, New York, NY, USA, 2008.
- [Zhu 09] L. Zhu, B. Choi, B. He, J. X. Yu, and W. K. Ng. “A Uniform Framework for Ad-Hoc Indexes to Answer Reachability Queries on Large Graphs”. In: *Proceedings of the 14th International Conference on Database Systems for Advanced Applications*, pp. 138–152, Springer-Verlag, Berlin, Heidelberg, 2009.

REFERENCES

- [Zhu 11a] L. Zhu, W. K. Ng, and J. Cheng. “Structure and Attribute Index for Approximate Graph Matching in Large Graphs”. *Information Systems*, Vol. 36, No. 6, pp. 958–972, 2011.
- [Zhu 11b] L. Zhu, W. K. Ng, and S. Han. “Classifying Graphs Using Theoretical Metrics: A Study of Feasibility”. In: *DASFAA Workshops*, pp. 53–64, 2011.
- [Zhu 11c] L. Zhu, A. Sun, and B. Choi. “Detecting Spam Blogs from Blog Search Results”. *Information Processing & Management*, Vol. 47, No. 2, pp. 246–262, 2011.
- [Zou 08] L. Zou, L. Chen, J. X. Yu, and Y. Lu. “A Novel Spectral Coding in a Large Graph Database”. In: *Proceedings of the 11th International Conference on Extending Database Technology*, pp. 181–192, ACM, New York, NY, USA, 2008.

Appendix A

Support for Other Degree of Matching Metrics

Maximum Common Edges Our approach could be easily extended to support the metric “maximum common edges”:

1. Given a query edge $e \in E(Q)$, if there exists a set of edges E' matching e in the data graph G , then the set of best paths for e is E' , and empty otherwise;
2. In SA-Index construction, we only need to maintain those inter-partition edges into the hash table HT ;
3. In on-line query processing, Algorithm 5.3 could be further simplified as follows: Starting from line 5, if x and y are in the same partition, we only need to scan x 's or y 's neighborhood in G_i in $H^*(G)$ for best paths; if G_i and G_j are different, we skip if neither x nor y is a boundary node; otherwise, we search the hash table HT for best paths.

Graph Edit Distance Normally, a path is a sequence of consecutive edges in a graph. Hence, a pair of isolated nodes (two zero degree nodes) can not be considered as a path. However, in order to support the metric “graph edit distance”, a pair of isolated nodes in data graph is allowed to be the best “path” for a query edge. The reason is that the number of insertions that transform a pair of isolated nodes to a query edge can be equal to or even smaller than the number of deletions that transform a path to a query edge. Therefore, we make the following amendments to support “graph edit distance”:

CHAPTER A. SUPPORT FOR OTHER DEGREE OF MATCHING METRICS

---

1. In the phase of “computation of the best paths”, line 11 in Algorithm 5.3 shall be replaced by the following statements: $p' = \arg \max_{p \in S} \{\text{sim}(e, p)\}$, $p^* = \text{sim}(e, p') \geq \text{sim}(e, \{x, y\})$? $p': \{x, y\}$, where $\{x, y\}$ denotes a pair of isolated nodes (this notation is different from (x, y) , which denotes an edge).
2. In the phase of “partial answer connection discovery”, in Algorithm 5.4, in each step k , now we can add the path p_k without considering whether p_k shares some common nodes with partial answer G_a or not. That is, two greedy algorithms can be unified as: starts from an empty answer graph G_a , and iteratively in step k , inserts into G_a the path p_k that matches an unmatched query edge (u, v) and maximizes $p_k = \arg \max_{\substack{p \in P \\ p \notin G_a}} \{\text{sim}(Q, G_a \cup \{p\}) - \text{sim}(Q, G_a)\}$.

Appendix B

Upper Bound of Optimum

Theorem B.1 Assume that $\mathbf{sim}_{\text{opt}}$ is the degree of matching between the query graph Q and the optimal answer graph. Then, we have

$$\mathbf{sim}_{\text{opt}} \leq \left(\sum_{\substack{e \in E(Q) \\ i=0 \text{ to } |E(Q)|}} \mathbf{sim}(e, p_i^*) \right) / |E(Q)| \quad (\text{B.1})$$

where p_i^* satisfies the following recursive equation:

$$p_i^* = \begin{cases} \emptyset & i = 0 \\ \arg \max_{\substack{p \in (P \setminus (\cup_{k=1}^{i-1} p_k^*)) \\ e \text{ is unmatched}}} \mathbf{sim}(e, p) & i > 0 \end{cases}$$

Proof (Sketch): In step i , $\mathbf{sim}_{\text{opt}}$ chooses path p'_i that results in the best connected global answer graph. However, the value of $\mathbf{sim}(e, p'_i)$, for each individual path p'_i , may not be the largest among that of all the other paths. While on the right side of Equation B.1, we simply choose a path p_i^* that matches an unmapped edge e and that has the maximum value of $\mathbf{sim}(e, p_i)$ among all the other paths. Hence, in each step, we have

$$\mathbf{sim}(e, p'_i) \leq \max_{\substack{p \in (P \setminus (\cup_{k=1}^{i-1} p_k^*)) \\ e \text{ is unmatched}}} \mathbf{sim}(e, p),$$

and therefore Equation B.1 holds. $\square$