A Study of Constrained Reachability Query Processing in Directed Graphs

A thesis submitted to the University of Hyderabad
in partial fulfilment of the requirements for the award of

Doctor of Philosophy

in

Computer Science

by

Bhargavi B

15MCPC02

Under the supervision of

Dr. K. Swarupa Rani

School of Computer and Information Sciences

University of Hyderabad

P.O. Central University, Gachibowli

Hyderabad – 500046

Telangana, India

2020

CERTIFICATE

This is to certify that the thesis entitled **"A Study of Constrained Reachability Query Processing in Directed Graphs"** submitted by **Bhargavi B.** bearing registration number **15MCPC02** in partial fulfilment of the requirements for award of **Doctor of Philosophy** in the School of **Computer and Information Sciences** is a bonafide work carried out by her under my supervision and guidance.

This thesis is free from plagiarism and has not been submitted previously in part or in full to this or any other University or Institution for award of any degree or diploma.

The student has the following publications before submission of the thesis for adjudication and has produced evidence for the same in the form of acceptance letter or the reprint in the relevant area of her research:

1. **Bhargavi B.**, and K. Swarupa Rani, "Finding Frequent Subgraphs and Subpaths through Static and Dynamic Window Filtering Techniques", *EAI Endorsed Transactions on Scalable Information Systems, Vol. 7, No. 27, p. 13, EAI* [**DBLP Indexed and ESCI Indexed**] ISSN: 2032-9407, **Web of Sciences**, 2020. Work reported in this paper appears in **Chapter 5**.

2. **Bhargavi B.**, K. Swarupa Rani, "Implicit Landmark Path Indexing for Bounded Label Constrained Reachable Paths", *International Journal of Recent Trends in Engineering (IJRTE), Vol. 8, No. 4, p. 10, ISSN: 2277-3878, 2019.* Work reported in this paper appears in **Chapter 3**.

3. **Bhargavi B.**, Swarupa Rani K., Rohit Kumar, and Sanmeet Kaur, "Static and Dynamic Techniques to Extract Frequent Subgraphs from Graph Stream Data". To appear in *Proceedings of the 2019 International Conference on Big Data, Machine Learning, and Applications (BigDML-2019).* Work reported in this paper appears in **Chapter 5**.

4. **B. Bhargavi** and K. Swarupa Rani, "Bounded Paths for LCR Queries in Labeled Weighted Directed Graphs", in *Proceedings of Advances in Computing and Data Sciences (ICACDS), Communications in Computer and Information Sciences, Vol. 905, Springer* [**Scopus Indexed**], pp. 124–133, 2018. Work reported in this paper appears in **Chapter 3**.

and has made the presentations in the following conferences:

1. 2019 International Conference on Big Data, Machine Learning, and Applications (BigDML-2019), December 16-19, 2019, NIT Silchar, Assam, India.

2. 2018 International Conference on Advances in Computing and Data Sciences (ICACDS-2018), April 20-21, 2018, Uttaranchal University, Dehradun, Uttarakhand, India.

Further, the student has passed the following courses towards fulfilment of coursework requirement for Ph.D:

Course Code	Name	Credits	Pass/Fail
CS801	Data Structures and Algorithms	4	Pass
CS802	Operating Systems and Programming	4	Pass
CS811	High Performance Computing	4	Pass
AI851	Trends in Soft Computing	4	Pass

Swarupa Rani K
21/12/2020

(Dr. K. Swarupa Rani)
Supervisor
School of Computer and Information Sciences
University of Hyderabad
Hyderabad - 500 046, India

Associate Professor
School of CIS
Prof. C.R. Rao Road,
Central University
Hyderabad-46. (India)

21/12/2020

(Prof. Chakravarthy Bhagvati)
Dean
School of Computer and Information Sciences
University of Hyderabad
Hyderabad - 500 046, India

DEAN
School of CIS
Dr. C.R. Rao Road,
Central University Campus PO
Gachibowli, Hyderabad-46. (India)

DECLARATION

I, **Bhargavi B**, hereby declare that this thesis entitled **"A Study of Constrained Reachability Query Processing in Directed Graphs"** submitted by me under the guidance and supervision of **Dr. K. Swarupa Rani** is a bonafide research work that is also free from plagiarism. I also declare that it has not been submitted previously in part or in full to this University or any other University or Institution for the award of any degree or diploma. I hereby agree that my thesis can be deposited in Shodhganga/INFLIBNET.

A report on plagiarism statistics from the University Library is enclosed.

Date : 21/12/2020

Bhargavi B

Signature of the Student
Name: Bhargavi B
Reg. No.: 15MCPC02

Signature of the Supervisor: Swarupa Rani. K
21/12/2020

Abstract

In today's big data era, a graph is an essential tool that models the semi-structured or unstructured data. Graph reachability with vertex or edge constraints is one of the basic queries to extract useful information from the graph data. In real-time, the vertices of the graph have multiple attributes and different relationships between them. We study different variants of vertex and edge constraints and the techniques to solve the constrained reachability queries.

One of the variants is the Label Constraint Reachability (LCR) query. It finds the existence of a path between the given vertices satisfying the given edge-label constraints. We extend the LCR queries by considering weighted directed graphs and propose a novel technique for finding not only the existence of paths but also the exact paths bounded by given path weight. We propose Implicit Landmark Path Indexing and query processing technique that includes the implicit paths which satisfy the user constraints but need not satisfy the minimality of edge label sets. The problem of Bounded Label Constrained Reachable Paths is challenging as (1) we need to handle exponential number of edge label combinations with an additional total path weight constraint, and (2) we need to discover a technique that finds exact reachable paths between the given vertices. This problem can be applied to real network scenarios like road networks, social networks and protein-protein interaction networks.

Another variant of constrained reachability queries is the problem of finding multidimensional constraint reachable paths. In this problem, we find the path between the given vertices that match the user specified multidimensional vertex and edge constraints. An important challenge is to store the graph topology and attribute information while constructing reachability index. We propose optimized hashing based heuristic search

technique to solve the multidimensional constraint reachability queries. In the proposed technique, we optimized hashing and applied an efficient clustering technique which is based on matrix factorization. Furthermore, we proposed an extended heuristic search technique to improve the accuracy.

During the process of solving constraint reachability queries, we observed the need and importance of finding frequent subgraphs and subpaths. Finding frequent subgraphs from the dynamic graph streams of big data is challenging as streams are non-uniformly distributed and are continuously needed to be processed. From these frequent subgraphs, we can extract unknown and useful information. Its applications include finding strongly interacting groups in social networks and sensor networks and finding frequent molecular interactions to predict protein functions and types of diseases in bio-informatics. To extract frequent subgraphs from graph streams of data, we proposed static and dynamic sliding window techniques. In addition, we applied our proposed static and dynamic techniques to extract frequent subpaths from sequence of paths of a directed graph.

Our contributions are further integrated into a novel query processing framework to solve the constrained reachable paths efficiently. Using our proposed query processing framework, we can store the resultant paths of queries in the Query Path Log (QPL) for future retrieval and also to extract frequent subpaths. The QPL constitutes the query and resultant path information. This log can be used to handle all types of queries such as new, same and similar queries. The frequent subpaths information is used for solving similar queries. We evaluated the performance of the proposed framework and techniques on real and synthetic datasets.

To my Family and Teachers

Acknowledgements

First and foremost, I would like to offer my sincere gratitude to my supervisor **Dr. K. Swarupa Rani** for her invaluable guidance, planning and support during all these years and giving me the freedom to pursue my research in my own way. Without her continual inspiration and support, it would not have been possible to complete this study.

My gratitude also goes to my Doctoral Review Committee (DRC) members **Prof. S. Durga Bhavani** and **Dr. P. S. V. S. Sai Prasad** for their constructive criticism and helpful suggestions that guided me in every aspect of my research.

I take this opportunity to convey my respectful regards to the present Dean **Prof. Chakravarthy Bhagvati** for his encouragement and support. I also thank previous Deans of School of Computer and Information Sciences for providing the necessary resources and a pleasant working atmosphere.

I sincerely thank and admire the contributions of the MCA students, **Ms. Sanmeet Kaur, Mr. Rohith Kumar and Mr. Arunjyothi Neog** of School of Computer and Information Sciences for many useful discussions at University of Hyderabad. I would also like to thank my research scholars of the school for their constant support.

My research would have been impossible without the support of my husband **Mr. B. Vinod Kumar**. His love and support has always been my strength and helped in every important aspect of the Ph. D. journey. I would also like to thank my parents **Mr. B. Venkateswara Rao** and **Mrs. B. Rama Devi**, and in-laws for their understanding and support

in pursuing my career goals and aspirations. Finally, I sincerely thank all my teachers for their constant support, suggestions, motivation and encouragement.

BHARGAVI B

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Graph Mining

Big data constitutes large amounts of data generated from different data sources very fast. The discovery of useful and unknown information in such data is challenging for research and technical groups. Graph is one of the important tools that can represent the complex relationships between the objects of big data. Graph mining refers to extracting knowledge from the data represented as a graph [29]. Some of specific operations of mining graph data from real-world domains such as graph pattern matching, clustering graphs and finding frequent subgraphs are the important contributions of deriving new knowledge. They have broad applications in social networks, biology, chemistry, Resource Description Framework (RDF), image processing and software engineering.

1.2 Graph Reachability

One of the fundamental operations to manage graph data is to find the reachability from one vertex to another vertex in the graph. Let G = (V, E) be a large directed graph that has n vertices and m edges. A reachability query between u and v (u and v are the vertices in G) returns true if and only if there is a path in the directed graph G from u to v. There are two possible approaches to process a reachability query in a graph G. It can be processed on demand using

Breadth-First Search (BFS) or Depth-First Search (DFS) over the graph G. It incurs high cost as O($n + m$) time. On the other hand, it can be processed offline by precomputing and maintaining the edge transitive closure on disk. The former requires too much time in querying and the latter requires too much space. Thus, in the state-of-the-art literature, many reachability techniques are developed that have a trade-off with time and space.

1.2.1 Label Constraint Reachability (LCR)

Many real-world graphs are edge-labelled graphs with edges in the graph having a label from a pre-defined label set. This changes a reachability-query into a Label-Constrained Reachability query or "LCR" query. The question for such a query is: "Can we reach from point A to another point B in the graph using only certain types of edges?" For instance, let us consider road networks of figure 1.1, in which each location can be represented as a vertex. Any location is connected with another location with relationships like road types (narrow, street, two-wheeler and highway). We may explore the graph to find paths between locations connected with only certain types of roads like either street or highway. Ruoming Jin et al. [45] formally defined Label-Constraint Reachability as " Given two vertices, s and d in the edge labeled directed graph G, and a label set A, where s, $d \in V$ and A $\subseteq T_l$ where T_l is a pre-defined label set, if there is a path p from vertex s to d whose path label $L(p)$ is a subset of A, i.e., $L(p) \subseteq$ A, then we say s can reach d with label-constraint A. We also refer to path p as an A-path from s to d".[45]

In other words, "Given two vertices s and d, and a label set A, the label-constraint reachability (LCR) query asks if there exists an A-path from s to d". [45]

1.2.2 Multidimensional Constraint Reachability (MCR)

Multidimensional Constraint Reachability queries are another variant of constraint reachability queries for attributed graphs. Attributed graph is widely used for modeling variety of information networks. In an attributed graph, every vertex can have a set of vertex attributes and their values. Similarly, every edge

can have edge attributes and their values. An MCR query finds the existence of a path from the source vertex to the destination vertex satisfying the given attribute constraints for attributed graphs. The attribute constraints are the conditions on vertex attributes' and edge attributes' values. This problem is applicable for many real-time information networks like social networks, transportation networks and metabolic networks. For instance, consider the attributed graph of figure 1.2. The vertex attributes include *Country* and *IncomeGroup*. The edge attribute is the communication content that is either *"xml"* or *"skyl"*. For example, an MCR query can be to find the existence of path between the vertices 'a' and 'h' with vertex constraints "I, H" and edge constraint "*xml*".

1.2.3 Frequent subgraphs and subpaths

A graph stream is a sequence of graphs that are updated dynamically as streams of edges. Graph streams are used to model streams of semantic web, sensor network, social network and road network data. In this data, there may exist implicit, previously unknown and potentially useful knowledge. One of the techniques to extract such useful knowledge is to find collections of frequent connected edges. From these frequent edges, we can extract frequent subgraphs by satisfying the threshold value. Similarly, we can find the frequent subpaths from the sequence of paths by satisfying the threshold value.

1.3 Solving Constraint Reachability Queries and Finding Frequent Subgraphs

By solving constraint reachability queries, we can find the existence of path between the vertices while satisfying the user specified vertex constraints and edge constraints. One of the varaints of constraints reachability is LCR queries. Maximal spanning tree based index framework [45] is one of the approaches to solve label-constrained reachability queries. Another approach [93] is the path-label transitive closure technique that uses Dijkstra like algorithm over augmented directed acyclic graph. But, these approaches are not scalable and cannot be applied for large and dense graphs to solve LCR queries. Landmark index and query

processing technique [79] is the current state-of-the-art technique that computes partial transitive closure as index for subset of vertices called landmark vertices. Using this partial index, another index is computed, which denotes reachability from nonlandmark vertices to limited number of landmark vertices. It includes path labels information along with the reachable landmark vertices. The resultant index along with BFS is used to solve LCR queries efficiently.

Another variant of constrained reachability queries is MCR queries. Indexing vertex attribute values and edge attribute values using hashing [88], [89] is a prominent indexing technique. Heuristic search technique using naive clustering [88] is developed to solve MCR queries faster for attributed graphs. In the current state-of-the-art literature, many graph clustering techniques are developed that consider graph topology or attribute information or both [92], [84], [37],[3],[83]. By optimizing hashing and adopting an efficient clustering technique, we can further enhance the efficiency of the heuristic search technique [88].

To solve similar queries, finding frequent subgraphs from the graph data is essential. Sliding window technique through direct 1-step algorithm [31] is the state-of-the-art technique to extract frequent connected edges or frequent subgraphs from graph stream data by satisfying the given threshold value. Besides, polynomial strategies and fuzzy techniques [91] are developed to compute actual threshold for finding frequent patterns.

1.4 Challenges of Graph Mining

The following are the current challenges of mining graph data [59], [49]:

- Graph data Integration: Due to heterogeneity of data, there is need to develop graph data systems that support different types of vertices and different types of edges with different attributes with no fixed schema.

- Graph data Visualization: Large graphs or a large group of small graphs should be easily visualized within the screen space of the user and based on the requirements of the user.

- Analysis of dynamic graphs: Most of the graphs like social networks are dynamic graphs in which the related vertices and edges change constantly. Hence, there is need to update the existing graph mining algorithms to support the dynamic graphs. For example, to identify highly influencing groups from Twitter data, products recommendations for users based on their click stream analysis or to identify on real-time the best path for users mobility taking account of traffic events.

- High performance and scalability: Another challenge is to improve the performance and scalability of graph processing and analysis to handle large graphs with billions of entities and relationships.

- User-friendly and Efficient Graph Search Engine: It is extremely challenging to develop a search engine for big graphs such that the user-interface is friendly, the results are accurate and retrieved with high efficiency.

1.5 Motivation

We address some of the challenges of graph mining such as graph data integration by the study of different techniques for handling multiple vertex attributes and edge attributes in solving constraint reachability queries. The challenges and applications that motivate our research are described as follows:

- We identified two significant challenges in finding the Bounded Label Constrained Reachable Paths, one of them is that there can exist an exponential number of label combinations as constraints between the given vertices. Another challenge is to compute the exact paths.

- We observed the need to store both graph topology and attribute information while indexing the reachability to solve constraint reachability queries. Besides, we observed that there is a need to find an efficient attributed graph clustering technique for faster query processing of large graphs.

- Finding frequent subgraphs can extract useful and interesting knowledge in social networks, bio-informatics and IP routing [55]. For instance, we can

derive the groups of users who are frequently communicating in the social network. In bio-informatics, based on the frequent interactions between molecules, we can predict protein functions and identify types of diseases. The applications for finding frequent subpaths can be in IP routing in which we can find the frequent paths of data flows across multiple networks. In a traffic network, we can find the paths/subpaths that are frequently traversed by commuters. Besides, we observed that the developed solutions in the literature have certain limitations which include the partially resolved duplicate calculations. Another possible limitation, frequent subgraphs in the past can be infrequent due to incomplete storage of edges in the sliding window.

These challenges and scope of usage in real-time applications motivate us to study LCR queries and its variants for the development of query processing framework.

1.6 Problem Definitions

The following problems of our interest are addressed in the thesis:

- We extend LCR queries by including path bound which is a real-time constraint and propose novel problem of bounded LCR paths. The path bound constraint refers to maximum allowed path weight for LCR query given by user in addition to edge label constraint. This bounded LCR paths problem is applicable for edge-labeled weighted directed graphs.

- We also solve another problem of multidimensional constraint reachability queries. It considers vertex constraints and edge constraints for attributed graphs. Besides, we extend MCR queries by finding the resultant paths information.

- Once we find paths for constrained reachability queries, we can store and maintain repository with the previous queries and its resultant paths to solve same queries. Besides, we can extract frequent subpaths from the repository to solve the similar queries. We identified the extraction of frequent subpaths by satisfying the threshold value as a special problem. We

related it to the problem of finding frequent subgraphs from the sequence of graph streams.

- We also propose query processing framework by integrating the constraint reachability query techniques along with subpath information.

1.7 Applications

One of the applications of our proposed Bounded Label Constrained Reachable Paths (BLCRP) is in transportation networks. For instance, in road networks, we can find the paths between two places ***X*** and ***Y*** bounded by the given distance ***d***, which are connected through labeled roads. Fig. 1.1 illustrates a local road network with vertices denoting the locations and edges representing the existence of road between the locations/places. Each edge has a label that constitutes two parts ***W:Ty***; ***W*** denotes the distance between the two locations and ***Ty*** denotes the type of roads. For instance, the type of roads are assumed to be Highway (H), Street (S), Narrow (N) and Two-wheeler road (accessible by only two-wheelers) (T) for the Figure 1.1. Suppose the user query is to find the paths from "P1" to "P4" within distance of 100Km in a two-wheeler without using Highway. The query can be considered as the problem of finding BLCRP for the given labeled weighted directed graph, G1, with label set constraint "SNT" and the bound for path weight 100. The resultant bounded label constraint reachable path is { "P1", "P2", "P3", "P4"}.

Multidimensional Constraint Reachability (MCR) queries have wide applications in social networks. For instance, let us consider the attributed graph for an email network as shown in Figure 1.2. Let the vertex attributes be *Country* and *Income Group*. The domain of attribute *Country* is $V_{Country}$={ India (I), United Kingdom (U)} and that of the attribute *Income Group* is $V_{IncomeGroup}$={ High (H), Medium (M), Low (L)}. The domain of edge attribute for communication content is { XML (xml), Skyline(skyl)}. Thus, for vertex *'a'*, $V_{Country}(a)$=*'I'* and $V_{IncomeGroup}(a)$=*'H'*. Similarly, the edge attribute between vertices *'a'* and *'c'* is *"xml"*. Let us consider the MCR query *q1('a', 'j', 'I:H', "xml")*, for the attributed graph of Fig. 1.2. The given MCR query *q1* returns true as the source

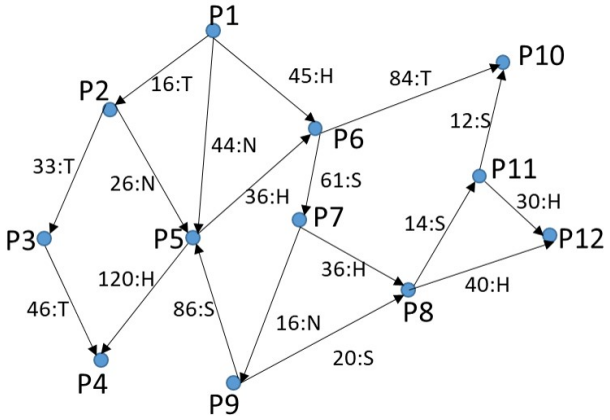


Figure (1.1) Road network, G1 with distances (Km.) and types of roads (S:Street, N:Narrow, H:Highway, T:Two-wheeler only)

vertex 'a' can reach the destination vertex 'j' through vertex 'c' while satisfying the given vertex constraints 'I:H' and edge constraint "xml". Thus, the MCR path is {*'a', 'c', 'j'* }.

1.8 Research Objective and Scope of Thesis

The objective of our research is to efficiently find the existence of path between the given vertices satisfying the given constraints. The constraints can be vertex constraints or edge constraints or both. We perform comprehensive study of LCR queries and their variants to identify optimal solutions that can efficiently

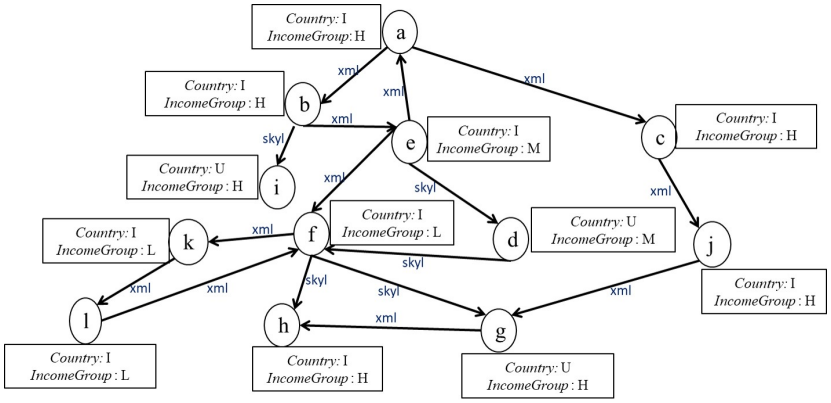


Figure (1.2) Attributed graph

handle the vertex or edge label constraints for reachability queries between the vertices. We identify the issues related to Label Constraint Reachability such as finding an efficient indexing technique. Besides, we identify the different types of constrained reachability queries of real world that demand novel and efficient techniques. Our research aims at finding techniques for different scenarios to find paths for reachability queries with constraints.

In the thesis, we focus on finding paths for LCR queries of edge labeled weighted directed graphs with path bound. The scope of the thesis also includes finding paths for vertex constraints and edge constraints of constrained reachability queries in attributed graphs. In addition, we solve the problem of finding frequent subgraphs from graph streams and finding frequent subpaths from the sequence of paths to solve similar queries. Besides, we propose a novel framework for efficiently finding paths for constrained reachability queries.

1.9 Thesis Contributions

We propose landmark path indexing technique by extending the landmark indexing and query processing technique [79] to find bounded label constrained reachable paths. This technique includes indexing the paths along with labels and path weights for edge labeled weighted directed graphs. Besides, the implicit paths are included that need not satisfy the minimality of edge label sets or Dijkstra's relaxation property. We find the BLCRP by using the proposed landmark path indexing and BFS based query processing.

We adopted the idea of hashing vertex attributes and edge attributes [88] and proposed an optimized hashing based search technique to solve the multidimensional constraint reachability queries. We enhanced the heuristic search technique [88] through including structural and attributed graph clustering based on matrix factorization [37] and proposed an efficient heuristic search technique.

To find frequent subgraphs, we proposed static single-window technique and dynamic sliding window techniques. We also proposed enhancements by extending the proposed static and dynamic approaches with its variations. Besides, we solved the subproblem to extract frequent subpaths from sequence of paths, a special case of the problem by using our proposed techniques.

We integrate our contributions and propose novel query processing framework. Our proposed query processing framework can find paths for constrained reachability queries. In the framework, the proposed techniques of BLCRP and MCR will address the new queries. Besides, using our proposed query processing framework, we can store the query and path information in the Query Path Log to extract frequent subpaths. These stored paths information can be used to handle same queries and similar queries.

1.9.1 Publications of the thesis

1. Bhargavi B., and K. Swarupa Rani, "Finding Frequent Subgraphs and Subpaths through Static and Dynamic Window Filtering Techniques", EAI Endorsed Transactions on Scalable Information Systems, Vol. 7, No. 27, p. 13, EAI [**DBLP Indexed and ESCI Indexed**] ISSN: 2032-9407, **Web of Sciences**, 2020. Work reported in this paper appears in **Chapter 5**.

2. Bhargavi B, Swarupa Rani K., Rohit Kumar, and Sanmeet Kaur. "Static and Dynamic Techniques to Extract Frequent Subgraphs from Graph Stream Data" , to appear in International Conference on Big Data, Machine Learning, and Applications (BigDML), 2019. Work reported in this paper appears in **Chapter 5**.

3. Bhargavi B., K. Swarupa Rani, "Implicit Landmark Path Indexing for Bounded Label Constrained Reachable Paths", International Journal of Recent Trends in Engineering (IJRTE), Vol. 8, No. 4, p. 10, ISSN: 2277-3878, 2019. Work reported in this paper appears in **Chapter 3**.

4. B. Bhargavi and K. Swarupa Rani, "Bounded Paths for LCR Queries in Labeled Weighted Directed Graphs", in Proceedings of Advances in Computing and Data Sciences (ICACDS), Communications in Computer and Information Sciences, Vol. 905, Springer [**Scopus Indexed**], pp. 124133, 2018. Work reported in this paper appears in **Chapter 3**.

5. Bhargavi B., and K. Swarupa Rani, and Arunjyothi Neog, "Finding Multidimensional Constraint Reachable Paths for Attributed Graphs", Applied

Intelligence, Springer, [**SCI Indexed**], **Web of Sciences**, 2020 [Communicated and received first level reviews]. Work reported in this paper appears in **Chapter 4**.

1.10 Organization of Thesis

The thesis is organized as follows: Chapter 1 briefly introduces the problems addressed in constrained reachability queries along with our contributions. Chapter 2 describes the survey of reachability techniques, different constraint reachability techniques, LCR techniques, and attributed graph clustering techniques. Besides, we discuss state-of-the-art techniques to find frequent subgraphs and frequent subpaths. In Chapter 3, we propose a novel problem of finding Bounded Label Constrained Reachable Paths in edge labeled weighted directed graphs. We then describe our proposed solution with experiments evaluated on real and synthetic benchmark datasets. In chapter 4, we solve Multidimensional Constraint Reachability queries using our proposed matrix factorization-based heuristic search techniques and validate the efficiency by experiments and analysis. Chapter 5 deals with finding frequent subgraphs and frequent subpaths by using our proposed static and dynamic approaches. In chapter 6, we integrate our contributions and propose a novel query processing framework. In chapter 7, we summarize our contributions with the conclusions and also provide future directions of our research.

Chapter 2

Literature Survey

Graph is a powerful modeling tool. Graph mining is the extraction of interested knowledge from the data represented as graph. We identified supergraph search [58], constraint based graph reachability [88], graph pattern mining [9], frequent subgraphs [31], community detection [37] and graph partitioning [70] as the trending problems in graph research. They have broad applications in social networks, biology, chemistry, Resource Description Framework (RDF), image processing and software engineering.

In this chapter, we review various techniques through 5 C's by Citing state-of-the-art literature, Comparing, Contrasting, Critiquing and Connecting with our research pertaining to the problems of our interest. Table 2.1 describes the extensive study of our problems of interest. Section 2.1 describes the different types of graph queries in general, while section 2.2 discusses about the current state-of-the art reachability techniques. Section 2.3 describes techniques to find shortest paths, and constrained paths. In section 2.4, we describe the different label constrained reachability techniques. Also, we survey on other constraint reachability techniques in section 2.4.1. Besides, we review Multidimensional Constraint Reachability (MCR) techniques and community detection techniques for attributed graphs in section 2.6. Furthermore, we review techniques that find frequent subgraphs from graph stream data and frequent subpaths from sequence of paths in section

2.8. We also briefly reviewed about query logs and framework development in section 2.10.

2.1 Queries in Data Graphs

In general, there are two types of queries for data graphs [38] : (1) pattern match query and (2) reachability query. Pattern match query finds the existence of patterns that match the user given graph pattern. This pattern match query can be solved by finding subgraphs that are isomorphic to the given graph pattern. Supergraph search [58] is one of the similar problems to that of graph pattern matching.

Another kind of solving graph pattern query is based on reachability and using join based techniques [21]. Reachability query is a special case of pattern match query [38] where we find if there exists path between the two given vertices. Reachability queries can be solved based on full index or partial index based techniques discussed in the next section. Shortest path query requires to find the shortest path between the given vertices. In this thesis, we focus on reachability queries wih constraints and constrained path queries and discuss the techniques to solve them.

2.2 Reachability Techniques

Graph reachability finds the existence of path between the vertices in a graph. There are two possible approaches to process a reachability query in a graph G. It can be processed on demand using Breadth-First Search (BFS) or Depth-First Search (DFS) over the graph G. It incurs high cost as O(n + m) time. On the other hand, it can be processed offline by precomputing and maintaining the edge transitive closure on disk. The former requires too much time in querying and the latter requires too much space. Thus, in the state-of-the-art literature, many reachability techniques are developed

that have a trade-off with time and space. Beamer et al. [19] developed direction optimizing breadth first search technique which combined top-down BFS and bottom-up BFS. In the hybrid algorithm, search began with top-down approach and continued untill the frontier is too large, at which point bottom-up search is started. This approach can be applicable to find reachability online between vertices. This work accelerated the processing of parallelizing BFS.

In [87], [9], detailed survey of reachability techniques is specified. H. Wei et al. [82] classified the reachability techniques into following two categories.

- Label-only approaches
- Label+G approaches

Label-only approaches directly find the existence of reachability between the given vertices from the constructed index. Label-only approaches include 2-hop [28], 3-hop [47], Chain-cover [27], Path-Tree Cover [48], Tree-Cover [10], and Dual Labeling [80].

Label+G approaches use index labels and BFS/DFS to find the reachability between the given vertices. Tree+SSPI [25] and GRIPP [82] are Label+G approaches which have linear index size and query time in terms of the number of vertices of the graph. H. Wei et al. [82] developed an independent permutation labeling approach with high probability guarantee which is Label+G approach to find the reachability between the vertices. Besides, they introduced level labels and huge vertex labels to handle reachability for dense graphs.

2.3 Path Finding Techniques

In this section, we explore the different techniques that find paths between the vertices. In the literature, many techniques are developed that find the

Table (2.1) Survey of Constrained Reachability Query Processing [2005-2018]

S. No.	Technique (Authors, Year)	Observations
1	MST-based index and query processing (Jin et al., 2010 [45])	The tree-based index framework used to solve LCR queries is not scalable
2	LandmarkIndex and Query algorithms(Valstar et al., 2017 [79])	Found reachability satisfying given label constraint with significant speedups in query processing for LCR queries
3	Spanning Tree based BPFilter algorithm (Minghan Chen et al., 2014 [26])	Generated subpath graph and approximate reachability is found in uncertain graphs for LDCR queries
4	Heuristic Search Technique through GuidedBFS and hashing(Duncan Yung et al., 2016 [88])	Developed heuristic search technique and computed reachability with multi-dimensional constraints in attributed graphs faster
5	ANCA Clustering (Falih et al., 2018 [37])	Computed clusters in attributed graphs considering both graph topology and attribute information through matrix factorization
6	Edge Induced Matrix Factorization (Qi et al., 2012 [65])	Used matrix factorization based technique on edge content to detect communities
7	Cluster Affiliation Model for Big Networks (Yang et al., 2013 [85])	Developed the algorithm to detect overlapping communities through non-negative matrix factorization
8	Ensemble gradient descent algorithm based on matrix factorization (Amin et al., 2017 [12])	Identified polarization and clusters in social networks specifically Twitter through matrix factorization
9	Probabilistic approach to find dense patterns in graph streams (Aggarwal et al., 2010 [8])	Formally defined graph streams and developed the probabilistic min-hash approach to mine dense patterns from graph streams
10	Direct 1-step algorithm (Cuzzocrea et al., 2015 [31])	Used DS Matrix and sliding window technique to extract frequent subgraphs from graphstreams
11	Sliding Window Filtering (SWF) algorithm (Lee et al., 2005 [52]	Developed sliding window filtering algorithm with relative support to extract frequent patterns and the resulting association rules
12	Polynomial and fuzzy strategies for actual minimum support (Zhang et al., 2008 [91])	Computed actual minimum support from the user-specified minimum support for frequent pattern mining

exact paths, approximate shortest paths, bounded paths and paths that satisfy constraints between vertices.

Chris Barrett et al. [17] defined edge label and edge weight constraints in formal language and also defined the formal language constrained shortest path and simple path problems. They developed a general approach that model the constrained path problems based on Nondeterministic Finite Automaton (NFA) to find the constrained paths. They used dynamic programming technique to further improve the performance. They proved that the problem of finding a simple path between a source and given destination is NP-hard when the label set is restricted to even simple classes of graphs like complete grids.

The constrained TreeSketch algorithm of Ankita et al. [57], considered edge label constraint (A) and found approximate A-paths between the given vertices. They formally defined label constrained shortest paths problem as the shortest path between the given vertices such that the edge labels involved in the path are subset of given labelset constraint. They extended labeled path sketches by adding edge label information and designed Constrained TreeSketch algorithm. This algorithm mainly involved BFS traversal through tree of constrained paths of source vertex and destination vertex to find the approximate shortest paths. This technique is applicable to label order constrained shortest path queries. Minghan Chen et al. [26] worked on uncertain graphs with their new sampling techniques to find approximate shortest paths constrained through distance parameter.

Thorup et al. [76] studied the distance queries problem which involved finding the distance between an arbitrary pair of vertices. They have developed approximate distance oracles by modifying Dijkstra's algorithm and designing index that used finite stretch to settle for approximate distance instead of exact one. These oracles had constant query time and are applicable for weighted undirected graphs. The distance query algorithm can be

extended to find the path whose length is atmost given distance based on lowest common ancestor algorithm. But, stretched distances may not be acceptable under all scenarios.

In [11], Akiba et al. studied the distance queries problem. They developed a landmark-based approach with efficient pruning to solve the queries to find the exact shortest path distance between the vertices. In the pruned landmark labeling approach, the landmark vertices are selected based on highest degree criterion. For every landmark vertex, BFS is performed and distance to the reachable vertices are indexed. Pruning is performed during BFS when there is an intermediate reachable vertex w between the vertices v and u whose sum of distances from v and u is lesser than total distance from v to u via another path. Thus, the vertex u is not added to label of v and any edges from u are not traversed along the path with longer distance. Moreover, Akiba et al. developed another labeling scheme that used bit-level parallelism on pruned labeling method to improve the performance. Their approaches can be easily extended to find shortest path between the vertices by storing the parent of every reachable vertex in labels and performing backtracking.

In [32], Delling et al. developed a scalable solution based on 2-hop labels for solving the distance queries in large networks. They have designed an exact algorithm which is based on hierarchical hub labeling and a special kind of 2-hop labeling. They introduced the label representation that finds good ordering of subset of vertices using which labels are computed and also compressed. They presented a token based compression that transforms labels into trees to achieve higher compression.

Su et al. [72] considered the more generalized problem of finding paths with desired bounded path lengths in acyclic networks. They developed a path length model of acyclic networks which transformed paths into simple parameters of vertices and arcs. The parameters are designed based on

shortest paths, longest paths and intermediate edges. They designed a simple polynomial algorithm based on the parameters of path length model to find the desired bounded paths.

Erez et al. [35] addressed the problem of automation of constrained clock routing in Integrated Circuit networks by considering the transistors as vertices, wires between them as edges and delay across them as edge-weights. Thus, this problem is reduced to finding bounded paths in a grid graph. They reduced the problem to bit vector logic by storing active neighbours as connectivity constraints using SAT-based bit-vector SMT (SATisfiability Modulo Theories) solver. This solver is further enhanced by a novel graph aware solving approach based on Dijkstra's algorithm and core decision strategies for graph-aware conflict analysis.

Bast et al. [18] performed an extensive survey of different techniques for route planning in transportation networks which had trade-offs between preprocessing effort, space requirements and query time. They have explored shortest path techniques for static networks that find the length of shortest path. These techniques include basic techniques based on Dijkstra's algorithm, goal-directed techniques such as A* search, separator based techniques such as Customized Route Planning algorithm, hierarchical techniques, bounded hop techniques such as hub labeling and combinations of some of these techniques. Bast et al. observed that the actual paths can be found from these techniques by either storing parent vertex information or building shortcuts. They investigated the applicability of these techniques to dynamic networks. They identified the techniques and use of finding alternate paths by either concatenation of shortest paths or compactly representing as a small graph. In this paper, journey planning in a multi-modal scenario is also discussed. It included the label-constrained shortest path approaches, combining costs and multicriteria optimization.

In [73, 74], a B+ tree index based solution was used to solve path queries. Path queries are specified by projections on label-paths over the given label set in graphs. Sumrall et al. [74] developed a path index based on B+tree to accelerate query processing to find paths. It is a workload based index that involved updating index with efficient joins based on previous queries. This index supported ordered access to paths and designed for external memory storage and retrieval. Besides, they used δ compression scheme for gap bits to further reduce key size in leaf nodes of B+tree. But, the limitation of this approach is only paths upto limited length can be indexed using B+tree.

Besides, we explored the centrality measures like degree, betweenness centrality, eigen vector centrality, pagerank and closeness centrality [64] and their usefulness in choosing highly central nodes [62]. Qin et al. [67] investigated the influence of an edge in the graph that lead to reachability changes in the graph brought by possible deletion of the edge in the case of dynamic graphs.

2.4 Label Constraint Reachability Techniques

The Label-Constraint Reachability (LCR) query which was first formally defined by Ruoming Jin et al. [45] is for finding the existence of the label-constrained reachable paths in an edge labeled directed graph. Ruoming Jin et al. [45] developed tree-based index framework where partial transitive closure and spanning tree are used. In this approach, maximal spanning tree is constructed for the directed graph using Chu-Liu/ Edmonds algorithm [34]. The maximal spanning tree would cover the maximum possible reachable paths in the graph. Partial transitive closure index is constructed considering paths whose starting edge and ending edge are non-tree edges. The path labels computed from spanning tree and the partial transitive closure index are used to solve LCR queries. But, this technique is not efficient and scalable for large and dense graphs.

Wenfei Fan et al. [38] developed bidirectional BFS technique to solve graph pattern matching queries, in which one of the special case is label-constrained reachability queries. Bidirectional BFS involved storing adjacent nodes based on label-constraint set from source node in one list and from destination node in another list and checking if there is any match in both the lists. If there is no match, adjacent node from the smaller list is searched recursively based on BFS technique till match is found. This technique is not scalable for large graphs.

Zou et al. [93] solved LCR queries based on augmented Directed Acyclic Graph (DAG). The local transitive closure index is computed for the augmented DAG to solve LCR queries. Besides, they developed partition based DFS technique for large graphs. In Zou et al. [92] technique, the edge-labeled graph is transformed into DAG by computing Strongly Connected Components (SCC) for the graph. The edge labels within SCC are combined to form the edge label of every edge in DAG. A Dijkstra-like method is used to compute single source transitive closure for every vertex by storing only minimal labels and paths for all the reachable vertices. The transitive closure of the entire graph is computed by extending the index to the vertices of SCCs. This transitive closure method had good performance, but it had huge offline processing cost for large graphs. Besides, they developed another technique in which a large graph is partitioned into subgraphs and for every local vertex, transitive closure is computed by creating augmented DAG to preserve the labels of nodes. Then, label constrained DFS and index is applied on partitioned graph to solve LCR query. This technique has greater index construction time and is not effective on graphs with relatively large strongly connected components.

Valstar et al. [79] technique of landmark based query processing is the current state-of-the-art technique to find the existence of reachability for LCR queries. The landmark vertices are selected based on criteria such as the top 'k' highest degree or any of the centrality measures of the graph.

In [79], while constructing the landmark index for the landmark vertices, the edge label sets minimality is considered and landmark vertices are selected based on highest total degree criterion. The index is constructed for landmark vertices using BFS and forward propagation of indexed landmark vertices. For the remaining vertices, reachability of upto 'b' landmark vertices are indexed [79]. These indices along with BFS is used to solve LCR queries.

2.4.1 Other constraint reachability techniques

Different variants of reachability are formed by incorporating constraints to vertices and edges of the graph. The vertex/edge constraints can be a specific bound for each edge weight, specific vertex labels, specific edge labels, membership of specific edge labels, membership of specific vertex labels, vertex labels in a specified order, edge labels in a specified order and bound on the entire path between the vertices. Reachability techniques such as 2-hop cover cannot be directly applied to solve the constraint reachability queries. This is because the vertex or edge attributes information is not stored while indexing. Hence, there is need to explore the different constraint reachability techniques.

Bonchi et al. [24] studied the problem of efficient approximation of shortest-path queries with edge-label constraints for undirected graphs. They developed indexes based on the idea of landmarks. The distance from all vertices of the graph to a selected subset of landmark vertices is computed based on shortest path minimality and is indexed. This index is further optimized through pruning search space by skipping unnecessary label sets. They developed an online query processing technique that used this index to find the approximate shortest path distance between the given vertices while satisfying the edge label constraints.

Miao Qiao et al. [66] studied the problem of weighted constrained reachability for weighted undirected graphs. Given two vertices and a range constraint on edge weight, the Weighted Constrained Reachability (WCR) query finds the existence of path between the vertices such that every edge weight along the path satisfies the given range constraint. Miao Qiao et al. developed efficient memory based algorithms and disk based algorithms by exploiting the cut property of Minimum Spanning Tree (MST). MST is built using Kruskal's algorithm with union-find technique [30]. According to cut property [30] of minimum spanning tree, for any cut C in the graph, if the weight of an edge e $\in$ C is smaller than weights of any other edges in C, then this edge belongs to all MSTs of the graph. By using this property, an edge-based index tree is constructed from the MST by choosing edge with maximum weight as root node and leaf nodes being the vertices of the graph. During query processing, the lowest common ancestor node of source vertex and destination vertex is found from edge-based index tree satisfying the range constraint. If such vertex exists, then WCR query returns true. Miao Qiao et al. further enhanced it by developing an I/O efficient disk based algorithm in which rebalancing MST is performed. They compared their algorithms with baseline approaches like BFS, DFS and LCR technique [45]. The LCR technique is applied by converting LCR query into WCR query. We observed that LCR problem is much harder than that of WCR.

Ruoming Jin et al. [46] investigated the problem of distance constraint reachability in uncertain graphs. For the user-defined distance constraint, this problem finds the probability that the distance between the given vertices is less than or equal to the given distance threshold. They solved this problem by computing possible subgraphs that satisfy the given constraints. They introduced a unified unequal probabilistic sampling estimation framework for finding possible subgraphs to significantly reduce the estimation variance.

Minghan Chen et al. [26] defined the Label and Distance Constraint Reachability (LDCR) problem over uncertain graphs. The LDCR problem is different from LCR problem, as it included querying for paths whose labels must contain all labels in the label constraint set. Given two vertices u and v, distance constraint and a label set, the label and distance constraint reachability problem computes the probability of paths for which the vertex v is LD-reachable from u. Minghan Chen et al. developed DFS based algorithm to compute subpaths between the given vertices. The subpaths that do not satisfy the given distance threshold are pruned. The subpaths with common path expressions are merged based on lowest common ancestor algorithm. Divide-Conquer tree (DC-tree) adopted from [46] for the resultant subpaths is built for which some of the leaf nodes are LD-paths. Besides, they developed branch path pruning algorithm, to remove subpaths that do not reach the given destination vertex. To perform quick and efficient approximations, DC-tree sampling techniques are developed. They have used an unbiased sampling estimator, i.e. Yates-Grundy Sen estimator [86] that avoids sampling the same nodes.

In[43], the partition replication method, workload prediction method, and workload balancing method addressed the data locality and workload balancing issues while finding reachability with node label constraints in large attributed graphs for distributed environment.

2.5 Identified Research Gaps of LCR

From the literature review, we observed that there is a scope for finding efficient reachability techniques that have lesser index construction time, lesser index size and faster query processing. The existing reachability techniques cannot be applied directly to constrained reachability queries as the attributes of vertices/edges are not stored while computing the index. Many approximate shortest path techniques do not compute exact paths. Hence, there is a need to find techniques that compute the exact reachable

paths satisfying the given vertex/edge attribute constraints. Hashing is found to be an efficient data structure for secondary storage access than B+tree [39]. We observed that dynamic programming is more complex than Dijkstra's algorithm and critical path method for finding paths in large scale networks[72].

We observed that the landmark indexing and query processing technique (Valstar et al. [79]) is current state-of-the-art literature and a scalable solution for solving LCR queries in edge-labeled directed graphs. Besides, we are motivated by the inclusion of bounded path weight constraint in real-time scenarios [35, 72]. These observations from the literature led to novel problem of bounded paths for LCR queries which is described in detail in Chapter 3.

2.6 Techniques in Attributed Graphs

In this section, we describe the survey related to techniques to solve Multidimensional Constraint Reachability (MCR) queries and graph clustering techniques for attributed graphs.

2.6.1 MCR techniques

An attributed graph acts as an efficient modeling tool to represent information networks [88] [81]. Ho et al. [43] investigated on processing node-label constrained reachability queries in distributed environment for attributed graphs. They addressed data locality and workload balancing issues of distributed processing that reduced the communication overhead and improved efficient cluster usage respectively by developing partition-replication, workload prediction and balancing techniques. We observed that the developed techniques used Ford-Fulkerson algorithm of flow networks [30].

Sakr et al. [68] developed G-SPARQL, a query execution engine with the defined algebraic operators on the graph by using join operations to find the reachability for large attributed graphs. They designed an efficient hybrid representation to store the topology of the graph in main memory and access the attributes of the graph from the secondary memory. The attributes from the secondary memory are stored in fully decomposed model which included unique table for storing the unique vertex and edge attributes. But, this technique is not scalable when number of vertex and edge attributes are very large.

Yung et al. [88] developed hashing based index for finding constrained reachability in attributed graphs. The hashing based index involved assigning unique hash index for group of vertex attributes or edge attributes. They used non-cryptographic hash function like Murmur hash function [2] for hashing which has almost no collision. The unique group of attribute values of all vertices and the corresponding hash values are stored in primary memory. The attribute values of every vertex and edge are stored in the relational database and are retrieved only when there is hash collision. They designed heuristic search technique based on GuidedBFS using naive clustering to traverse across the graph regions from the source vertex that are likely to reach destination to solve the MCR queries. We observed that the use of naive clustering is not efficient as the computation of probability cost considering attribute values during search is vaguely mentioned. Hence, there is need to find efficient clustering technique that can be used in solving multidimensional constraint reachability for attributed graphs.

2.6.2 Clustering techniques

Many graph clustering techniques are developed in the literature that are based on the topology of the graph, attribute similarity in the graph or both. In this subsection, we discuss the attributed graph clustering techniques that consider both graph topology and attribute information while clustering.

Zhou et al. [92] developed Structure and Attribute clustering (SA Clustering) that finds the clusters for an attributed graph based on both graph topology and attributes of the graph. SA clustering involved construction of an augmented attributed graph and computation of random walks from the augmented attributed graph. SA clustering is limited to small networks with few attribute values.

Xu et al. [84] developed Bayesian model based approach to cluster attributed graphs. But, this approach is found to be slow and not scalable.

Z. Wu et al. [83] developed Structure and Attributes using Global structure and Local neighborhood features (SAGL) clustering. SAGL clustering considered both global structure and local neighbours and assigned different weights to different topological links. SAGL clustering technique is faster than SA clustering as the former technique doesn't increase the size of attributed graph, yet uses both global importance of the vertex and attribute information to find clusters. We observed that although SAGL clustering is faster than SA clustering, to determine attribute similarity, SAGL clustering adopts voting mechanism similar to that of SA clustering. This leads to further construction and use of augmented attributed graph.

Falih et al. [37] observed that social networks are dense and hence require high attribute similarity factor whereas road networks need a balanced attribute similarity and topological similarity metric while computing node similarity. Topological distance metric can be categorized into neighborhood based metric and path based metric. Based on type of attribute data (categorical/numerical/binary), the attribute similarity measure can be, in general euclidean distance computed between pair of vertices [33]. Falih et al. [37] developed ANCA clustering algorithm by considering shortest path metric for topological measure and Euclidean distance for attribute similarity. Then, matrix factorization is applied on both topological and

attribute similarity measures. Finally, they used k-means clustering on the resultant matrix to form k clusters.

Guo Qi et al. [65] used matrix factorization based technique on edge content to detect communities. Yang et al. [85] developed non-negative matrix factorization based model to identify disjoint or overlapping communities at large scale. Alim et al. [12] developed matrix factorization and gradient descent based technique to identify polarization and clusters in social networks like Twitter.

2.7 Identified Research Gaps of MCR

From the literature, we observed that matrix factorization is a standard technique that has scope to find similarity by considering graph topology as well as node /edge attributes. We identified the hashing based heuristic search is the current state-of-the-art literature that can provide scalable solution for multidimensional constraint reachability (MCR) for attributed graphs. We also observed that there is need to reduce the missing constrained reachable paths due to the heuristic [88]. Hence, we adopted the clustering that used matrix factorization to heuristic search to solve the problem of MCR queries using optimized hashing efficiently described in Chapter 4 of the thesis.

2.8 Techniques for Frequent Subgraphs and Subpaths

Massive graphs are considered as streams of data to analyze and extract useful information. Henzinger et al. [42] were the first to introduce graph streams and they also considered graph problems of paths and connectivity.

Aggarwal et al. [8] determined frequent and dense patterns in graph streams. They defined graph streams as sequence of edge sets. They assumed that the graph constituted large number of nodes, but the edge sets contained only small fraction of the nodes.They designed a probabilistic approach based on node co-occurrence and edge density by utilizing the sparsity property of underlying graphs. They used min-hash approach to summarize the graph streams and extract dense patterns efficiently.

Andrew McGregor [61] presented a detailed survey of graph streams. Due to the dynamic nature [40], [44] and the large volume of graph stream data, Nan Tang et. al. [75] proposed graph summarization sketch that can store frequent counts and paths of graph streams.

Alfredo Cuzzocrea et al. [31] studied various methodologies of mining dense patterns in graph streams and proposed probabilistic algorithms for determining such structural patterns effectively and efficiently. They presented two algorithms to extract frequent subgraphs - (i) Indirect 2-step algorithm (ii) Direct 1-step algorithm using Data Stream Matrix (DS Matrix) and sliding window technique. DS Matrix stored the existence of edges in bit vectors. The sliding window tracked the latest window of graph streams from which frequent singleton edges are extracted. Their experimental results showed that their techniques with DSMatrix consumed less memory.

Kyoungsoo Bok et al. [23] observed that the algorithm proposed by Alfredo Cuzzocrea et al. [31] had the limitation of duplicate calculations. They introduced *slidenum* variable [23] to store the frequency of edges incrementally for batches of graph streams to resolve duplicate calculations.

Leung et al. [55] investigated on mining frequent subgraphs from streams of uncertain data. They used Data Stream Matrix and expected support to find frequent connected edges. The expected support is computed from the product of existential probability of edges. They developed uncertain

frequent trees instead of Frequent Pattern trees (FP-trees) and direct 1-step algorithm to extract the frequent subgraphs from the uncertain data.

We observed that finding frequent subpaths from paths is another problem in the literature that can be related to the problem of finding frequent subgraphs. Sumanta Guha [41] developed Apriori based technique to extract frequent subpaths from paths in an undirected graph. Schwartz et al. [69] studied demand of frequent subpaths in a transportation network traversed by several users. Hence, there is need to find techniques that discover frequent subpaths efficiently by storing useful historical information.

2.9 Identified Research Gaps of Frequent Subgraphs

From the literature, we observed that the direct 1-step algorithm [31] that used DS Matrix is an efficient technique to find the frequent edges and the resultant frequent subgraphs. Besides, we observed that while finding frequent subgraphs, although sliding window based techniques execute fast, they may lead to loss of useful historical information. This motivated us to find the actual minimum support from the given minimum support [91] and develop static and dynamic sliding window filtering techniques [52].

We also observed that we need to reduce duplicate calculations further. We related the problem of finding frequent subgraphs to the problem of finding frequent subpaths from sequence of paths. Thus, we applied the proposed frequent subgraph finding techniques to find frequent subpaths from the sequence of paths described in Chapter 5 of the thesis.

2.10 Query Logs and Framework

Query logs are used to find the previous repeated queries and its results faster. Bonifati et al. [14] analysed query logs for Resource Descrition Framework (RDF) queries. They have extracted syntactic structure as well as semantic hypergraph representation of queries. The study on large number of queries led to the queries classification and their shape analysis.

A framework involves integration of components or modules to solve the specific problem efficiently. Yusoff et al. [90] define framework as generic combination of data and processes where subcomponents may be substituted. An architecture is defined as the combination of data and processes where subcomponents are not substituted.

Petrou et al. [36] developed big data framework that involved trajectory prediction algorithms in aviation and maritime domains. The framework involved different modules with batch processing and stream processing layers. The major modules include synopses generator, semantic integrator, data manager, trajectory clustering and future location predictions with demonstrations.

2.11 Developing Query Processing Framework

We observed that we can store the query information and path information for constrained reachability queries in Query Path Log for faster query processing. We find that there is a need to combine our proposed techniques and Query Path Log to develop a novel framework for constrained reachability queries.

The description of different modules and their integration into a novel query processing framework is discussed in Chapter 6 of the thesis. Table 2.1 describes a summary of the prominent techniques and significant observations in the literature that are reviewed by Citing, Comparing, Contrasting, Critiquing and Connecting with the research corresponding to constrained reachability query processing in the thesis.

2.12 Summary

We performed extensive survey of LCR techniques, other constraint reachability techniques, clustering techniques and techniques to find frequent subgraphs from graph streams. We identified the research gaps and progressed further by proposing techniques that are presented in the subsequent chapters of the thesis as contributions. Besides, we integrated our contributions and developed a novel query processing framework presented in Chapter 6.

Chapter 3

Bounded Label Constrained Reachable Paths

One of the fundamental operations to manage graph data is to find the reachability from one vertex to another vertex in the graph. In real-time, the vertices and edges of a graph consist of attributes. In this chapter, we solved and formulated the novel problem by extending the problem of Label Constrained Reachability by developing efficient techniques. The publications of this chapter are listed below. In section 3.2, we describe and illustrate the preliminaries that are helpful in understanding our research. In section 3.3, we describe the techniques in the literature related to reachability, constrained reachability, and finding paths. In sections 3.4 and 3.5, we describe our contributions, which constitutes the proposed landmark path indexing and query processing algorithms. Section 3.6 describes the index construction, evaluation of our proposed techniques based on accuracy measures and statistical analysis on datasets.

B. Bhargavi and K. Swarupa Rani, "Bounded paths for LCR queries in labeled weighted directed graphs", in Proceedings of Advances in Computing and Data Sciences, Springer ,pp. 124–133, 2018.

Bhargavi B. and Swarupa Rani K., "Implicit Landmark Path Indexing for Bounded Label Constrained Reachable Paths", International Journal of Recent Technology and Engineering (IJRTE), Vol. 8, Issue 4, pages 10, pp. 10661-10669, November 2019.

3.1 Introduction

In the big data era, there are huge databases with relational and graphical information. The discovery of useful and unknown information in such data is challenging for research and professional groups. The graph is one of the important tools that can represent the complex relationships between the objects of big data. Graph mining refers to mining data represented as a graph [29]. One of the challenges is to develop algorithms that can store, manage and provide analysis over a large number of graphs for the real-world applications. Another challenge is to develop efficient graph database systems. Neo4j [5] and InfiniteGraph [6] are some of the graph databases optimized for handling graph data. In addition, big data companies like Twitter and Google designed graph database systems such as FlockDB [7] and Pregel [60] respectively. Some of the specific operations of mining graph data from real-world domains such as graph pattern matching, clustering graphs, and finding frequent subgraphs are the important contributions of deriving new knowledge.

One of the fundamental operations to manage graph data is to find the reachability from one vertex to another vertex in the graph. In real-time, the vertices and edges of a graph consist of attributes. These attributes give information about the type of vertices, type of relationship and strength of the relationship between the vertices. For instance, consider road network of figure 3.1 where the vertices denote the loactions or places and the edges denote the link between the locations. The edge weight denotes the distance between the locations. The edge labels denote the type of roads such as Narrow, Street, Two-wheeler and Highway. The constrained reachability query finds the existence of reachability between the two given vertices while satisfying the given constraints. The Label-Constraint Reachability (LCR) query that was first formally defined by Ruoming Jin et al. [45] is for finding the existence of the label constrained reachable path between the given vertices in an edge labeled directed graph.

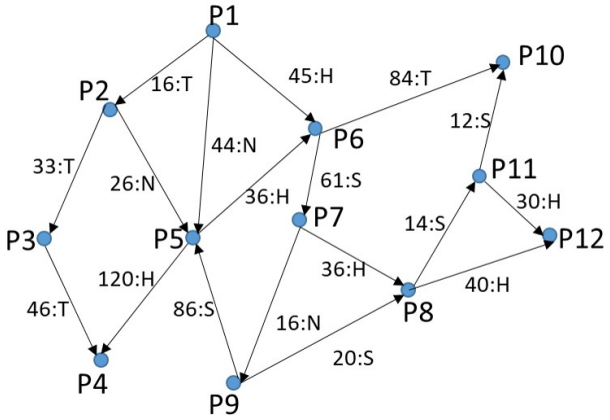


Figure (3.1) Road network (G1) denoting an edge labeled weighted directed graph

We extended the LCR query by formulating a novel problem formally referred as Bounded Label Constrained Reachable Paths (BLCRP) defined in Section 3.2. It involves finding the exact paths satisfying the given constraints instead of finding only the existence of paths [45].

3.1.1 Applications and challenges

One of the applications of BLCRP is in road networks. For instance, in road networks, we can find the paths between two places $\boldsymbol{X}$ and $\boldsymbol{Y}$ bounded by the given distance $\boldsymbol{d}$, which are connected through labeled roads. Fig. 3.1 illustrates a local road network with vertices {P1, P2, ..., P12} denoting the locations or places and edges representing the link between the adjacent locations. Each edge has a label that constitutes two parts $\boldsymbol{W:Ty}$; $\boldsymbol{W}$ (*edge weight*) denotes the distance between the two locations in Kms. and $\boldsymbol{Ty}$ (*edge label*) denotes the type of roads. For instance, the type of roads are assumed to be Highway (H), Street (S), Narrow road (N) and Two-wheeler road (accessible by only two-wheelers)(T) for the Figure 3.1. Suppose the user query is to find the paths from "P1" to "P4" within distance of 100Km in a two-wheeler without using High-way. The query can be considered as the problem of finding BLCRP for the given labeled weighted directed

graph, G1, with label set constraint "SNT" and the bound for path weight 100.

Another application is in protein-protein networks [63] where the edge weight represents the reliability of interaction between two proteins and edge labels are enzymes that transform the proteins. In this case, the BLCRP query can find the transformation path from one protein to another within the given bound. Furthermore, in social networks, the vertices denote the users, the edge labels between the users represent the relationships like *isFriend*, *isRelative* or *isColleague* and the edge weights can be the strength of the relationships. Thus, the BLCRP query, in social network, can find the paths between the given remote users within the constraints on relationships.

One of the significant challenges to find the BLCRP is that there can exist an exponential number of label combinations between the two given vertices. Another challenge is to compute the exact paths. But, in the literature, shortest path finding techniques [57] exist that compute only the approximate paths. These observational studies persuade us towards the real network context that considers the categorical edge label constraints and real-valued edge weight constraints while finding the exact paths.

In this chapter, we focused on edge labels and total path weight constraints for labeled weighted directed graphs. (1) The edge labels must satisfy the given membership constraint while finding graph reachability. (2) We find not only the reachability between the two given vertices but also the exact paths whose path weight is less than or equal to the given maximum path weight. Considering the above two constraints are challenging, as it leads to the slower index construction time.

We formulated the problem and solved Bounded LCR Paths by extending the Landmark Index and Query algorithms [79] and proposed novel algorithms. We extended by incorporating paths and path weights in the index in addition to the reachable vertex and the path label [79] for the landmark vertices. By including minimality of label sets [79], a new index is constructed that is termed as Landmark Path Index [20]. Besides, we also considered Dijkstra's relaxation property for specific cases while indexing.

We also observed that some of the implicit paths were not included in the index that satisfy the given constraints. In order to strengthen the efficiency of our proposed approach [20], in this chapter, we modified the proposed algorithm [20] and improved it by constructing an index that considers the implicit paths for the specific cases.

In summary, our main contributions in this chapter are:-

- Proposed a novel problem of finding Bounded LCR Paths in labeled weighted directed graphs.
- Proposed algorithms to find the exact Bounded LCR Paths described in section 3.4 instead of finding only the existence of paths [79].
- Proposed and extended algorithms of landmark path index by including the implicit paths while indexing to find the paths described in section 3.5.
- Theoretically proved the correctness of our proposed algorithms.
- Evaluated the accuracy by computing the precision and recall for all the queries of the real and synthetic datasets in section 3.6.
- Conducted experiments and statistically evaluated the efficiency of our proposed approaches on real and synthetic benchmark datasets.
- Presented extensive survey of constrained reachability techniques, which is given in Table 3.2 for future reseach directions.

3.2 Preliminaries

An edge-labeled directed graph is denoted by G *(V, E, T_l, λ)*, where V is the set of vertices, E is the set of edges, T_l is the set of edge labels, and λ is the function that assigns each edge e $\in E$, a label $\lambda(e) \in T_l$. We describe the path p from vertex '*vs*' to '*vd*' in the edge labeled directed graph G as a vertex sequence, i.e., p = (*vs*, v_{m1}, ..., v_{mi}, ..., *vd*). v_{mi} indicates the ith intermediate vertex along the path from '*vs*' to '*vd*'. We use path label *L(p)* to denote the set of all edge labels in the path p, i.e., *L(p)* = { $\lambda(e_1)$ $\cup$ $\lambda(e_2)$ $\cup$ $\cup$ $\lambda(e_n)$}.

DEFINITION 1. (**Label-Constraint Reachability**) *Given two vertices, 'vs' and 'vd' in the edge labeled directed graph G, and a label set A, where 'vs', 'vd' $\in$ V and A $\subseteq$ T_l, if there is a path p from vertex 'vs' to 'vd' whose path label L(p) is a subset of A, i.e., L(p) $\subseteq$ A, then we say 'vs' can reach 'vd' with label-constraint A. We also refer to path p as an A-path from 'vs' to 'vd'.*[45]

LCR Query: Given two vertices '*vs*' and '*vd*', and a label set A, the label-constraint reachability (LCR) query asks if there exists an A-path from '*vs*' to '*vd*'.[45]

We demonstrate the LCR query for the road network in Fig. 3.2 through the following cases:
Case 1: For instance, the LCR query (*P1, P4, "HN"*) returns true as there exists a path {P1, P5, P4} satisfying the given constraint.
Case 2: For the LCR query *(P1, P7, "NT")*, it returns false because there doesn't exist any path from *P1* to *P7* satisfying the given label-set constraint.
Case 3: There can exist a reachability query *(P3, P1, "NT")* for which there does not exist any path in the entire graph even without satisfying the given label set constraint.
Case 4: There can exist a reachability query *(P1, P7, "NT")* for which

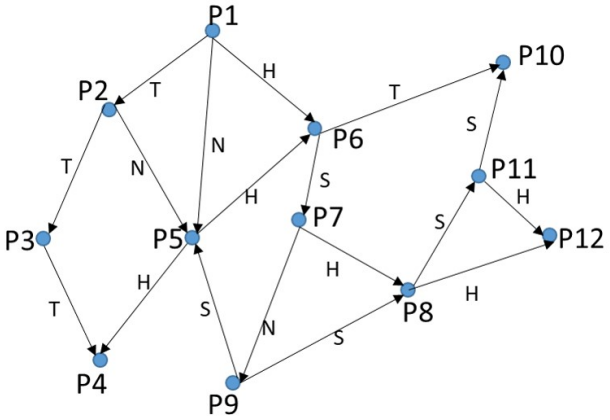


Figure (3.2) Road network with types of roads (S:Street, N:Narrow, H:Highway, T:Two-wheeler only)

there exists a path p= {*P1, P6, P7*} for which L(p) is not a subset of the given label set, i.e., L(p)="HS" $\not\subset$ "NT".

In this chapter, we handle LCR queries of type case 1, case 2 and case 3. Case 4 type of reachability query is a why-not reachability query that is beyond the scope of the thesis.

DEFINITION 2. (**Landmark Vertex**) The landmark vertices (denoted by V_L) are defined as the subset of vertices for the given graph G(V, E), i.e., $V_L \subseteq V$. These are selected based on criteria such as the top 'k' vertices of highest degree or any of the centrality measures of the graph. For instance, in Fig. 3.2, the top 4 vertices of highest total degree that can be considered as landmark vertices are {*P5, P6, P8, P1*}.

3.2.1 Problem definition

We denote an edge labeled weighted directed graph as G' (V, E, T_l, λ, w) where V represent vertex set, E denotes the edge set, T_l is the total set of different labels in G and for every edge $e \in E$, $w(e) \in \mathbb{R}^+$ and $\lambda(e) \in T_l$. The path weight or path cost is computed by adding edge weights (w(e_i))

along the path (p). Thus, the path weight for the path p denoted by C(p) is $\Sigma(w(e_i))$.

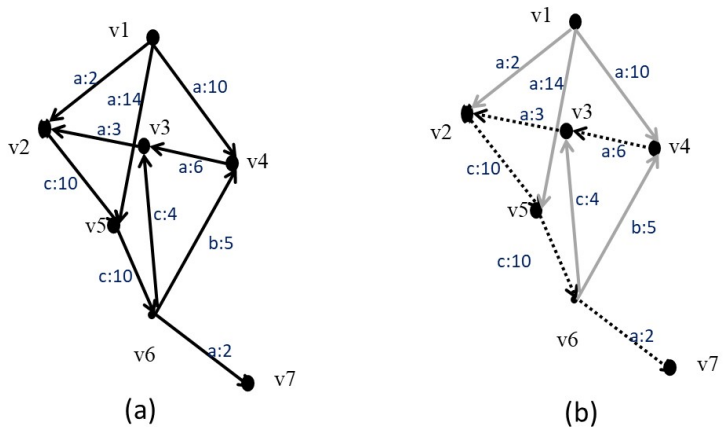


Figure (3.3) (a)Edge labeled weighted directed graph, G and (b) Resultant Bounded LCR Path for query (*v4, v7, 'ac', 50*)

Figure 3.3(a) illustrates an edge labeled weighted directed graph, G with V={ v1, v2, v3, v4, v5, v6, v7} and E={ (v1, v2), (v1, v4), (v1, v5), (v2, v5), (v3, v2), (v4, v3), (v5, v6), (v6, v3), (v6, v4), (v6, v7) }, the set of edge labels T_l={'a', 'b', 'c'}, for instance, λ (v1, v2)='a' and w(v1, v2)=2. We formulated a new problem by extending the LCR definition of Ruoming Jin et al. [45] as follows:

DEFINITION 3. (**Bounded Label Constrained Reachable Paths**)

Given two vertices *'vs'* and *'vd'*, the label set $A \subseteq T_l$ and bound for the path-weight $\delta \in \mathbb{R}^+$ in an edge-labeled weighted directed graph G', if there is an A-path lp_i from *'vs'* to *'vd'* such that the path weight C(lp_i) $\leq \delta$, then we say *'vs'* can reach *'vd'* with label-constraint A and the path weight bound δ. In other words, it can also be referred as follows: Given two vertices *'vs'* and *'vd'*, a label set A and bound δ, the bounded label constrained reachable paths are the A-paths lp_i, between *'vs'* and *'vd'* that satisfy the bounded path weight constraint C(lp_i) $\leq \delta$.

In the thesis, we referred and termed the **B**ounded **L**abel **C**onstrained **R**eachable **P**aths as BLCRP.

For example, let us consider the edge labeled weighted directed graph of figure 3.3(a). The resultant path for the BLCRP query (*v4, v7, 'ac', 50*) is p= { v4, v3, v2, v5, v6, v7 } as shown in figure 3.3(b). The path cost C(p)= w(v4, v3)+ w(v3, v2)+ w(v2, v5)+ w(v5, v6)+ w(v6, v7)=31.

For instance, in figure 3.1, we need to find the path for BLCRP query *q(vs, vd, A, δ)* with *vs*="P1", *vd*="P4", *A*="SNT" and δ=100. The resultant bounded A-path *lp* is {P1, P2, P3, P4} with the path cost of 95.

We further extend our proposed Landmark Path index algorithm by including implict paths during indexing.

DEFINITION 4. (Implicit Paths) Implicit paths are defined as the paths in the given graph that implicitly satisfy the given reachability constraints but need not satisfy the minimality of label sets.

For instance, in Fig. 3.1, there exist two paths from *P1* to *P5*, i.e., lp_1={*P1, P2, P5*} and lp_2={ *P1, P5*}. For the BLCRP query *q(P1, P5, "NT", 45)*, the resultant bounded paths satisfying the given constraints can be both lp_1 and lp_2. The path lp_1 although violating minimality of label sets property [79], satisfies the given reachability constraints. Hence, lp_1 is an implicit path.

Table 3.1 shows the different notations used in this chapter with their description.

3.3 Related Work

In this section, we reviewed various techniques that find the reachability, constrained reachability, shortest paths and constrained paths. Also, we surveyed on finding path compression techniques and efficient ways to index paths.

Table (3.1) Description of Notations

Notation	Description
G	Given graph
V	Set of vertices
E	Set of edges
T_l	Total set of edge labels in G
λ(e)	Edge label of the edge $e \in$ E
w(e)	Edge weight of the edge $e \in$ E
L(p)	Path label of path p
C(p)	Total path weight or cost of path p
δ	Given bound on path weight
k	Number of landmark vertices
n	Total number of vertices in G
m	Total number of edges in G

Reachability techniques Graph reachability finds the existence of path between the vertices in a graph. In [87], [9], the detailed survey of reachability techniques is specified. H. Wei et al. [82] classified the reachability techniques into two categories; Label-only approaches and Label+G approaches. Label-only approaches directly find the existence of reachability between the two vertices from the constructed index. Label-only approaches include 2-hop [28], 3-hop [47], Chain-cover [27], Path-Tree Cover [48], Tree-Cover [10], and Dual Labeling [80]. Label+G approaches use index labels and BFS/DFS to find the reachability between two vertices. Tree+SSPI [25] and GRIPP [78] are Label+G approaches which have linear index size and query time in terms of the number of vertices of the graph. H. Wei et al. [82] also developed an independent permutation labeling approach and two additional labels to find the reachability between two vertices.

Different variants of reachability are formed by incorporating constraints to vertices and edges of the graph. The vertex/edge constraints can be a specific bound for each edge weight, specific vertex labels, specific edge labels, membership of specific edge labels, membership of specific vertex labels, and edge/vertex labels in a specified order. Table 3.2 describes the different constraints with their state-of-the-art reachability techniques.

Table (3.2) Survey of Constrained Reachability Techniques from [2010-2018]

S.No.	Constraint	Technique (Authors,Year)	Advantages
1	edgelabel set member	Maximal Spanning Tree based Query Processing (Jin et al., 2010 [45])	compress the transitive closure of the graph to more than two orders of magnitude
2	edgelabel set member	Bidirectional labeled BFS(Fan et al., 2011 [38])	used as base technique in JoinMatch and SplitMatch algorithms to find matching graph patterns in large graphs
3	every edge weight bound	Memory-based and Disk-based Index and query processing algorithms(Miao Qiao et al., 2013 [66])	highly scalable disk based algorithm and faster query processing
4	path distance and exact edge label match in uncertain graphs	Spanning Tree based BPFilter algorithm(Minghan Chen et al., 2014 [26])	Generate subpath graph and find approximate reachability in uncertain graphs
5	edge label set member	Augmented Transitive Closure technique and Partition approach(Zou et al., 2014 [93])	scalability is addressed using sampling based solution for good partition
6	multidi-mensional vertex and edge attributes	Hashing and graph clustering based solution(Duncan Yung et al., 2016 [88])	finds reachability with multi-dimensional attribute constraints
7	edge label set member	LandmarkIndex and Query algorithms(Valstar et al., 2017 [79])	finds reachability satisfying given label constraint with significant speedups in query processing

Ruoming Jin et al. [45] formally defined label-constraint reachability as the problem of finding if there exists path between two vertices that satisfies the

given edge-label membership constraint. Ruoming Jin et al. [45] developed spanning tree based solution to LCR problem. Fan. et al [38] developed bidirectional BFS technique for LCR queries and used constraint reachability solution as the base technique for finding matching graph patterns. Lei Zou et al. [93] constructed augmented Directed Acyclic Graph (DAG) and developed transitive closure technique and partition-based technique to solve LCR queries. Valstar et. al. [79] developed a landmark based indexing technique which can handle LCR queries efficiently for large graphs. Duncan Yung et al. [88] designed hashing based technique to handle multi-dimensional vertex and edge label constraints while finding the reachability. Miao Qiao et al. [66] formalized weight constraint reachability query in which every edge weight through the path must satisfy the given range constraint.

Path finding techniques Chris Barrett et al. [17] defined edge label and edge weight constraints in formal language and computed the constrained shortest paths through dynamic programming. Ankita et al. [57] developed constrained TreeSketch algorithm that considered edge label constraint(A) and found approximate A-paths between the given vertices. Frances et al. [24] developed landmark-based indexes to compute approximate shortest path distance. Minghan Chen et al. [26] worked on uncertain graphs with sampling techniques to find approximate shortest paths constrained through distance parameter.

In [73], [74], a B+ tree index based solution is developed to solve reachability queries for graphs. Akiba et al. [11] developed a landmark-based approach with efficient pruning to solve queries of finding the exact shortest path distance between two vertices. Delling et al. [32] designed a scalable solution based on 2-hop labels for the distance queries in large networks. Ho L-Y et al. [43] developed partition replication, workload prediction and workload balancing methods to address the data locality and workload balancing while finding vertex label constrained reachability in large attributed graphs.

From the literature review, we observe that there is a scope for finding efficient reachability techniques that have lesser index construction time, lesser index size, and faster query processing. The current state-of-the-art reachability techniques cannot be applied directly to constrained reachability queries as the attributes of vertices/edges are not stored while computing the index. Many approximate shortest path techniques are developed which do not compute exact paths. Hence, there is a need to find techniques that compute the exact reachable paths satisfying the given vertex/edge attribute constraints. Hashing is found to be an efficient data structure for secondary storage access than B+tree [39]. Duncan Yung et al. [88] used BFS with hashing techniques to compute vertex-labeled reachability on big attributed graphs.

3.4 Proposed Technique to find Bounded Label Constrained Reachable Paths

In this section, our contributions are explained. We extended and modified the landmark based indexing [79] and proposed algorithms to compute path index and to find the Bounded Label Constrained Reachable Paths. In the landmark based indexing technique [79], 'k' landmark vertices were selected based on highest total degree. Min-heap based prioirty queue with path label size as priority was used to add labelsets satisfying minimality.

During index construction, we selected 'k' vertices sorted in descending order of total degree as landmark vertices. The 'k' value is considered to be $\lceil\sqrt(n)\rceil$ which is derived from theoretical observation of upper bound on the burning number of graph [51], [79]. To find bounded paths, we extended the landmark index by including intermediate paths as well as path weight while indexing. We considered path-weight as prioirty in min-heap based priority queue. While adding paths, we incorporated Dijsktra's relaxation property when path labels are same. While processing the BLCRP query(s, t, L, δ), we find the L-paths using BFS-based query processing along with

the path index and return the L-paths whose path-weights are bounded by δ.

First, we preprocessed the graph data for faster indexing and query execution. In preprocessing, vertices are numbered from 0 to n-1. The labels are also numbered from 0 to $|T_l|$-1. The corresponding bits of numbered labels are set during path index construction. For instance, fig. 3.4 shows the preprocessed directed graph G′ of fig. 3.1. Thus, the preprocessing step saves storage space while indexing.

3.4.1 Path indexing algorithm

In the algorithm (LWPathIndex), for each landmark vertex, all reachable vertices, their labels, the path and path-weight are computed using LWPathIndexPerLM() procedure and stored in Path LandMark index(PLM). In AddPathInfo() procedure, we add paths to path index by considering minimality of labelsets [79] and cost constraints. While adding paths with same labels, we add only paths that preserve Dijkstra's relaxation property.

Table 3.3 describes how label constraints and weight constraints are considered while indexing. For any reachable vertex v from s, let (L′, cost′) be the path label and path-weight (cost) for path p′ that is already inserted and (L, cost) represent the path label and cost for path p that is to be inserted based on Table 3.3. Minimality of labelsets is preserved as well as Dijkstra's relaxation property for weights is not violated for the cases 5-8. But, for the cases 1-4, we have performed trade-off for faster indexing by adding paths preserving minimality of labelsets. For instance, let '0' be one of landmark vertices for the graph G′ in Fig. 3.4. Suppose PLM[0] for vertex '6' has L′=7 , p′='0-4-5-6' and cost′=141. If a tuple('6', L, p, cost) with L=5, p='0-5-6' and cost=106 is encountered, it is inserted into PLM[0] and the record ('6', L′, p′, cost′) is deleted. We add only those simple paths to the path index whose path length$\leq \lceil$diameter/2$\rceil$ for faster indexing.

Algorithm 1: LWPathIndex

```
PLM[v_i]←LWPathIndexPerLM (v_i)// i ∈ 1:k
PNL[v_j]←LWPathIndexPerNM (v_j)// j ∈ remaining (n-k) vertices
procedure LWPathIndexPerLM (v) // Let q be priority queue
while q is not empty do
    Dequeue u and add its path information to PLM[s] through
    AddPathInfo ()
    Add u to transL[s][S] if same labeled data exists else add(u, L) to
    transL[s]
    if u is indexed then
        ExpandOut (); continue
    for w ∈ adj(u) do
        if PathLength(s,w)≤ ⌈diameter(g)/2⌉ then
            Enqueue w
procedure LWPathIndexPerNM (v, b)
while q is not empty do
    Add u and its path information to PNL[s] through AddPathInfo ()
    if u is indexed then
        ExpandOutNM (s, u, L, iv, cost) upto b landmark vertices
    for w ∈ adj(u) do
        Enqueue w
procedure AddPathInfo (s, v, L, intv, cost)
if (v, L′, intv′, cost′) ∈ PInd[s] and L′ ⊆ L then
    return false
Delete any (v, L′, intv′, cost′ ) with L⊂ L′ or (L′ = L and cost′ > cost)
from PInd[s].
Add (v, L, intv, cost) to PInd[s].// PInd=PLM for LM index, else
    PInd=PNL
return true
```

An additional index *transL* is created for the landmark vertices for which we either generate a new entry (L, v) or append v to the previous entry in $transL[l_i][L]$ used for efficient pruning in query processing. For each non-landmark vertex, LWPathIndexPerNM() computes Path Non-Landmark index (PNL). ExpandOut() and ExpandOutNM() methods propogate the reachability information of indexed vertices that lead to faster index construction for PLM and PNL respectively.

Table (3.3) Cases of label constraints and cost constraints while indexing

Case No.	Label set condition	Cost condition	(L′, cost′) removed?	(L, cost) added?	Dijkstra's property preserved?	Minimality preserved?
1	$L' \subset L$	cost<cost′	No	No	No	Yes
2	$L \subset L'$	cost>cost′	Yes	Yes	No	Yes
3	$L \not\subset L'$ & $L' \not\subset L$	cost<cost′	No	Yes	No	Yes
4	$L \not\subset L'$ & $L' \not\subset L$	cost>cost′	No	Yes	No	Yes
5	$L \subset L'$	cost≤cost′	Yes	Yes	Yes	Yes
6	$L = L'$	cost≤cost′	Yes	Yes	Yes	Yes
7	$L = L'$	cost>cost′	No	No	Yes	Yes
8	$L' \subset L$	cost≥cost′	No	No	Yes	Yes

3.4.2 Query processing algorithm

We modified and extended the BFS-based query processing approach [79] by accessing the path index and returning the label constrained paths whose path-weight is within given maximum bound. The query processing of BLCRP queries is evaluated based on QBPath Algorithm. If s is landmark vertex, then QPathLM() procedure is invoked that checks if there exists (l, p, $cost$) in PLM[s] for the target vertex t where l is the path label for p and $cost$ is path-weight with $l \subseteq L$ and $cost \leq maxcost$, then p is returned. If s is non-landmark vertex, the vertices are either checked in PNL or traversed through breadth-first search, till t is reached. The vertices from s along the path that cannot reach t are marked as visited using QCheckMark(). For example, let the BLCRP query be (0, 6, 5, 150) for the graph of figure 3.4. Since, '0' is a landmark vertex, the constraints are directly checked in PLM index and the resultant retrieved path is {0, 5, 6}. Thus, we find the multiple exact paths by using proposed path indexing and query processing algorithms (referred as LM2 in section 3.6) by satisfying the given constraints of the query.

Algorithm 2: QBPath

```
Input  : s, t, L, maxcost
Output: Bounded Paths p[i] ∈ p
if s∈ V_L then
    QPathLM (s, t, L, maxcost)
// PLM index is invoked in QPathLM()
for (v, L', intv', cost') ∈ PNL[s] do
    if ( L' ⊆ L and QCheckMark (v, t, L, marked, maxcost)=true) then
        Add path s∼v∼t to p
    // v∼t derived from QPathLM()in QCheckMark()
while q is not empty do
    if v=t then
        Add path s∼t to p; break
    if v ∈ V_L and QCheckMark (v, t, L, marked, maxcost)=true then
        Add path s∼v∼t to p
    for w ∈ adj(v) do
        if (marked(w)=false and λ(v, w) ⊆ L) then
            Insert w into q
if (p is not empty and pcost(p[i])≤maxcost, p[i] ∈ p) then
    return p
```

3.5 Extended Proposed Technique by including Implicit Paths

The extended works include incorporating implicit paths. By including the implicit paths in the index, we achieve higher recall than LM2. These extensions along with implicit paths, i.e., LWPathIndexImplicit and BImplPath algorithms are referred as LandMark path extensions (LM3) in section 3.6.

3.5.1 Path indexing algorithm by including implicit paths

For each landmark vertex (v), we construct a reachable Implicit Path Landmark index (IPL) that includes reachable vertex (from v), its label,

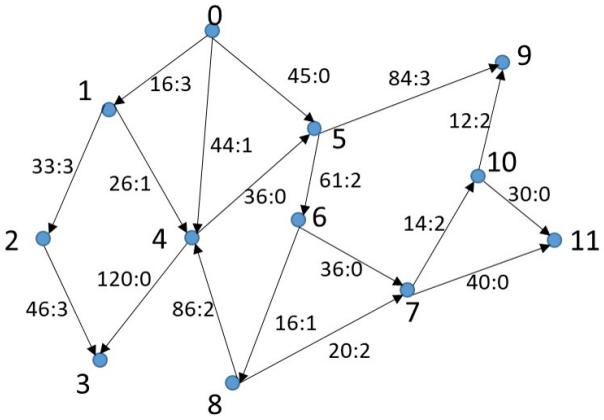


Figure (3.4) Pre-processed directed graph G′ of the road network G1

path and weight. While inserting the same vertex with different labels or weights, we consider the minimality of label sets [79] and Dijkstra's relaxation property based on Table 3.6.

Assumption We assume only simple paths, whose path length is less than or equal to half the diameter of the graph exist between the source landmark vertex and most of its reachable vertices. Therefore, faster indexing is achieved.

Table 3.4 shows the path landmark index for vertices 0, 4 and 7 of pre-processed directed graph G′ of Fig. 3.4. For instance, the tuple <4, 10, 1, 42> in IPL[0] of Table 3.4 indicates the vertex 0 can reach vertex 4 via vertex 1 with bit processed label constraint 10, and total path weight of path {0, 1, 4} is 42. For non-landmark vertices, upto 'b' reachable landmark vertices are indexed in Path Non-Landmark (PNL). Table 3.5 shows the PNL index for non-landmark vertices 5, 6 and 8 of Fig. 3.4.

Table 3.6 shows the different possible cases during path indexing for which minimality and Dijkstra's relaxation properties are handled. We denote the path label to be indexed by L and path label already indexed by L'. We denote v as the reachable vertex and iv and iv' denote the

Algorithm 3: LWPathIndexImplicit

Input : Edge-Labeled Weighted Directed Graph, G
Output: transL, IPL, PNL
procedure `ImplPathperLV` (v) `// Enqueue (v,,,0)in priority queue q`
while *q is not empty* **do**
  Add (u, L, iv, wt) IPL[s] through `AddIPath` ()
  `// L is the path label from s to u`
  Append u to transL[s][S] if L=S, else insert (u, L) to transL[s]
  if *u is indexed* **then**
    `ImplExpand` (); **continue**
  for *(w, L′, wt′) ∈ outneighbors(u)* **do**
    iv←join(iv,w); L″ ← L∪ L′
    wt″ ← wt+wt′
    if *PathLength of s∼w is less than half of graph diameter* **then**
      Add (w, L″, iv, wt″) to *q*

procedure `AddIPath` (s, v, L, intv, C)
if *(v, L′, intv′, C′) ∈ PInd[s] and L′ ⊆ L* **then**
  return false
if *(L⊂ L′ or (L′ = L and C′ > C))* **then**
  Delete entry (v, L′, intv′, C′) from PInd[s]
. **if** *(L⊂ L′ and C′ > C)* **then**
  Delete (v, L′, intv′, C′) from PInd[s].
IE← (v, L, intv, C)
Insert IE into PInd[s].`// PInd=IPL for LM index, else PInd=PNL`
return true
procedure `ImplPathIndNV` (v, b)
while *q is not empty* **do**
  Dequeue (u, L, iv, wt)
  if *(isLandmark(u)=**true**)* **then**
    Add (u, L, iv, wt) to PNL[s] through `AddIPath` ()
  if *((u is indexed)& b>0)* **then**
    `ImplExpandNM` (s, u, L, iv, C);b=b-1;
  for *(w, L′, wt′) ∈ outneighbors(u)* **do**
    iv←join(iv,w); L″ ← L∪ L′
    wt″ ← wt+wt′; Enqueue (w, L″, iv, wt″)

procedure `ImplExpand` (s, u, PL, iv, C)
for *every (v, PL′, intv′, C′) in IPL[u]* **do**
  PL″=PL ∪ PL′;intv″=join(iv, intv′);C″=C+C′; Add (v, PL″, intv″, C″) through `AddIPath` ()
procedure `ImplExpandNM` (s, u, PL, iv, C)
`// PInd=IPL if isLandmark(u)=true, else PInd=PNL`
for *every (v, PL′, intv′, C′) in PInd[u]* **do**
  if *(v is indexed & isLandmark(v)=true)* **then**
    PL″ ← PL ∪ PL′;intv″ ← join(iv,intv′);C″ ← C+C′; Add (v, PL″, intv″, C″) through `AddIPath` ()

Table (3.4) Path Landmark index of landmark vertices for Fig. 3.4

Index	Tuples
IPL[0]	<1, 8, -, 16>, <4, 10, 1, 42>, <4, 2, -, 44>, <5, 1, -, 45>, <2, 8, -, 49>, <3, 8, 1-2, 95>, <6, 5, 5, 106>, <8, 7, 5-6, 122>, <7, 5, 5-6, 142>, <3, 3, 4, 164>
IPL[4]	<5, 1, -, 36>,<6, 5, 5, 97>, <8, 7, 5-6, 113>, <3, 1, -, 120>, <7, 5, 5-6, 133>
IPL[7]	<10, 4, -, 14>, <9, 4, 10, 26>, <11, 1, -, 40>

Table (3.5) Path Non-Landmark index for Fig. 3.4

Index	Tuples
PNL[5]	<7, 5, 6, 97>, <4, 6, 6-8, 163>
PNL[6]	<7, 1, -, 36>, <4, 6, 8, 102>
PNL[8]	<4, 4, -, 86>, <7, 4, -, 20>
...	...

intermediate paths of the new tuple (to be indexed) and indexed tuple respectively. We denote the path weight (or cost) to be indexed by C and indexed path weight by C'.

Table (3.6) Label and cost constraints while indexing with implicit paths

Case No.	Label set condition	Cost condition	(L′, C′) removed?	(L, C) added?	Dijkstra's property preserved?	Minimality preserved?
1	$L' \subset L$	$C<C'$	No	Yes	No	No
2	$L \subset L'$	$C>C'$	No	Yes	No	No
3	$L \not\subset L'$ & $L' \not\subset L$	$C<C'$	No	Yes	No	Yes
4	$L \not\subset L'$ & $L' \not\subset L$	$C>C'$	No	Yes	No	Yes
5	$L \subset L'$	$C \leq C'$	Yes	Yes	Yes	Yes
6	$L = L'$	$C<C'$	Yes	Yes	Yes	Yes
7	$L = L'$	$C \geq C'$	No	No	Yes	Yes
8	$L' \subset L$	$C \geq C'$	No	No	Yes	Yes

For the cases 1 and 2 of Table 3.6, both minimality of label sets and Dijkstra's relaxation property are not preserved; yet the resultant paths may implicitly satisfy the given label constraints and the bounded path constraint. Hence, both the existing tuple $<v,\ L',\ iv',\ C'>$ and new tuple $<v,\ L,\ iv,\ C>$ are retained. For instance, in Table 3.4, for IPL[0], both tuples < 4, 10, 1, 42> and <4, 2, -, 44> are retained although minimality of label sets and Dijkstra's relaxation property constraints are violated. Thus, implicit paths are included in the constructed index through case 1 and case 2 of Table 3.6.

When $L \neq L'$, irrespective of $C \leq C'$ or $C \geq C'$, the tuple $<v,\ L,\ iv,\ C>$ can be added to path index for cases 3 and 4 of Table 3.6. For case 5 and case 6, $L \subseteq L'$ and $C < C'$, both minimality of label sets and Dijkstra's relaxation property constraints are satisfied. Hence, the existing index tuple $<v,\ L',\ iv',\ C'>$ is removed and the new tuple $<v,\ L,\ iv,\ C>$ is added. For case 7 and case 8, $L' \subseteq L$ and $C > C'$, both minimality of label sets and Dijkstra's relaxation property constraints are satisfied. Hence, the existing index tuple $<v,\ L',\ iv',\ C'>$ is retained and the new tuple $<v,\ L,\ iv,\ C>$ is not added.

The proposed Algorithm 3 describes landmark path index construction by including implicit paths for the edge-labeled weighted directed graph. Landmark vertices ($V_L \subseteq V$) are obtained by sorting vertices in decreasing order of their total degree and selecting the first 'k' vertices from the sorted vertices. Algorithm 3 first generates Implicit Path Landmark index (IPL) for landmark vertices by invoking call to ImplPathperLV() for each landmark vertex. Next, ImplPathIndNV() procedure is invoked that generates Path Non-Landmark index (PNL).

For each landmark vertex v_s, its descendent vertices are traversed and pushed into a priority queue q. In priority queue, vertices are prioritized

based on increasing path length from the vertex v_s. Each descendant vertex (v), its label (L), intermediate path (iv) and total path weight (wt) are stored as tuple $<v,\ L,\ iv,\ wt>$ in $IPL[v_s]$ through AddIPath() in the algorithm. *IPL* is the path landmark index which stores the tuples in a list for each landmark vertex. AddIPath() adds the tuples by considering minimality of labelsets and Dijkstra's relaxation property constraints as shown in Table 3.6.

If v_s encounters an already indexed descendant vertex v', all the tuples of v' are joined to v_s using ImplExpand(). The reachable paths with intermediate path length less than or equal to half of the diameter of the graph are enqueued. For instance, in Table 3.4, for vertex 0 of graph G′ (Fig. 3.4), IPL[0] stores the reachable vertices with their path information.

ImplPathIndNV() stores upto 'b' reachable landmark vertices for each non-landmark vertex in Algorithm 3. In ImplPathIndNV(), for each non-landmark vertex v_n, its descendent vertices are pushed into the priority queue. If dequeued vertex v' is a landmark vertex, the vertex (v'), its label (L), intermediate path (iv) and total pathweight (wt) are stored as tuple $<v',\ L,\ iv,\ wt>$ in PNL[v_n] by invoking AddIPath(). If v' is already indexed, its tuples are joined and stored in PNL[v_n] using ImplExpandNM(). In ImplExpandNM(), if vertex passed is non-landmark vertex, PNL[v'] is used, otherwise IPL[v'] is used to join the tuples. Thus, upto 'b' reachable landmark vertices are stored in PNL for each non-landmark vertex. For instance, for non-landmark vertex 8 of graph G′, PNL[8] in Table 3.5 shows its reachable landmark vertices 4 and 7 and their path information.

3.5.2 Query processing algorithm by including implicit paths

During query processing, the first step is to check if the source vertex is a landmark vertex or not. The proposed Algorithm 4(BImplPath) describes

the query processing of BLCRP query *q(s, t, L, maxcost)* through implicit path indexing. If 's' is a landmark vertex, QPathLM() is invoked. QPathLM() uses IPL[s] to find the target vertex. If the target vertex is reached and the given constraints are satisfied, then the corresponding paths are added to *p*. For instance, consider BLCRP query q(0, 7, 5, 180) of Fig. 3.4, the resultant path {0, 5, 6, 7} is returned using IPL[0] through QPathLM() of Algorithm 4.

If source vertex is non-landmark vertex and target vertex is landmark vertex, the target vertex can be reached through PNL. If target vertex is non-landmark vertex, source vertex may reach the target vertex through an intermediate landmark vertex by invoking QMark(). For instance, BLCRP query *q(8, 10, 4, 60)* of Fig. 3.4 returns the resultant path {8, 7, 10} through PNL[8] and IPL[7]. The descendant vertices are traversed till target vertex is reached satisfying the label constraints. If an intermediate landmark vertex is reached, its reachable vertices are checked for the target vertex using QMark().

3.5.3 Correctness proof

Lemma 1: Let G(V, E, L, w) be a graph and $k \in \mathbb{N}$. *Let* $\{IPL[v_i]\}_{i\in[k]}$ *be index constructed by LWPathIndexImplicit(G) where* v_1, v_2, v_3,..v_k *are landmark vertices. Then, for every* $i \in [1, k]$, $IPL[v_i]$ *is sound but not complete.*

Proof: The paths indexed in IPL[v_i] are valid from v_i to any reachable vertex v as we traverse through the descendant vertices of v_i. IPL[v_i] is not complete as we consider indexing only those paths whose intermediate path length is less than or equal to half of the diameter of the graph. We include the labels by union of labels along the path. We also include the intermediate vertices and path weight for each record in the path index. Thus, all possible reachable vertices are traversed and indexed with their path information. Hence, we can say that IPL[v_i] is sound.

Algorithm 4: BImplPath

Input : s, t, L, maxcost
Output: Bounded Paths p[i], i∈[1, maxp] where maxp is maximum number of paths allowed

if $s \in V_L$ **then**
  p[i]=`QPathLM` (s, t, L, maxcost)
  return p

for *(v, PL′, intv′, C′) ∈ PNL[s]* **do**
  if *(PL′ ⊆ L* ***and*** *(v= t))*`// when target vertex is a landmark vertex` **then**
    if *C(s∼v)≤maxcost* **then**
      Insert s∼*v* to p;**break**
  if *(PL′ ⊆ L* ***and*** `QMark` *(v, t, L, marked, maxcost)=**true**)* **then**
    Insert s∼*v*∼p_i into p `//` p_i `is path from v to t through QMark()`

Enqueue s
while *q is not empty* **do**
  Dequeue vertex v
  if *(v=t)* **then**
    p′ ← s∼t
    if *(pcost(p′) ≤ maxcost)* **then**
      Add path p′ to p;**break**
  if *(v ∈ V_L* ***and*** `QMark` *(v, t, L, marked, maxcost)=**true**)* **then**
    p′ ← s∼v∼p_i
    `//` p_i `is intermediate path from QMark()`
    if *(pcost(p′) ≤ maxcost)* **then**
      Add path p′ to p
  for *(w, PL′) ∈ outneighbors(v)* **do**
    if *(marked(w)=**false*** ***and*** *PL′ ⊆ L)* **then**
      Enqueue w
if *(p is not empty & pcost(p[i])≤maxcost, p[i] ∈ p)* **then**
  return p
procedure `QPathLM` (s, t, PL, bound)
for *((u, PL′, iv′, wt′) in IPL[s])* **do**
  if *(u=t) & (PL′ ⊆ PL)* **then**
    p[i] =s ∼iv′ ∼ t
    if *(wt′ ≤ bound)* **then**
      Add p[i] to p
procedure `QMark` (s, t, PL, marked, bound)
if *(*`QPathLM` *(s, t, PL, bound)=**true**)* **then**
  return true
for *(S′, PL′) ∈ transL[s]* **do**
  if *(PL′ ⊂ PL)* **then**
    marked= S′ ∪ marked
return false

Lemma 2: Let G(V, E, L, w) be a graph and $k \in \mathbb{N}$. Let $\{PNL[v_i]\}_{i\in[k+1,n]}$ be index constructed by LWPathIndexImplicit(G) where v_{k+1}, v_{k+2}, ..v_n are non-landmark vertices. Then, for every i ∈[k+1, n], PNL[v_i] is sound.
Proof: For each non-landmark vertex v_i, its descendant vertices are traversed by invoking ImplicitPathIndNV(). Only landmark vertices are added to PNL[v_i]. It is not complete because upto 'b' landmark reachable vertices are only indexed. Thus, every reachable landmark vertex and its path information is included in PNL[v_i] upto 'b' records. Hence, PNL[v_i] is sound.

Theorem 3.1. *Bounded path for LCR query, q=(s, t, L, δ) is a true query if BImplPath(s, t, L, δ) using the path index from LWPathIndexImplicit algorithm returns atleast one bounded path.*

Proof: We can prove the above theorem in the following cases:
Case 1: If s is a landmark vertex, BImplPath(s, t, L, δ) invokes call to QPathLM() which finds the target vertex information from IPL[s] in the algorithm and returns the resultant path if it is within the bound(δ).
Case 2: If s is not landmark vertex, its reachable landmark vertices are traversed through PNL[s]. From each of the landmark vertex, v_l, reachable vertices are checked for target vertex through IPL[v_l] in QMark() procedure. Thus, the query returns the resultant bounded path when target vertex is reached.
Case 3: If s is not landmark vertex, and the target vertex t is not reached through PNL[s], then breadth first traversal of s is carried out till landmark vertex or target vertex is reached. If an intermediate landmark vertex (v_l) is reached, the existence of target vertex is checked in IPL[v_l]. If target vertex is reached, the query returns the resultant bounded path.

If no paths are found from all the above cases, then BImplPath() for the query q returns zero paths.

3.5.4 Time complexity

To compute time complexity of LWPathIndexImplicit algorithm, it involves computing time complexity for the following significant steps:

- Algorithm 3 invokes call to ImplPathperLV() k times. It takes O(kx) time where x is time complexity of ImplPathperLV().
- It takes O((n-k)y) time where y is the time complexity of ImplPathIndNV().
- It takes atmost $2^{|L|}$ labels for each landmark vertex in the worst case where |L| denotes the total number of labels in T_l. Each push into priority queue requires O(logn) time. Thus, for all landmark vertices, the time complexity is O(x)=O(n*(logn+$2^{|L|}$)).
- It requires atmost $2^{|L|}$ time to store all possible labels in transL.
- ImplicitPathIndNV() method has O((n*(logn+b)+m)$2^{|L|}$) time complexity because upto 'b' landmark vertices are considered.

Thus, the total time complexity is O(k((n(logn+$2^{|L|}$)+m)*$2^{|L|}$))+O((n-k)(n(logn+b)+m)*$2^{|L|}$). The query processing time complexity is computed through following steps:

- QPathLM() of BImplPath algorithm requires O($2^{|L|}$+logn) as finding any specific w in IPL needs O(logn) time.
- QMark() requires worst case running time of QPathLM() and setting n bits as marked for atmost $2^{|L|}$ labels. Thus, the worst case running time of QMark() is O(n+$2^{|L|}$).
- The remaining graph exploration part of Algorithm 4 takes atmost O(n+m) time.

Thus, the total query processing requires O(m+k($2^{|L|}$+n)) time.

3.6 Experimental Evaluation

We conducted experiments on real and synthetic data sets shown in Table 3.7. The following subsection describes the datasets. While constructing landmark path index, we considered k=$\lceil\sqrt{n}\rceil$ and b=20 based on the parameter values set in [79] for the proposed approaches. Fig. 3.5 shows

the resultant index path construction size and Fig. 3.6 shows the index construction time for the datasets described in Table 3.7 respectively. We observe that our proposed approach (LM3) has a faster index construction time and lesser index size than LM2 [20], which is because we do not consider the minimality of label sets for all the cases.

Table (3.7) Dataset repository

S.No.	Dataset	n	m	$\|T_L\|$	real/synthetic
1	Robots[1]	1724	3596	4	real
2	Erdos-Renyi Graph[53]	987	2000	8	synthetic
3	Preferential-Attachment graph[53]	1000	1997	8	synthetic

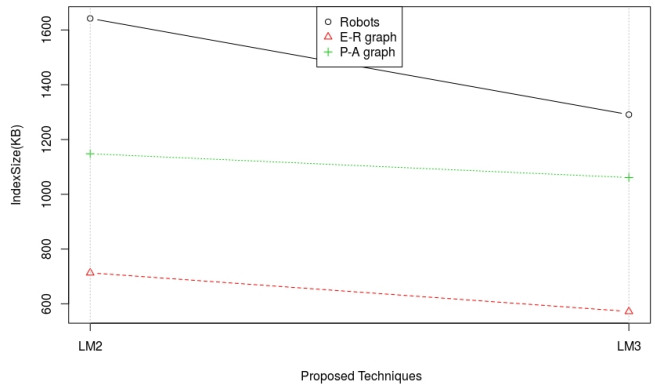


Figure (3.5) Landmark path index construction size for the proposed LM2 and LM3 techniques

3.6.1 Dataset description

We describe the significance and generation of the real and synthetic datasets of Table 3.7. The edge weights are assigned values from {10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120} randomly to all datasets described in Table 3.7.

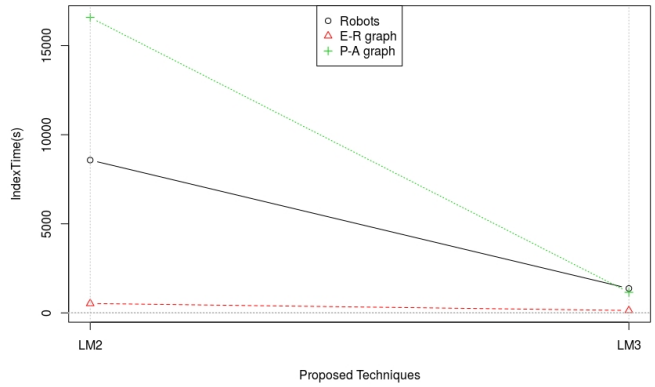


Figure (3.6) Landmark path index construction time for the proposed LM2 and LM3 techniques

3.6.1.1 Erdos-Renyi Graph

Erdos-Renyi (E-R) graphs are the basic random graphs [13] with near uniform degree distribution. E-R graph is generated using SNAP [53] with possible edges for each vertex set to 2 and n=1000. We assign 8 labels are randomly to the edges.

3.6.1.2 Preferential-Attachment Graph

Preferential-attachment (P-A) graphs are synthetic graphs with the property that vertices with higher degree are more likely to have edges to be added in the future. These graphs have skewed degree distribution. P-A Graphs are scale-free networks [13] and hence, mimic the real world networks. P-A graph is generated using SNAP [53] with n=1000 and m=2000. The 8 labels are randomly distributed to the edges.

3.6.1.3 Robots

Robots is a real trust network [1] with 4 edge labels; M-master, A-apprentice, Journeyer-J and Observer-O. The edge labels denote the level of trust interaction between the users.

3.6.2 Query generation and evaluation

We generated 100 true queries and 100 false queries based on BFS query generation [79] with random path weights and tested with different labels for different datasets. For synthetic datasets, we generated queries with labels, |L|=4, 6 out of the total eight labels, and for Robots dataset, we generated queries with |L|=2, 3 out of the 4 labels. During query generation, we randomly generated the bound for all queries with their values in range [10*diameter, 120*diameter]. We have considered this range for bound, as paths, in general, have path length near to the diameter of the graph.

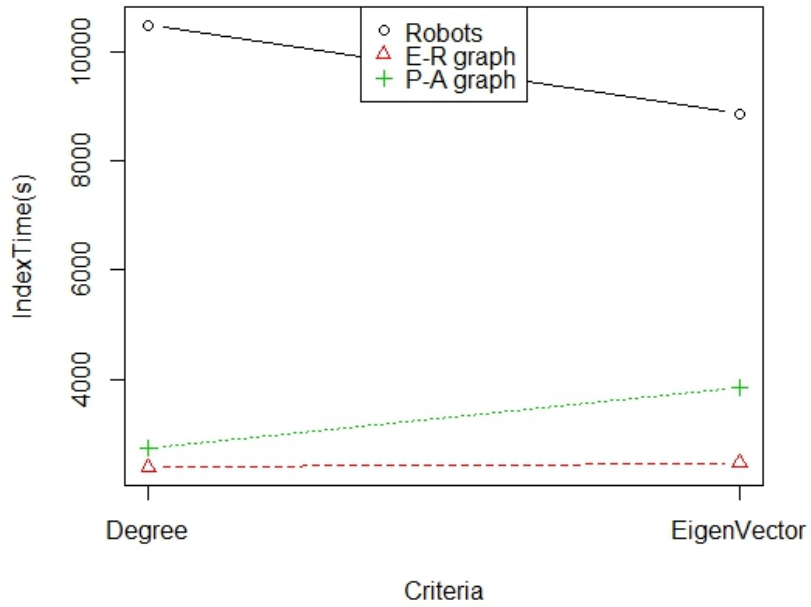


Figure (3.7) Landmark path index construction time for the proposed LM2 technique with degree vs eigenvector centrality

Figure 3.7 shows the index construction time and figure 3.8 shows the index construction size of the proposed LM2 technique using degree vs eigenvector centrality for selection of landmark vertices. Figure 3.9 shows

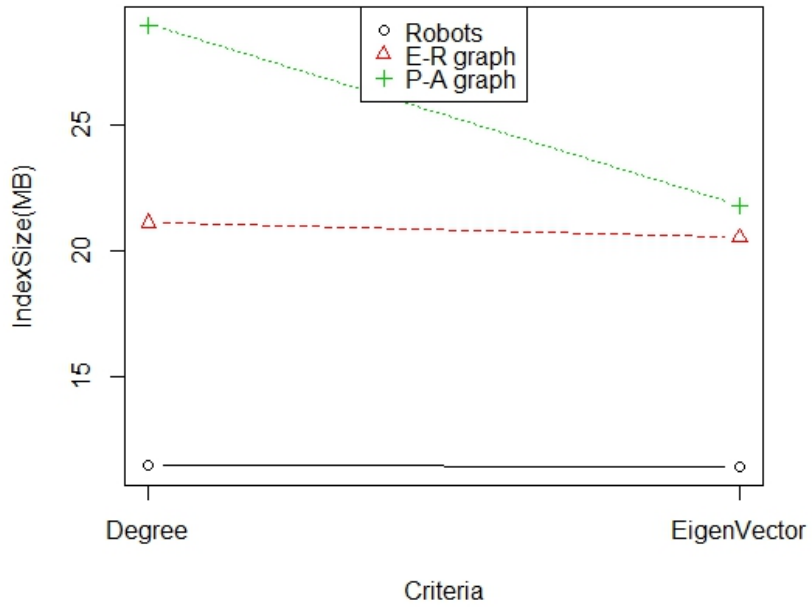


Figure (3.8) Landmark path index construction size for the proposed LM2 technique with degree vs eigenvector centrality

the index construction time and figure 3.10 shows the index construction size of the proposed LM3 technique using degree vs eigenvector centrality for selection of landmark vertices. From these, we observe that using eigen vector centrality measure, the index was constructed relatively faster and occupied lesser space.

Table 3.8 shows the average query execution time and false negative ratio of BLCRP queries for the proposed technique using degree and eigen vector centrality. From this table, we observe that by using degree, the average query execution time is relatively lesser for almost all the datasets. Hence, it shows that using degree for landmark selection is better than using eigen vector centrality, although index construction size and time may be lesser. The false negative ratios are due to the inclusion of the condition path length$\leq$ (.5*diameter) of the graph in the proposed landmark path index.

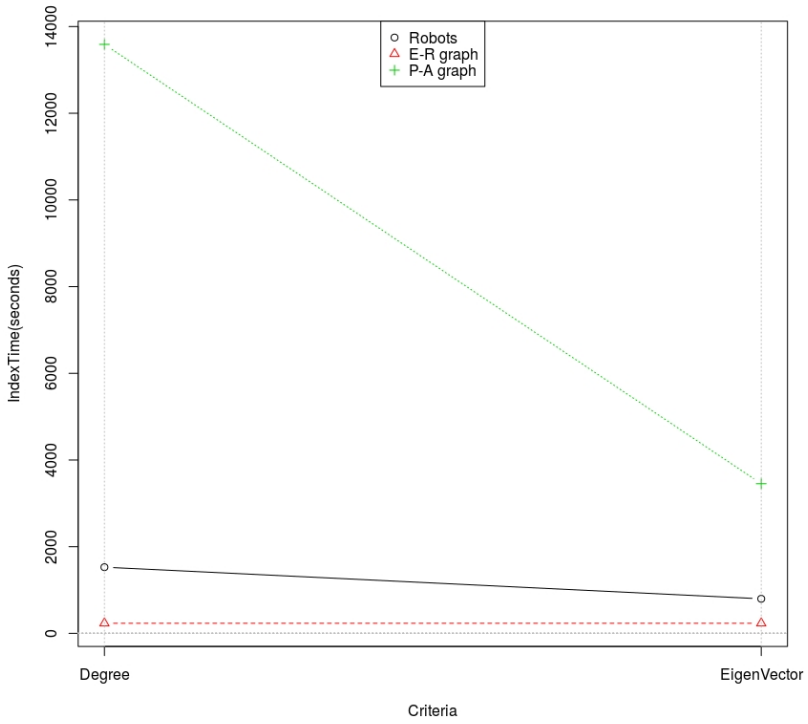


Figure (3.9) Landmark path index construction time for the proposed LM3 technique with degree vs eigenvector centrality

It is found to be considerably small.

We computed accuracy measures of Precision and Recall for the proposed landmark based path indexing algorithms. We computed precision based on equation 3.1 by checking if the resultant retrieved paths are present in the graph and if they satisfy the given label constraints and bounded weights.

$$Precision = \frac{Numberofretrievedpaths}{Numberofretrievedandrelevantpaths} \tag{3.1}$$

Recall is computed based on equation 3.2 considering minimality of label sets and Dijkstra's relaxation property based on Table 3.6 for the LM3 approach to find whether all possible relevant paths are retrieved from the

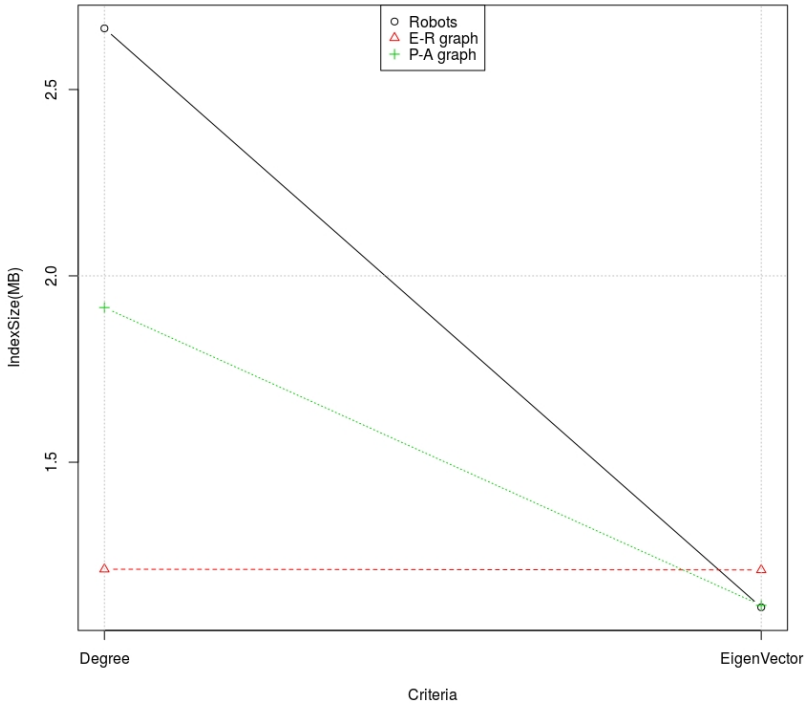


Figure (3.10) Landmark path index construction size for the proposed LM3 technique with degree vs eigenvector centrality

given graph. The total number of possible paths is computed based on DFS satisfying the given label constraints and bound constraints.

$$Recall = \frac{Numberofretrievedpaths}{TotalNumberofrelevantpaths} \tag{3.2}$$

The precision percentage is computed on the real and synthetic datasets for proposed Landmark based path indexing technique (LM3) and found to be 100% for all the datasets. The recall percentage is computed for verification and is found to be between 16% and 100% for Landmark based path indexing technique tested on real and synthetic benchmark datasets for the true queries. Table 3.9 shows the minimum, maximum, and average recall computed for 100 true queries on different datasets which reveals that landmark based path indexing computes bounded label constrained

reachable paths efficiently.

Table (3.8) Average query execution time and the false negative ratio(τ) of true queries(tq) and average query execution time of false queries(fq) in milli seconds using degree(D) and eigen vector centrality(EV) as criteria with the number of labels, nl for LM3

Dataset	nl	tq(D)	tq(EV)	τ (D)	τ (EV)	fq(D)	fq(EV)
Robots	2	.0433	.0452	.072	.072	.0113	.0116
	3	.038	.0344	.063	.063	.0118	.0124
E-R graph	4	.0008	.0985	.0858	.0858	.0135	.0135
	6	.0379	.0442	.205	.06	.06	.0168
P-A graph	4	.0967	.0837	.08	.08	.0252	.0185
	6	.0755	.0942	.078	.078	.0165	.0168

We evaluate the efficiency of our proposed approach (LM3) by statistical analysis of recall. Paired t-test is used to find the statistical significance of LM3 over LM2. The sample data values need to be in the normal distribution to use the paired t-test. Also, the sample size must be at least 30 for non-normal data [71]. We have tested recall values for 100 random true queries with |L|=4 for synthetic datasets and |L|=2 for real dataset to find the statistical significance of LM3 compared to LM2.

The following are the parameters set and formulas used to compute paired t-test [71] :
The null hypothesis is set to difference of recall values to 0.
α=0.05.
Number of samples, n=100
The t-statistic is computed using t=$\bar{x}/SD/\sqrt{n}$ where $\bar{x}$ is the average and SD is the standard deviation of differences(x) respectively.
The probabilistic value p is calculated using statistical function tdist() as p=tdist(t,df,1) where df is the degrees of freedom (df=n-1).

Table (3.9) Recall analysis of the proposed approach (LM3)

S.No.	Dataset	$\|L\|$	*Min. recall*	*Max. recall*	*Average Recall*
1	Erdos-Renyi Graph	4	0.3	1	0.8097
2	Preferential -Attachment Graph	4	0.1666	1	0.5673
3	Robots	2	0.1666	1	0.59385

Table (3.10) Statistical analysis of recall between LM3 and LM2

S.No.	Dataset	$\|L\|$	*t-statistic*	*p value*
1	Erdos-Renyi Graph	4	4.328	0.00001
2	Preferential -Attachment Graph	4	7.228	0.000005
3	Robots	2	3.155	0.001076

Table 3.10 shows that our proposed approach is statistically significant than previous approach with respect to recall for real and synthetic datasets. Between two groups, if p≤0.05, they are statistically significant. If p≤0.01, then they are statistically highly significant. If p≤0.001, then they are statistically extremely significant. From Table 3.10, we observe that LM3 is statistically significant than LM2.

3.7 Conclusions

We proposed an efficient solution to the problem of finding BLCRP by extending the landmark based indexing and query processing. We find the reachable paths that satisfy given label constraints and bounded by weight. Landmark vertices are selected by choosing top 'k' vertices based on the descending order of the total degree of vertices. For each landmark vertex, all reachable paths and their path information are stored in the path landmark index. We also addressed the specific cases by including implicit paths. While computing reachable paths, the implicit paths that may satisfy the edge label constraints but do not consider the minimality of label sets and Dijkstra's relaxation properties are also included. The bounded path based LCR query is processed by using IPL and PNL indices to get the resultant paths. We evaluated accuracy of our proposed techniques by

using precision and recall measures on the resultant paths. We observed that our approach is statistically significant on synthetic and real datasets.

Chapter 4

Multidimensional Constraint Reachable Paths for Attributed Graphs

In big data era, a graph can be rendered as a significant modeling tool to represent complex relationships between the objects. In real time, these objects or vertices of the graph have multiple attributes and different relationships between them. The problem of multidimensional constraint reachable paths is to find the path between the given vertices that match the user specified multidimensional vertex and edge constraints. An important challenge is to store the graph topology and attribute information while constructing reachability index. We proposed hashing based heuristic search technique to solve the multidimensional constraint reachability queries. In this chapter, we experimented our proposed techniques and evaluated their performance on real and synthetic datasets. This chapter publication is communicated and listed below. Section 4.1 introduces the multidimensional constraint reachability queries and summarizes main findings and contributions of this chapter. Section 4.2 describes the terminologies and problem statement. In section 4.3, we describe the literature on attributed graphs, the

Bhargavi B., and K. Swarupa Rani, and Arunjyothi Neog, "Finding Multidimensional Constraint Reachable Paths for Attributed Graphs", *Communicated to* Applied Intelligence, Springer, 2020.

constrained reachability techniques and attributed graph clustering techniques. Section 4.4 and section 4.5 describe our proposed approaches with illustrations. Section 4.6 describes the experiments and evaluation of our proposed techniques.

4.1 Introduction

Graph mining is the process of extracting useful knowledge from the graphs. Here, the data is represented in the form of graphs. Some of the important operations of graph mining include extracting subgraphs, finding the reachability satisfying the given constraints and detecting the communities in graph.

Graph reachability is one of the basic operations that finds the existence of paths between the vertices of the given graph. But, in real time, there are queries which require certain constraints to be satisfied while finding the reachability of the graph. The constraints are usually the conditions on vertex attributes or edge attributes or both. For instance, in social networks, the vertex denotes the unique identifier assigned to a person, vertex attributes can be the name of a person and organization of the user. The edge attributes include relationships like *friendOf*, *colleagueOf* or *supervisorOf*. For example, the constraint reachability query is to find if Bob and Adam are from same organization and are colleagues in the organization.

An attributed graph is a graph that stores attribute information of vertices and edges. This attributed graph acts as an efficient modeling tool to represent information networks [88] [81]. Figure 4.1 illustrates an attributed graph. It includes different vertices of type *paper, conference, author and affiliation* and its vertex attributes are *location, keyword, age and state*. The edge attributes constitute labels such as *published*, *authorOf*, *affiliatedTo*, *citedBy*, *knows* and *supervisorOf*. Other edge attributes include *Vol*, *Issue*, *Order* and *StudentOf*. For instance, consider the vertex with label

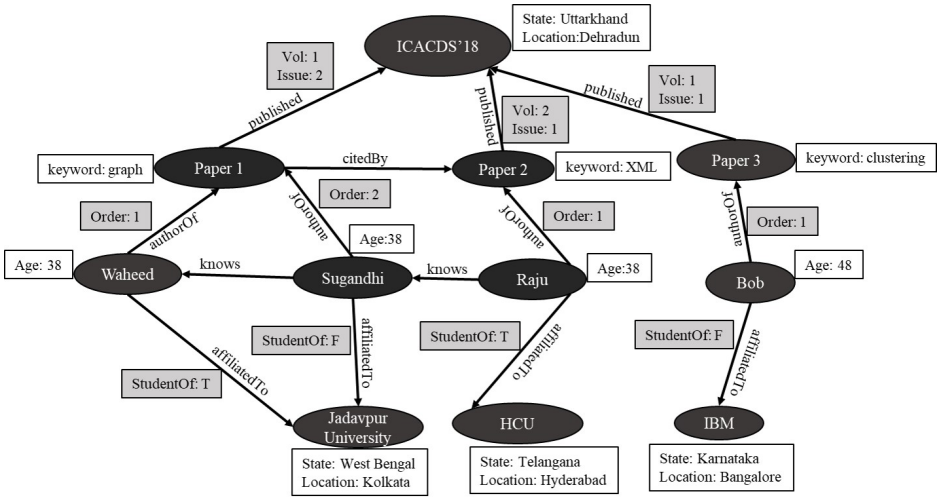


Figure (4.1) Example of an attributed graph

Paper1. Its attributes include *keyword* whose value is *graph*. Similarly, the domain of vertex attribute *age* is numerical values. The domain of vertex attributes *location, keyword* and *state* are categorical values. The domain of edge attributes *Vol*, *Issue* and *Order* are numerical values. The domain of edge attribute *StudentOf* is boolean value, i.e. True (T) or False (F).

A multidimensional constraint reachability query finds the existence of a path from the source vertex to the destination vertex satisfying the given attribute constraints. For instance, for the given attributed graph G of Figure 4.1, the query is to find if there is a path from Raju to Waheed whose *Age* is 38 through edge labels *knows*. The vertex constraint is *Age*=38 and the edge constraint is *knows*. From the figure, there exists a path between Raju and Waheed satisfying the given constraints.

One of the challenges of the constraint reachability is that we need to store both graph topology and attribute information while indexing the reachability. Another challenge is that there is no prior information of constraints before query processing. This problem is applicable for many real-time information networks like social networks, transportation networks and metabolic networks. These observations motivate us to find a faster

and efficient solution to solve the problem of multidimensional constraint reachability queries.

Duncan Yung et al. [88] developed a constraint verification approach to solve the multidimensional constraint reachability queries. They also implemented a heuristic search technique that offered direct passage across graph regions which are likely to satisfy attribute constraints from source to destination. The heuristic search involved the construction of a super graph. They used clustering based on BFS by choosing a random subset of vertices and their traversal forming clusters.

We observed from the state-of-the-art literature [88] that we can further optimize the hashing through Murmur hash function which has least collision. We also observed that there is a need to identify an efficient clustering technique that considers both graph topology and attributes information while clustering. Furthermore, we observed that there is a scope of extending the problem of MCR queries by finding the resultant paths.

We enhanced the heuristic search [88] by using optimized hashing to handle multidimensional attributes and proposed an efficient clustering technique based on matrix factorization to detect the clusters for efficient supergraph construction. In addition, we extended the MCR queries problem by finding the resultant paths.

Our proposed solution is based on heuristic search that considers both graph topology and attribute information while creating super graph for the given attributed graph. Thus, we can solve the constraint reachability queries faster for even large attributed graphs.

4.1.1 Assumptions

The assumptions in this chapter include

(a) We assume that the vertex attribute values and edge attribute values are single and discrete. We also assume that all vertices are of same type.

(b) We assume that if reachability exists, it is found along the super path of the super graph.

4.1.2 Findings

The main findings of this chapter include

- We find that the hashing used in [88] can be optimized as there is least chance of hash collision. We adopted the hashing [88] and optimized the usage of hashing during index retrieval in our proposed hashing based BFS search technique.
- We find that heuristic search developed by Yung et al. [88] had vague description of computation of probability cost for super vertices based on attributes. They have used random clustering based on BFS for constructing supergraph which does not consider graph topology and attributes information while clustering. Besides, we observed that the technique had a limitation of possible false negative outcomes.

4.1.3 Contributions

The main contributions in this chapter include

- To overcome the limitations, we performed comprehensive literature survey on recent structural and attributed graph clustering techniques [37], [92], [83], [84], [65], [12], [85]. We identified an efficient structural and attributed graph clustering technique [37] which is based on matrix factorization and applied during super graph construction to solve multidimensional constraint reachability queries.
- We computed the optimal number of clusters by applying gap statistic [77] for ANCA clustering and evaluated the proposed techniques.

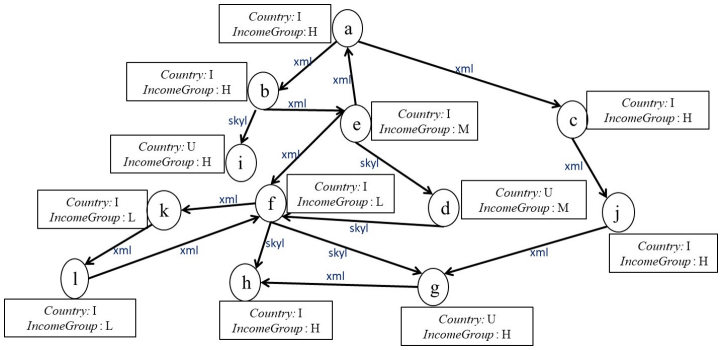


Figure (4.2) A toy dataset of an email network

- We solved the multidimensional constrained reachability queries by computing the path information instead of finding the existence of paths [88].
- We proposed an extended heuristic search technique to reduce the false negative outcomes.
- We compared our proposed techniques to constrained BFS and heuristic search with naive clustering [88] to solve multidimensional constraint reachability queries on real and synthetic datasets.
- We evaluated the accuracy of our proposed techniques using false negative ratio and tested the scalability for large graphs.

4.2 Preliminaries

Definition 1.(Attributed Graph) *"An attributed graph, G, is a graph denoted as G=(V, E, V_a, E_a), where V is a set of vertices, $E \subseteq V \times V$ is a set of edges, V_a is a set of vertex-specific attributes and E_a is a set of edge-specific attributes"* [81].

Let $V_a = (V_{a1}, V_{a2}, ..., V_{ax})$ is a set of x vertex-specific attributes. For each vertex $p \in V$, there exists a multidimensional tuple $V_a(p)$ denoted as $V_a(p) = (V_{a1}(p), V_{a2}(p), ..., V_{ax}(p))$. Let $E_a=(E_{a1}, E_{a2}, ..., E_{ar})$ is a set of r edge-specific attributes. There is a multidimensional tuple $E_a(q)$ denoted as $E_a(q)= (E_{a1}(q), E_{a2}(q), ..., E_{ar}(q))$ for every edge $q \in E$.

For instance, let us consider the attributed graph for an email network as shown in figure 4.2. Let the vertex attributes be *Country* and *IncomeGroup*. The domain of attribute *Country* is {India (I), United Kingdom (U)} and the attribute *IncomeGroup* is {High (H), Medium (M), Low (L)}. The domain of edge attribute for communication content is {XML (*xml*), Skyline(*skyl*)}. Thus, for vertex *'a'*, $V_{Country}(a)=I$ and $V_{IncomeGroup}(a)=H$. Similarly, the edge attribute between vertices *'a'* and *'c'* is *xml*.

Table 4.1 shows the different notations used in this chapter with their description.

Table (4.1) Notations

Notation	Description
V_a	Set of vertex attributes
V_a(p)	Set of vertex attributes values for vertex p
E_a	Set of edge attributes
E_a(q)	Set of edge attributes values for edge q
CV_a	Constraints on Vertex attribute values
CE_a	Constraints on Edge attribute values
G(V, E, V_a, E_a)	Attributed graph
G_s	Super graph
SV	Super Vertex
SE	Super Edge
SP	Super Path

4.2.1 Problem statement

Definition 2. (Multidimensional Constraint Reachability) "Given an attributed graph G, a source vertex s, a destination vertex t, vertex constraint CV_a, and edge constraint CE_a, the multidimensional constraint reachability query on attributed graph verifies whether s can reach t under vertex and edge constraint CV_a, CE_a" [88].

In this thesis, we use MCR query in short, to denote the term **M**ultidimensional **C**onstraint **R**eachability query. We define Multidimensional Constraint Reachable path (or MCR path) as the resultant path from the given source vertex to the destination vertex while satisfying the vertex and/or edge attribute constraints.

Let us consider the MCR query *q1('a', 'j', 'I:H', 'xml')*, for the attributed graph of Fig. 4.2. The given MCR query *q1* returns true as the source vertex 'a' can reach the destination vertex 'j' via vertex 'c' while satisfying the given vertex constraints 'I:H' and edge constraint 'xml'. Thus, the MCR path is {'a', 'c', 'j' }. Consider another instance of MCR query *q2('b', 'c', 'I:M', 'xml')*. The MCR query *q2* returns no path as the source vertex 'b' cannot reach 'c' as well as the given constraints are not satisfied.

The objective of our research is not only to find the resultant paths for MCR queries faster but also to propose a scalable solution based on clustering for large attributed graphs. We observed that we can optimize hashing for faster hash generation and constraint verification. We identified that there is need to find an efficient graph clustering algorithm that considers both graph topology and attributes while clustering. Thus, we solve MCR queries by optimizing hashing and proposing the efficient clustering technique in our proposed approaches described in section 4.4 and section 4.5.

4.3 Related Work

In this section, we describe the survey related to constraint reachability techniques, attributed graph clustering techniques and we derived important observations to solve efficiently and effectively.

4.3.1 Constraint reachability techniques

Many graph reachability techniques are developed in literature which include 2-hop [28], 3-hop [47], Dual labeling [80] and Path-Tree cover [9]. But, these indices do not include attribute information. Hence, the reachability techniques cannot be applied directly to solve the constraint reachability queries.

Ruoming Jin et al. [45] introduced formally the problem of Label Constraint Reachability (LCR) query which is a special case of attribute constraint reachability queries. They developed spanning tree based solution to solve LCR queries. With this cited state-of-the-art, we further performed extensive survey about different types of constraint reachability queries and techniques in chapter 3. Besides, we developed landmark index based path indexing and query processing technique [20] to find bounded paths for LCR queries in case of edge labeled weighted directed graphs.

An attributed graph acts as a modeling tool to represent information networks [88] [81]. Sakr et al. [68] developed G-SPARQL, a query execution engine with the defined algebraic operators on the graph by using join operations to find the reachability for large attributed graphs. They have designed a model that stored the topology of the graph in main memory and accessed the attributes of the graph from the secondary memory. The attributes from the secondary memory are stored in fully decomposed model which includes unique table for storing the unique vertex attributes and edge attributes in separate tables. Graph pattern matching queries are mainly solved by SPARQL-based systems.

We observed that Yung et al. [88] developed hashing based index instead of fully decomposed model [68] to store vertex attributes or edge attributes for attributed graphs. The hashing based index involves assigning unique hash index for group of vertex attributes or edge attributes. The attributes

and the corresponding hash values are stored in secondary storage. They have also designed BFS based heuristic search using random clustering to solve the multidimensional constraint reachability queries.

4.3.2 Attributed graph clustering techniques

Zhou et al. [92] designed **S**tructure and **A**ttribute (SA-Cluster) clustering which is one of the prominent attributed graph clustering technique based on random walks over augmented attributed graph. SA clustering is limited to small networks with few attribute values. Xu et al. [84] developed Bayesian model based approach to cluster attributed graphs. But, we observed that this approach is slow and not scalable.

Z.Wu et al. [83] developed **S**tructure and **A**ttributes using **G**lobal structure and **L**ocal neighborhood features (SAGL) clustering algorithm. SAGL clustering considers both global importance of the vertex and local neighbours structure while assigning weights to different topological links. SAGL clustering technique is faster than SA clustering as the former technique doesn't increase the size of attributed graph, yet uses both global importance of the vertex and attribute information to find clusters. We have observed that though SAGL clustering [83] is faster technique than SA clustering to find the clusters in an attributed graph, it relies on SA clustering to find the attribute similarity between the vertices.

Issam Falih et al. [37] developed Attributed Network Clustering (ANCA) algorithm that is based on matrix factorization of both graph topology and vertex attributes. Falih et al. [37] observed that social networks are dense and hence require high attribute similarity factor whereas road networks need a balanced attribute similarity and topological similarity metric while computing vertex similarity. Topological distance metric can be categorized into neighborhood based metric and path based metric. Based on type of attribute data (categorical/numerical/binary), the attribute similarity

measure can be the euclidean distance computed between the pair of vertices [33].

Falih et al. [37] developed ANCA clustering algorithm by considering shortest path metric for topological measure and Euclidean distance for attribute similarity. Then, matrix factorization is applied on both topological and attribute similarity measures. Finally, they use k-means clustering on the resultant matrix to form k clusters.

Guo Qi et al. [65] used matrix factorization based technique on edge content to detect communities. Yang et al. [85] developed non-negative matrix factorization based model to identify disjoint or overlapping communities at large scale. Amin et al. [12] developed matrix factorization and gradient descent based technique to identify polarization and clusters in social networks specifically Twitter.

From the literature, we observed that Yung et al. [88], [89] used a probability cost metric by sampling attributes for each super vertex that is vaguely mentioned. Besides, we observed that matrix factorization is a standard technique that has scope to find similarity by considering graph topology as well as vertex /edge attributes. Hence, we apply matrix factorization in supergraph construction without the probability cost metric and develop a heuristic based BFS search to solve the problem of MCR using hashing.

4.4 Proposed Approach: Heuristic search using Hashing and Matrix Factorization

In this section, we describe our proposed technique to solve the problem of MCR queries. We adopt the hashing and heuristic search developed by Yung et al. [88] to solve the multidimensional constraint reachability

queries. First, we perform pre-processing which includes hashing the vertex attributes or edge attributes and construction of supergraph based on attributed graph clustering [37]. We observed that Yung et al. [89] used a probability cost metric by sampling attributes for each super vertex which is vaguely mentioned. We identified an efficient attributed graph clustering algorithm [37] which is based on matrix factorization. We use the clustering algorithm [37] to construct the super graph.

In our proposed approach, we use the supergraph without considering any probability cost metric. We optimize the usage of hash index and use the graph clustering based supergraph to implement our proposed heuristic based BFS approach and efficiently solve multidimensional constraint reachability queries.

4.4.1 Hashing based index

During pre-processing, first, we construct an attribute hash index by collating attribute values of every vertex into a single string s_a. Every unique s_a is compressed to a hash value and stored in primary storage for answering queries. This hash value is mapped to its vertex. For instance, consider vertex attribute values of vertex *'b'* in figure 4.2, i.e., V_a(b)={I, H}. The resultant hash value computed for collated attribute values "I, H" is 2555692664 as shown in Table 4.2. Similarly, for every vertex and edge, the corresponding hash values for the attribute values are computed and stored in primary memory.

Secondly, the hash value for the given vertex/edge attribute constraints of the given query is computed. This hash value is verified against stored hash values in primary memory withot approaching the secondary storage. Hence, it leads to faster query procesing.

Algorithm 5 describes hash index construction of vertex attributes for an attributed graph. First, it gets attribute set *aio*, of vertex *u*, from secondary

Algorithm 5: HashIndexConstruct

```
Input  : Attributed graph G
Output: Hash Index, hInd
procedure HashIn(G,hInd)
for all u ∈ G do
    h ← GetHashAttr(u, G)
    aio ← AttrIO(u, G)
    if hashaddr(h)=NULL then
        Generate addr from hInd and aio
        Set count=1
        hashaddr[h]=addr
    else Append aio to addr in hInd and update count of the hash value
    hashaddr[h]=addr
```

memory. Then it checks whether its hash value h is already present in *hInd*. If it is NULL, then the new hash value of u, corresponding set of attribute and *count* is stored in *hInd*. If hash value is already present in *hInd*, then the new set of attributes are appended to the already present set and *count* is incremented by one. A non-cryptographic hash function like Murmur hash function [2] is used to generate hash values for attributes. Murmur hash function has no hash collision. During heuristic search, the constraints of vertex in *hInd* are verified with the constraints given by the user by retrieving the hash of constraints and comparing with the hash values stored in primary memory. This reduces the need to access secondary memory for multidimensional attributes verification.

For instance, let us consider the attributed graph of Fig. 4.2. Table 4.2 shows the computed hash values for vertex attribute combinations of Fig. 4.2. The *count* variable with value one indicates the assignment of unique hash value to each combination. The hash value is mapped to every vertex corresponding to its attribute values' combination and stored in primary memory as array.

Table (4.2) Hash Index

vattrHash	attr	*count*
1071913501	U, M	1
1139838478	I, L	1
2555692664	I, H	1
2608081465	U, H	1
29059796901	I, M	1

4.4.2 Super graph construction

We divide the graph based on clustering and construct a structure called super graph.

Definition 3. (Super Graph): A Super graph G_s is a directed graph constructed by computing super vertices and super edges for the given attributed graph G.

Definition 4. (Super Vertex): A super vertex, SV_i, is a vertex in G_s such that every vertex p of G belongs to only one super vertex SV_i. Thus, $\forall\ p \in$ V in G, $p \in SV_i$, $p \notin SV_j$, if i $\neq$ j where SV_i, $SV_j \in G_s$.

Definition 5. (Super Edge): A super edge SE_{ij}, is a directed edge in G_s formed between the super vertices SV_i and SV_j. This edge is formed only when, for any pair of vertices (p, q) $\in$ G such that p $\in SV_i$ and q $\in SV_j$, there exists an edge between p and q. Thus, if there exists an edge e(p, q) $\in$ E in G, p $\in SV_i$, q $\in SV_j$ and i $\neq$ j then $\exists$ SE_{ij} (SV_i, SV_j) $\in G_s$.

Definition 6. (Super Path): A super path, SP_i, is a simple path in G_s formed by sequence of super vertices (SV_1, SV_2, ..., SV_d) such that (SV_{i-1}, SV_i) $\in G_s$.

For instance, Fig. 4.3 shows the super graph for fig. 4.2. Thus, the super vertices include SV_1, SV_2, SV_3 and SV_4. The super edges are { (SV_1, SV_3), (SV_3, SV_2), (SV_3, SV_1), (SV_4, SV_2), (SV_3, SV_4) }.

During super graph construction, we choose K vertices as super vertices from K clusters using an efficient clustering algorithm. Algorithm 6 describes the super graph construction using clustering based on matrix factorization adopted from [37]. We further improved [37] by applying gap-statistic [77] to find the optimal K value.

- In the algorithm, first the subset of vertices are identified as seeds. The seeds are selected by considering top 15% of vertices by using centrality measures such as highest degree, closeness centrality, page rank and eigen vector centrality [37]. The seeds also include the outlier vertices for coverage by considering 5% of vertices with the least centrality [37].
- Next, we compute the topological matrix for the vertices and seeds based on shortest path distance between them. This matrix is normalized and singular value decomposition is applied.
- We find the attribute similarity between the vertices by computing euclidean distance [33] between them. The euclidean distance between the two vertices $u,\ v \in V$ is given by equation 4.1.

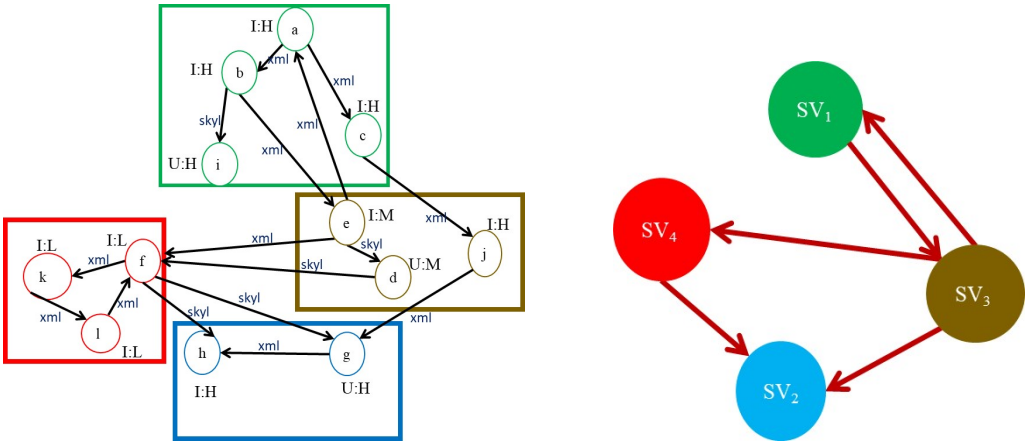
$$d(u,v) = \sqrt{\sum_{j=1}^{t}(|A_j(u) - A_j(v)|)^2}, \forall u,\ v \in V \tag{4.1}$$

- We use matrix factorization on attribute similarity between the vertices.
- We join the topological similarity and attribute similarity factorized vectors to get the decomposed matrix U and normalize it.
- Then, we apply k-means clustering on the decomposed matrix U to get resultant K clusters.
- If there exist $p \in SV_i$ and $q \in SV_j$ such that there exists edge from p to q in G, then we add the super edge SE_{ij} to the super graph G_s as described in the algorithm.

Algorithm 6: SuperGraphMF

Input : Attributed graph G(V, E, A), Number of clusters K
Output: Super Graph G_s with K Clusters
Select subset of vertices as seeds *S*.
Compute topological closeness between the vertices and seeds using shortest path metric
Form topological matrix, M_{topo}(v,s)=SPath(v,s), ∀v ∈ V and ∀s ∈ S.
Apply singular value decomposition on M_{topo} to get U_{topo}.
Compute attribute similarity matrix M_{attr} between the vertices using Euclidean distance based on equation 4.1.
Apply matrix factorization on M_{attr} to get U_{attr}.
U← U_{topo} + U_{attr}
Normalize each of U's rows defined by U_{ij}=$U_{ij}/\sqrt{\sum_j U_{ij}^2}$
Apply k-means clustering on U to get K clusters.
Construct Super graph G_s (V_s,E_s) with K vertices V_s={sv_1, sv_2,..., sv_K}with each cluster considered as super vertex sv_i
for *each edge e(u,v)* ∈ *E* **do**
    svu=findSuperVertex(u)
    svv=findSuperVertex(v)
    if *(svu≠svv)* **then**
        Add edge se_i=(svu, svv) to E_s
        i←i+1

Illustration of super graph construction Let us consider the attributed graph of Fig. 4.2. The resultant set of seeds based on the centrality measures is { *'a'*, *'e'*, *'f'*, *'h'*, *'i'*}. We use singular value decomposition as described in the SuperGraphMF algorithm by considering both topological distance and attribute similarity. Let us assume the number of clusters K=4. We can also compute an optimal K value by applying gap statistic [77]. The resultant clusters with K=4 after applying k-means algorithm are the subsets {'a', 'b', 'c', 'i'}, { 'g', 'h'} , {'d', 'e', 'j'} and {'f', 'k', 'l'}. These clusters are denoted by vertices as SV_1, SV_2, SV_3 and SV_4 respectively which form the super vertices as shown in figure 4.3a. We add the super edge based on existence of edge between vertices of the clusters. For instance, in Fig. 4.2, there exists edge between vertices *'e'* and *'f'*. The super vertex of *'e'* is SV_3, while the super vertex of *'f'* is SV_4. Hence, we add the super edge (SV_3, SV_4). Thus, the resultant super graph is constructed



(a) Clusters (b) Super graph

Figure (4.3) Clusters and the resultant super graph

as shown in Fig. 4.3b.

4.4.3 Proposed heuristic search technique

We solve the multidimensional constraint reachability query using our proposed heuristic search which is based on optimized hashing and efficient attributed graph clustering. Algorithm 7 describes the heuristic search to find the existence of reachability between the given vertices satisfying the user constraints. Given the source vertex, destination vertex and vertex constraints, the super vertex of source vertex is first identified from the super graph. Then, the super vertex of destination vertex is identified. These super vertices are checked for the existence of super path between them in the supergraph. The super vertices of source vertex, destination vertex and their intermediate super vertices along the super path are stored in *sPath*. BFS along with *sPath* information finds the existence of reachability by verifying the user given constraints through hashing (from Algorithm 5) while finding the reachability between the vertices.

Heuristic In the proposed approach, we assume that if reachability exists, it is found along the path *sPath*. By including this heuristic, we can find

Algorithm 7: HeuristicSearchMF

```
Input  : Attributed graph G, source vertex s, destination vertex t, Vertex
         Constraint C_v, Super Graph G_s, Hash index.
Output: rp_i/"No path"
Let q be queue
Enqueue (s)
superSrc←findSuperVertex(s)
superDst←findSuperVertex(t)
sPath ← findPathBFS(superSrc, superDst, G_s)
while isEmpty(q) do
    Dequeue v
    if (visited[v] = true) then
        continue
    for v' ∈ v.adjList do
        if (visited[v']=true) then
            continue
        superiv← findSuperVertex(v')
        if (superiv ∈ sPath) then
            visited[v']=true
            if (CheckConstraint(v', hInd, C_v, G)=true) then
                if (v'=t) then
                    return rp_i
                Enqueue v'
    visited[v] ← true
return "No path"
procedure CheckConstraint(v, HashIndex, C_v)
hc←getHash(C_v)
hv←getHashAttr(v,G)
if (hc≠hv) then
    return false
if (getCount(HashIndex,hc)=1) then
    return true
else Attr← Get attributes from secondary storage
if (CheckAttrConstraint(Attr,C_v)=true) then
    return true
```

the reachability between the vertices faster as we traverse only the vertices that belong to the super vertices of the path *sPath* thus minimizing the search space. In the algorithm, we include the heuristic through finding super path (*sPath)* and verifying if each super vertex of adjacent vertex belongs to *sPath*.

Optimized Hashing In Algorithm 7, we optimize the hash retrieval through *CheckConstraint()* procedure. In this procedure, we retrieve the hash of given constraints and compare with the hash of vertex. If both hash values are same, we check the *count* by retrieving it from hash index. If *count* returned is one, we need not check the secondary storage and we declare the two hash values are equal and return true.

Illustration For instance, consider the MCR query *q1*(*'a', 'j', 'I:H'*). The super vertex of a is SV_1 and super vertex of j is SV_3. Since there exists path in the super graph from a to j, our proposed heuristic search technique then traverses only the vertices within super vertices SV_1 and SV_3. The vertex constraint is combined and its hash value is computed. While traversing, the hash value of the given vertex constraint is compared to the existing hash value in hash index table (Table 4.2). If the match exists, it traverses to the next adjacent vertices until the destination vertex (j) is reached. Thus, our proposed heuristic search technique returns the path {'a', 'c', 'j'} for the MCR query *q1*.

4.5 Extended Heuristic Search

In our proposed HeuristicSearchMF algorithm, we observe that there may exist path between two vertices that is not included in the super path of the constructed supergraph. To overcome this problem, we have modified our proposed approach by extending the heuristic to include those vertices whose super vertex has destination super vertex as the adjacent vertex.

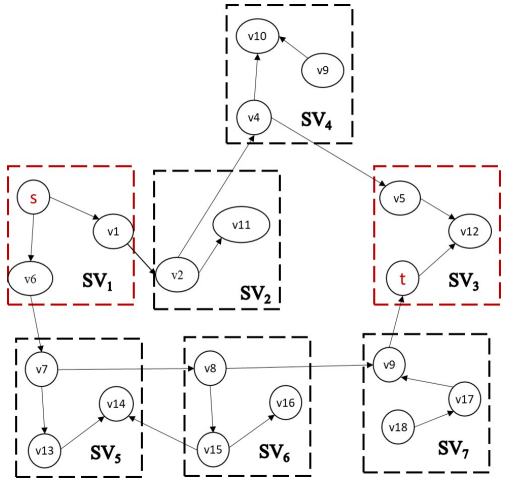


Figure (4.4) Example for extended heuristic search technique

Algorithm ExtendedHeuristicSearchMF describes our proposed extensions to the previously proposed heuristic search technique. In our proposed technique, we have extended the heuristic by including those vertices whose super vertex has destination super vertex as the adjacent vertex. For each adjacent vertex traversed, we find its super vertex and check if it is neighbor to the super vertex of the destination vertex. Thus, our proposed extended heuristic search technique can find most of the missed reachable paths resulted by using HeuristicSearchMF algorithm.

Let us consider the example of Fig. 4.4. To find the path from source vertex *'s'* to destination vertex *'t'*, we first compute the super path between the super vertices of *'s'* and *'t'*. The resultant super path is {SV_1, SV_2, SV_4, SV_3 }. When we execute the HeuristicSearchMF algorithm, we cannot reach the destination vertex through the super path. But, in the ExtendedHeuristicSearchMF algorithm, we can reach the destination vertex via intermediate vertex '*v9*' whose super vertex SV_7 is adjacent to the destination super vertex SV_3. Thus, the number of missed reachable paths using extended heuristic can be effectively reduced.

Algorithm 8: ExtendedHeuristicSearchMF

```
Input  : Attributed graph G, source vertex s, destination vertex t, Vertex
         Constraint C_v, Super Graph G_s.
Output: rp_i/"No path"
Let q be queue
Enqueue (s)
superSrc←findSuperVertex(s)
superDst←findSuperVertex(t)
sPath ← findPathBFS(superSrc, superDst, G_s)
while isEmpty(q) do
    Dequeue v
    reached← false
    if (visited[v] = true) then
        continue
    for v' ∈ v.adjList do
        if (visited[v']=true) then
            continue
        superiv← findSuperVertex(v')
        for v'' ∈ v'.adjList do
            superiv2← findSuperVertex(v'')
            if (edgeExists(superiv2,superDst,G_s) OR superiv2=superDst) then
                reached←true
        if ((superiv ∈ sPath)OR reached=true) then
            if (CheckConstraint(v', G_h, C_v)=true) then
                if (v'=t) then
                    return rp_i
                Enqueue v'
                if (reached=true) then
                    break
    visited[v] ← true
return "No path"
```

4.6 Experiments and Results

In this section, we describe the datasets used for experiments, the parameters set, the environment of experiments and the results of our proposed techniques.

4.6.1 Experiment setup

During experimentation, for hashing, we used Murmur hash function [2]. We assume the number of super vertices (K) to be 15 based on size of the dataset and constructed the supergraph. Besides, we also computed optimal K value by applying gap statistic [77].

Table 4.3 describes the different parameter settings used in the experiments adopted from [88]. We used vertex attributes and vertex constraints throughout our experiments. Besides, we used edge attributes and edge constraints along with vertex constraints for real dataset. Table 4.4 shows the different vertex attributes and edge attributes assigned to the datasets. We generated 25 to 100 MCR true queries (whose path length is greater than 1) for the real and synthetic datasets by randomly selecting attribute values and verifying the constraints through constrained breadth first search and traversal.

Table (4.3) Parameter values

Parameter	Value
Number of Vertex Attributes	**2**, 3
Number of Edge Attributes	3
Number of Super-vertex (K)	**15**, 50
Number of Vertex Constraints	2
Number of Edge Constraints	1

4.6.2 Baselines

We evaluated the efficiency of our proposed approaches (HeuristicSearchMF and ExtendedHeuristicSearchMF algorithms) by comparing with two existing techniques as follows :

(1) Breadth First Search or BFS [30] in which the constraints are checked while performing breadth first search from source vertex till the destination vertex is reached. This is considered as *ConstrainedBFS*.

(2) Yung et al. developed BFS based heuristic search technique using naive clustering [88].

To solve MCR queries, we have the following two approaches with respect to proposed techniques.

(1) We can solve only by using hashing mechanism described in section 4.4.3 (with optimized hashing only). We denote this approach by *Constrained-Hash*.

(2) We can solve using both hashing and clustering mechanism to obtain the resultant path efficiently. We consider *HMF* as the implementation of our proposed HeuristicSearchMF algorithm described in section 4.4. We consider *EHMF* technique as the implementation of our proposed Extended-HeuristicSearchMF algorithm described in section 4.5.

Table (4.4) Vertex attributes and edge attributes

Vertex Attribute	**Domain Size, Distribution**
Country	5, *uniform*
Region	3, *uniform*
Gender	2, *uniform*
Edge Attribute	**Domain Size, Distribution**
Trustlevel	4, *real*
isFamily	2, *uniform*
isFriend	2, *uniform*

4.6.3 Datasets description

Table 4.5 summarizes the real and synthetic datasets used for experiments. We generated synthetic graphs from SNAP [53]. We assigned randomly vertex attribute values for the vertices and edge attribute values for the edges. Table 4.4 states the synthetic vertex attributes that are assigned randomly to the datasets.

Table (4.5) Datasets overview

Graph	**|V|**	**|E|**
Robots [1]	1724	3596
Erdos-Renyi [53]	1000	2000
ForestFire [53]	5000	12620
	4000	10252
	3000	7751
	2000	4865
	1000	2833

4.6.3.1 Robots

Robots is a real trust network [1] with edge labels that denote the level of trust interaction between the users. We pre-process the dataset by assigning unique identifier to the vertices, resulting in 1724 vertices and 3596 edges. Each vertex has synthetic attributes whose values are randomly assigned as shown in Table 4.4. Each edge has *Trustlevel* as the real attribute whose value is derived from the data set. The trust level can be Master (M), Apprentice (A), Journeyer (J) or Observer (O). Besides, we assigned two synthetic attributes whose values are randomly assigned for every edge of the Robots dataset.

4.6.3.2 Erdos-Renyi Graph

Erdos-Renyi (E-R) graphs are the synthetic graphs that follow power law distribution [13]. These graphs have their degree near uniformly distributed. We generate E-R graph using SNAP [53] with number of vertices set to 1000 and

maximum degree for each vertex set to 2. Besides, we assign two attributes described in Table 4.4 for each vertex with randomly assigned values within the domain.

4.6.3.3 ForestFire Graph

ForestFire graphs are the synthetic random graphs [54]. The ForestFire model graphs exhibit the properties of time-evolving real-world graphs [54] that include densification of graphs and decreasing effective diameter. We generate ForestFire graphs using SNAP [53] for testing scalability with the number of vertices varying from 1000 to 5000. The other parameters including forward probability is set to 0.4 and backward probability is set to 0.2 [53] and maximum degree for each vertex set to 2. Besides, we assign two attributes for each vertex as described in Table 4.4.

4.6.4 Results and analysis

We evaluated the efficiency of our proposed techniques based on average execution time and false negative ratio. The average execution time for true queries denotes the average time taken to execute given set of true queries. The MCR true queries are the constrained reachable queries that have atleast one path between the given vertices. We evaluated the accuracy of our proposed techniques based on false negative ratio. The false negative ratio (τ) is defined as *"The fraction of queries which fail to return any path that satisfies the given constraint, although at least one such path exists"* [57].

Table (4.6) Average execution time of true queries for Erdos-Renyi graph with only vertex constraints

S.No.	Technique	Average Execution time (s)	τ
	Proposed		
1	*HMF*	0.01280	0.65
2	*EHMF*	0.01290	0.05
3	*ConstrainedHash*	0.00009	0
	Existing		
4	*ConstrainedBFS*	0.01162	0
5	*Yung et al. [88]*	0.013038	0.55

Table 4.6 shows the average execution time and false negative ratio on MCR true queries using our proposed techniques compared to constrained BFS for E-R graphs. In the Table 4.6, we find that there is considerable decrease in the false negative ratio for our proposed extended heuristic technique i.e. *EHMF* than that of *HMF*. We also observed that *ConstrainedHash* technique executed faster for MCR queries than the other techniques.

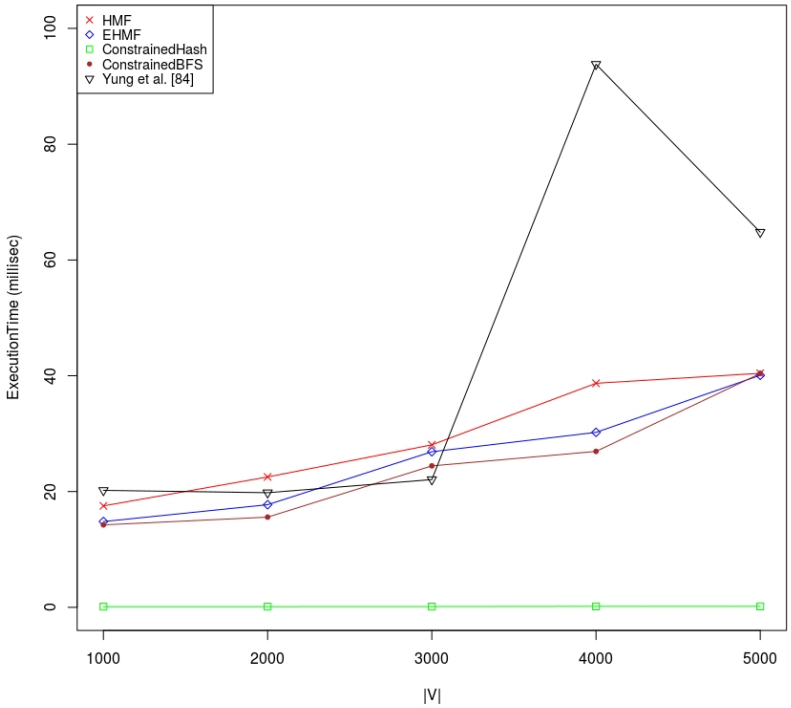


Figure (4.5) Varying graph size for **Forest Fire** synthetic graph

Figure 4.5 shows the average execution time for Forest Fire graphs with varying graph size from 1000 vertices to 5000 vertices. From Fig. 4.5, we observe that *ConstrainedHash* technique has the least average execution time than our other proposed techniques and *ConstrainedBFS*. We also observe that our proposed techniques executed faster than the heuristic search based on naive clustering ([88]). Besides, the false negative ratio varied from 0.04 to 0.36 for the MCR true

queries on ForestFire graphs using our proposed *HMF* approach. By using our proposed *EHMF* and *ConstrainedHash* techniques, the false-negative ratio is 0.

Table (4.7) Average execution time of true queries for Robots dataset with only vertex constraints

S.No.	Technique	Average Execution time (s)	τ
	Proposed		
1	*HMF*	0.214636	0.32
2	*EHMF*	0.229042	0
3	*ConstrainedHash*	0.00011	0
	Existing		
4	*ConstrainedBFS*	0.301144	0
5	*Yung et al. [88]*	0.092135	0.6

Table 4.7 shows the average execution time and false negative ratio on MCR true queries using our proposed techniques compared to constrained BFS for Robots dataset. We computed optimal K value for Robots dataset by applying gap statistic [77]. The resultant computed K value is 15. We generated 100 MCR true queries for evaluation of Robots dataset. From Table 4.7, using our proposed *HMF* approach, the false negative ratio (τ) is 0.32. Based on our proposed extended heuristic technique, i.e., *EHMF*, the false negative ratio reduced to 0.

Table (4.8) Average execution time of true queries for Robots dataset with vertex constraints and edge constraints

S.No.	Technique	Average Execution time (s)	τ
	Proposed		
1	*HMF*	0.4684	0.8
2	*EHMF*	0.178258	0
3	*ConstrainedHash*	0.0151944	0
	Existing		
4	*ConstrainedBFS*	0.300613	0
5	*Yung et al. [88]*	0.034883	0.96

Table 4.8 shows the average execution time and false negative ratio for 25 true MCR queries with vertex constraints and edge constraints for Robots dataset.

We choose the *TrustLevel* as edge constraint and generated MCR true queries based on constrained BFS. For clustering, we assumed K to be 50. We observed that our proposed techniques have lesser false negative ratio than that of existing technique [88]. Besides, we observed that *ConstrainedHash* technique executed faster for MCR queries than the other techniques.

4.7 Conclusions

In this chapter, we studied the problem of MCR queries on attributed graphs. We solved this problem by using our proposed heuristic search technique that includes hashing and clustering. We computed hash value for multidimensional attribute values for faster comparison of attributes. We used matrix factorization based graph clustering on the attributed graph to construct supergraph. We used shortest path from super graph and hashing for checking constraints in our proposed approach to efficiently solve the multidimensional constraint reachability queries for large graphs. Besides, we proposed an extended heuristic search technique that increased the accuracy. From the experiments and evaluation, we find that our proposed techniques are scalable and solved MCR queries efficiently.

Chapter 5

Frequent Subgraphs and Frequent Subpaths

Finding frequent subgraphs from the dynamic graph streams can be a challenging task as streams are non-uniformly distributed and are continuously needed to be processed. From these frequent subgraphs, we can extract unknown and useful information. In this chapter, we not only developed techniques to extract frequent subgraphs from graph stream data but also applied the techniques to the special case of finding frequent subpaths from the data of paths. This chapter publications are listed below. In section 5.1, we introduce the problem of finding frequent subgraphs and frequent subpaths. Section 5.2 deals with the preliminaries and problem definitions. Section 5.3 deals with literature survey of frequent subgraph algorithms and extracting frequent subpaths from sequence of paths. Section 5.4 describes our proposed static and dynamic techniques to extract frequent subgraphs. Besides, it discusses the enhancements to our proposed techniques by solving the problem of finding frequent subpaths from paths data for the directed graph. Section 5.5 describes the experiments and evaluation of results.

Bhargavi B, Swarupa Rani K., Rohit Kumar, and Sanmeet Kaur. "Static and Dynamic Techniques to Extract Frequent Subgraphs from Graph Stream Data" , to appear in *Proceedings of International Conference on Big Data, Machine Learning, and Applications* (BigDML), 2019.
Bhargavi B., and K. Swarupa Rani, "Finding Frequent Subgraphs and Subpaths through Static and Dynamic Window Filtering Techniques", *EAI Endorsed Transactions on Scalable Information Systems*, Vol. 7, No. 27, p. 13, EAI [DBLP Indexed and ESCI Indexed] ISSN: 2032-9407, Web of Sciences, 2020.

5.1 Introduction

In Big data era, we find large amounts of data generated from different data sources very fast. For instance, there are 22.2 million Twitter users in India and 152 million daily active Twitter users all over the world [3]. Data stream model deals with such Big data and data stream algorithms [22] make very few passes and take lesser space. Data streams constitute of structured, semi-structured and unstructured data which are time consuming as streams are continuous and are also unbounded. Massive graphs are rendered as streams of graphs to analyze and extract useful and unknown information. Graph streams as dynamic stream model has been studied in the literature [44, 61, 75] which are the sequence of 'm' edges between 'n' nodes with the edges being updated sequentially. Processing graph streams are challenging as they have large volume and are highly dynamic in nature.

During the process of solving constraint reachability queries, we observed the need and importance of finding frequent subgraphs and subpaths. In this chapter, we review the problem of finding frequent subgraphs from the graph streams. A frequent subgraph is a connected subgraph that occurs above the given threshold in the sequence of graph streams. The problem of finding frequent subgraphs is defined as follows: Given a sequence of graph streams and a minimum support threshold, the problem is to find the frequent subgraphs having useful information from the graph streams efficiently. One of the applications of finding frequent subgraphs can be in social networks [56]. For instance, we can derive the groups of users who are frequently communicating in the social network. In bio-informatics, based on the frequent interactions between molecules, we can predict protein functions and identify types of diseases.

We also solve another sub-problem of extracting frequent subpaths from sequence of paths. The applications for finding frequent subpaths can be in IP routing in which we can find the frequent paths of data flows across multiple networks. In a traffic network, we can find the paths/subpaths that are frequently traversed by commuters.

Alfredo Cuzzocrea et al. [31] proposed two algorithms to discover collections of frequent subgraphs, one of which is the direct 1-step algorithm based vertical

mining approach using Data Stream Matrix (DSMatrix). Besides, this approach [31] used sliding window technique to process the graph streams in finding the frequent subgraphs. This sliding window technique has the limitation of repeated calculations. To overcome this limitation, Kyoungsoo Bok et. al. [23] proposed an incremental frequent subgraph detection technique. The limitation of Kyoungsoo Bok et al. [23] approach is that their approach did not completely resolve the duplicate calculations.

We observed that the above solutions have certain limitations which include the partially resolved duplicate calculations. Another possible limitation is that frequent subgraphs in the past can be infrequent due to incomplete storage of edges in the sliding window. To overcome these limitations, we proposed static and dynamic approaches to find the frequent subgraphs. The key contributions of this work include

- Proposed static and dynamic techniques to extract frequent subgraphs
- Compared the proposed techniques with the conventional approach thus evaluating the efficiency
- We improved and proposed static approach by computing actual minimum support.
- We also proposed partition based static approach with actual minimum support for sequential and parallel environments.
- We improved and extended the dynamic sliding window filtering technique with variable batch size.
- We solved the sub-problem to find frequent subpaths from sequence of paths by applying our proposed static and dynamic techniques.
- We analysed our proposed static and dynamic techniques for efficiency on real and synthetic datasets.

The above key contributions of proposed approaches and its variations are also given in Table 5.1.

Table (5.1) Proposed approaches and its variations

Static Approach	Dynamic Approach
1. Single window with minimum support	1. Incremental approach with fixed batch size of graph data with relative support
2. Single window approach with actual minimum support	2. Incremental approach with variable batch size of graph data with relative support
3. Partition based approach with actual minimum support in sequential and parallel environments	

5.2 Preliminaries

Definition 1. (Graph stream) *"Graph stream is defined as the sequence* G_1, G_2, ... G_i . . .G_s, *where each graph* G_i *is a set of edges. We assume that the edge set* G_i *contains only a small fraction of the underlying nodes"* [8].

Definition 2. (Frequent Subgraph) A subgraph $G_s(V_s, E_s)$ is a part of a graph G(V,E) such that $V_s \subset V$ and $E_s \subset E$. A frequent subgraph is a connected subgraph that occurs above the given threshold (***th***) in the sequence of graph streams.

Definition 3. (Path) "*Given a graph G(V, E), a path p of length k from a vertex u to u′ is a sequence* (v_0, v_1, . . . , v_k) *of vertices such that* $v_i \in V$, $v_0 = u$ *and* $v_k = u'$ *and* (v_{i-1}, v_i) $\in E$ *for* $i = 1, 2, \ldots, k$" [41].

Definition 4. (Subpath) *"A path Q in G is said to be a subpath of P if Q =* (w_0, w_1, . . ., $w_{k'}$), *where* (w_0, w_1, . . . , $w_{k'}$) *is a contiguous sub-sequence of path P*(v_0, v_1, . . . , v_k), *i.e., if, for some i such that* $0 \leq i \leq i+k' \leq k$, *we have* $w_0 = v_i$, $w_1 = v_{i+1}$, . . . , $w_{k'} = v_{i+k'}$" [41].

Definition 5. (Minimum Support) Minimum support is defined as the threshold specified by the user.

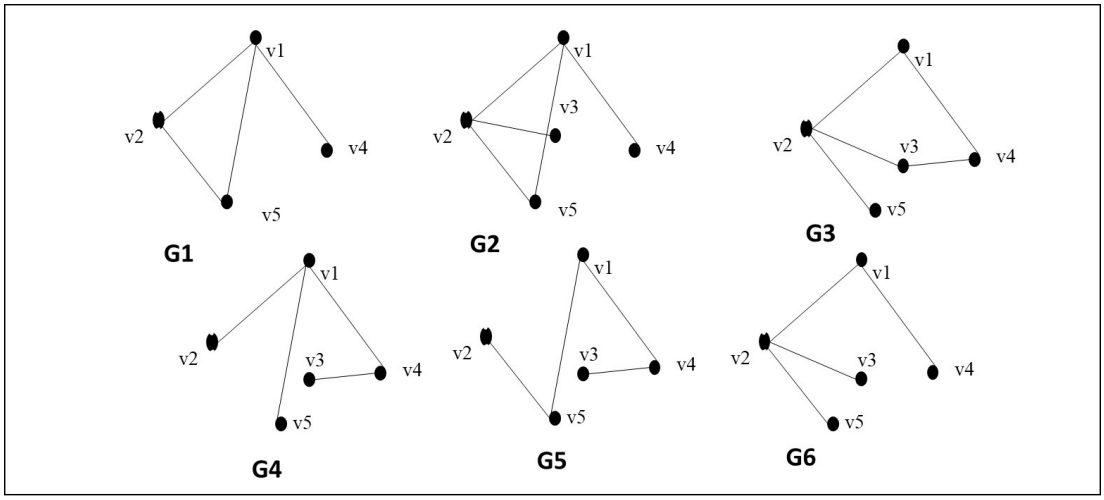


Figure (5.1) Sequence of graph streams G1, G2, G3, G4, G5, G6

Definition 6. (Actual Minimum Support) Actual minimum support is defined as the minimum support based on user's mining requirements which is appropriate to the database to be mined [91].

Definition 7. (Relative Support) We define the relative support as the partial minimum support assigned to subset of data. In this chapter, we adopt the filtering threshold [52] to find the relative support.

5.2.1 Problem definitions

In this chapter, we propose techniques to find frequent subgraphs from graph stream data and to find frequent subpaths from sequence of paths for a directed graph efficiently.

5.2.1.1 Finding frequent subgraphs from graph stream data

Given a sequence of graph streams, for a minimum support threshold (***th***), the problem is to find the frequent subgraphs from the graph streams efficiently.

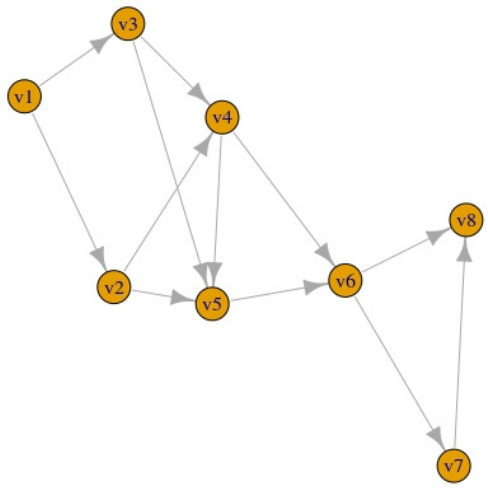


Figure (5.2) Directed graph

For instance, for the sequence of graph streams, shown in Fig. 5.1, with ***th***=3, the set of frequent subgraphs include { <(v1, v2), (v1, v4)>, <(v1, v2), (v1, v5)>, <(v1, v2), (v3, v4)>, < (v1, v4), (v3, v4) >, <(v1, v4), (v1, v5) > , <(v1, v2), (v1, v4), (v1, v5) >, < (v1, v2), (v1, v4), (v2, v3)>}

5.2.1.2 Finding frequent subpaths from paths data

Given a sequence of paths of a graph, for a minimum support (***th***), the problem is to extract frequent subpaths from paths data.

For instance, for the sequence of paths in Table 5.2 of the graph in Fig. 5.2, the set of frequent subpaths with ***th***=3 are {(v5, v6, v7), (v3, v5, v6)}.

Table (5.2) Data of paths

S.No.	Paths
1	(v1, v2, v5, v6)
2	(v1, v3, v5, v6, v7)
3	(v1, v2, v4)
4	(v1, v2, v4, v5)
5	(v2, v4, v5)
6	(v2, v5, v6)
7	(v2, v4, v6, v7)
8	(v4, v5, v6, v7)
9	(v4, v6, v8)
10	(v3, v4, v5, v6)
11	(v3, v5, v6)
12	(v3, v5, v6, v7)

5.3 Related Work

Massive graphs are considered as streams of data to analyze and extract useful information. Henzinger et. al. [42] were the first to introduce graph streams and they also worked on graph problems of paths and connectivity. Andrew McGregor [61] presented a detailed survey of graph streams. Due to the dynamic nature [40], [44] and the large volume of graph stream data, Nan Tang et. al. [75] proposed graph summarization sketch that can store frequent counts and paths of graph streams.

Alfredo Cuzzocrea et al. [31] studied various methodologies of mining dense patterns in graph streams and proposed probabilistic algorithms for determining such structural patterns effectively and efficiently. Alfredo Cuzzocrea et al. [31] presented two algorithms to extract frequent subgraphs - (i) Indirect 2-step algorithm (ii) Direct 1-step algorithm. Experimental results by Alfredo Cuzzocrea et al. [31] stated that mining with DSMatrix consumes lesser memory due to the information stored in a secondary storage device as they store the existence of edges in bit vectors.

Kyoungsoo Bok et. al. [23] observed that the algorithm developed by Alfredo Cuzzocrea et. al. [31] has a limitation of duplicate calculations. They introduced *slidenum* variable [23] to store the frequency of edges incrementally for batches of graph streams to resolve duplicate calculations.

Table (5.3) DSMatrix for graph streams of Fig. 5.1

Edge	G1	G2	G3	G4	G5	G6
(v1, v2)	1	1	1	1	0	1
(v1, v4)	1	1	1	1	1	1
(v1, v5)	1	1	0	1	1	0
(v2, v5)	1	1	1	0	1	1
(v2, v3)	0	1	1	0	0	1
(v3, v4)	0	0	1	1	1	0

From the literature, we observe that while finding frequent subgraphs, although sliding window based techniques execute fast, they may lead to loss of useful historical information. We also observe that we need to reduce duplicate calculations further. We observed that finding frequent subpaths from paths is another problem in the literature that can be related to the problem of finding frequent subgraphs. We identified and formulated ways to apply our proposed and extended techniques to find frequent subpaths from sequence of paths. Sumanta Guha [41] developed Apriori based technique to extract frequent subpaths from paths in an undirected graph. Schwartz et al. [69] studied demand of frequent subpaths in a transportation network traversed by several users. Hence, there is need to find techniques that discover frequent subgraphs and frequent subpaths efficiently by storing useful historical information.

5.4 Proposed Static and Dynamic Techniques for Finding Frequent Subgraphs

We have extended the direct 1-step algorithm of Alfredo Cuzzocrea et al. [31] by modifying the parameters of sliding window size and by using relative support. We have proposed static single window approach and dynamic approach of

Table (5.4) Characteristics of Proposed Static and Dynamic Approaches compared to Conventional Approach

Characteristic	Proposed approaches		Conventional
	Static	Dynamic	
Size	Entire graph stream data	incremental data	sliding window size
Minimum Support	user given minimum support and actual minimum support [91] can be used	relative support [52]	user given minimum support
Applicability	for existing data	present stream of data	present stream of data
Streams of data	single window	incremental in batches	sliding window
Loss of historical data	No loss of frequent subgraphs	No loss of significant frequent subgraphs	Loss of some significant frequent subgraphs
Time	faster technique in distributed and parallel environments	moderate technique without loss of significant information	faster technique with loss of information
Parallelism	can be applied by dividing the data and *minsup* into equal partitions	cannot be applied	cannot be applied
Distributed	can be applied by using MapReduce technique	cannot be applied	may not be applicable

sliding window to find frequent subgraphs from graph streams using DSMatrix [50]. Table 5.4 shows the characteristics of our proposed approaches w.r.t. size, environment and compatible features. The proposed approaches are explained in

the following subsections.

5.4.1 DSMatrix

Data Stream Matrix or DSMatrix constitutes assigning the presence (or absence) of each edge by a bit 1 (or 0) for each graph of graph stream data [31]. For instance, consider the sequence of graph streams in figure 5.1. Table 5.3 shows the contents of DSMatrix. We find the rowsum from DSMatrix to compute the frequency of every edge in graph stream data.

5.4.2 Static single window technique

In the static single-window, we consider the entire data set of graph streams as a single window. We describe the proposed static single window approach in StaticFreqSubgraph algorithm. In this algorithm, for the entire graph streams, it creates a DSMatrix. The rowsum of the DSMatrix is computed and compared to the minimum support threshold *minsup*. If rowsum is greater than or equal to *minsup*, the resultant edges are the frequent singleton edges.

Then, combination of edge pairs are found based on the neighbouring information (in list, N). The AND operation is performed between the k-frequent singleton edge combinations' bit vectors by checking whether the edges are present for each graph to form a bit vector for the combination. The sum of non-zero bits is computed and compared to the given minimum support to check if it is frequent or not. If the resultant sum is greater than or equal to *minsup*, the k+1-subgraph is frequent. Thus, the algorithm generates all the possible frequent subgraphs.

For example, for graph streams of Fig. 5.1 with *minsup*=3, the resultant frequent singleton edges are {(v1, v2), (v1, v4), (v2, v5), (v1, v5),(v2, v3),(v3, v4)}. The resultant frequent subgraphs for the graph streams of Fig. 5.1 generated using StaticFreqSubgraph algorithm are { <(v1, v2), (v1, v4)>, <(v1, v4), (v1, v5)>, <(v2, v3), (v2, v5)>, <(v1, v2), (v2, v3)>, <(v1, v2), (v1, v4), (v1, v5)> }.

Algorithm 9: StaticFreqSubgraph

Input : Edges in graph streams, total number of graphs in the whole data and *minsup* = minimum support threshold

Output: Set of frequent subgraphs

```
// m= number of graph streams, n= number of edges in the entire
   graph streams sequence
// Mat_A[m][n] = 2-D DSMatrix with m rows and n columns
// rowsum = count of 1s in a row of DSMatrix
```

Create neighbouring list, N for the graph streams by finding common vertex for the stream of edges

for *each graph* g_j **do**
  if *edge* $i \in g_j$ **then**
    Mat_A[i][j]=1
  else
    Mat_A[i][j]=0

Calculate *rowsum* for each row of Mat_A.

if *(rowsum(i)* $\geq$ *minsup)* **then**
  Edge i is frequent

else
  Edge i is not frequent

Join k frequent connected edges with common vertices using neighboring list N to get *k+1* frequent subgraphs , f_l for all $k \leq$ m.

Compute freq(f_l) by AND of the bit vectors of k recurrent edges for the graph streams in the DSMatrix

if *freq*(f_l)$\geq$*minsup* **then**
  f_l is frequent.

else
  f_l is infrequent.

5.4.3 Dynamic approach of sliding window technique

In this approach, we proposed a dynamic sliding window technique that is described in DynamicFreqSubgraph algorithm. DSMatrix for each batch is created and a relative support is applied. For example, if batch 1 contains 3 graph instances, then we calculate 40% of frequent edges of the 3 graph instances and store it in the map. This technique is incrementally applied for the next batches. The frequency threshold for the edges is based on the number of graph instances. Thus, this approach preserves the previous history information. After the singleton frequent edges are calculated, the combination is formed based on the neighbouring information. DynamicFreqSubgraph algorithm computes AND operation on the edges and calculates the final row sum. If the final row sum satisfies the minimum support threshold, then it is marked as frequent.

5.4.3.1 Illustration for dynamic sliding window technique

For instance, in the graph streams of Figure 5.1, Let batch size=2. Then, batch 1 constitutes graph streams G1, G2 of Fig. 5.1. For batch 1, the frequent singleton edges along with edge count are stored in map $\boldsymbol{M}$ for each edge satisfying 40% of batch size (*percentminsup*=0.4), i.e., $\lceil 0.4*2 \rceil = 1$. Thus, the initial map $\boldsymbol{M}$ for Fig. 5.1 with relative support 1 includes all the edges with count greater than or equal to 1 as shown in Table 5.5.

This map is incrementally maintained as shown in Table 5.6 for the next batch with relative support 2. When the batch 3 is encountered, the edge counts are incremented as shown in Table 5.7 with relative support of 4. Thus, the resultant frequent singleton edges by applying Algorithm 10 with the given minimum support of 4 are {(v1, v2), (v1, v4), (v1, v5), (v2, v5)}.

5.4.4 Enhancements to the proposed static and dynamic sliding window filtering techniques

We adopt the polynomial strategy [91] and modify it to compute the actual minimum support based on dataset information for graph stream data. We use this

Algorithm 10: DynamicFreqSubgraph

Input : Edges in the graph streams, *percentminsup*
Output: Set of Frequent subgraphs
`// n=Total number of graph streams in the entire sequence`
`// DSMatrix[m][n]= 2D matrix of m rows and n columns`
`// batchsize = number of graphs in a batch`
totalbatch= n / batchsize
batchpointer=0
k1=0
repeat
  Compute DSMatrix for each batch of graph streams
  Calculate edge_count for each batch
  minsup=*percentminsup**(batchpointer*batchsize+1)
  k1=k1+1
  if *(edge_count[i] ≥ minsup)* **then**
  Store < edge i, edge_count[i] > in a map M
  batchpointer= batchpointer + batchsize
until *k1 < totalbatches*;
relativesup=⌈(*percentminsup**n)⌉
for *i=0 to m* **do**
  if *(edge_count[i]≥ relativesup)* **then**
    Edge i is frequent
  else
    Edge i is not frequent

Join k recurrent connected edges with common edge based on neighbouring information to *k+1* recurrent connected edges by intersecting their bit vectors from DSMatrix, f_l for all $k \leq$m.
if *freq*(f_l)$\geq$*minsup* **then**
  f_l is frequent.
else
  f_l is infrequent.

Table (5.5) Frequent edges with count for Batch 1

edge	count
(v1,v2)	2
(v1, v4)	2
(v1,v5)	2
(v2,v5)	2
(v2, v3)	1

Table (5.6) Frequent edges with count for Batch 1 and Batch 2

edge	count
(v1,v2)	4
(v1, v4)	4
(v1,v5)	3
(v2,v5)	3
(v2, v3)	2
(v3, v4)	2

Table (5.7) Frequent edges with count for Batch 1, Batch 2 and Batch 3

edge	count
(v1,v2)	5
(v1,v4)	6
(v1,v5)	4
(v2,v5)	5

minimum support in the proposed static approach to compute frequent singleton edges. Besides, we propose partition based sequential and parallel static approach with actual minimum support. In addition, we observed that our earlier proposed dynamic approach may miss some of the frequent singleton edges for large graph streams with the iterative increase in relative threshold. To overcome this limitation, we modified the dynamic approach by using incremental relative support along with variable batch size.

5.4.4.1 Enhancement #1: Computing actual minimum support for proposed static approach

We observed that there is a need to compute actual minimum support while finding frequent singleton edges in real-time as the user may not have prior knowl-

edge about the characteristics of the graph database. Another observation is that there may be loss of significant and required frequent subgraphs because of lack of knowledge of the actual minimum support. Hence, we consider the minimum support given by the user as input to compute the actual minimum support based on distribution of frequent singleton edges in the graph stream database in the interval [*amin, bmax*]. *amin* denotes minimum frequency of the edge and *bmax* denotes the maximum frequency of the edge in the graph stream data. We adopt the approximate polynomial function of degree i [91] and compute the actual minimum support (*Actualminsup*) based on the user given minimum support (*minsup*) through the equation below:

$$Actualminsup^i = minsup * (bmax^i - amin^i) + amin^i \tag{5.1}$$

We then apply the proposed static approach with the actual minimum support to compute the resultant frequent subgraphs. For instance, consider the graph stream data in figure 5.1, with the user given minimum support, *minsup*=0.5, the minimum frequency *amin*=1/6 and maximum frequency *bmax*=5/6. The actual minimum support computed using equation (1) with linear strategy (i=1) is 0.49 and with cubic strategy (i=3) is 0.66. We use actual minimum support when the user is not an expert about the minimum support using which the significant frequent subgraphs can be retrieved.

5.4.4.2 Proposed partition based static approach

We modify our proposed static approach with single window by partitioning the data into windows of fixed size for faster execution. We then compute the actual minimum support for each window used to compute frequent singleton edges. We can run each partition in parallel to extract the frequent singleton edges. Algorithm 11 describes our proposed partition based static approach with actual minimum support.

Illustration of Proposed Partition based Static Approach For graph streams, with actual minimum support using linear strategy of 2.774∼3 (given user *minsup*=2), with partition size=2, the frequent singleton edges for the first

Algorithm 11: PartitionStaticFreqSubgraph

Input : Edges in graph streams, total number of graphs in the whole data, *minsup* = minimum support threshold, partition_no be number of partitions

Output: Set of frequent subgraphs

`// m= number of graph streams, n= number of edges in the entire graph streams sequence`

amin=$\frac{1}{m}$ `// Mat_A[m][n] = 2-D DSMatrix with m rows and n columns`

`// rowsum = count of 1s in a row of DSMatrix`

Create neighbouring list, N for the graph streams by finding common vertex for the stream of edges

for *each partition p ∈ [1, partition_no]* **do**

  for *each graph* g_j **do**

    if *edge i* $\in g_j$ **then**

      Mat_A[i][j]:=1

    else

      Mat_A[i][j]:=0

  Calculate *rowsum* for each row of Mat_A.

  Compute *bmax*, maximum frequency of any edge by considering maximum *rowsum*

  bmax=$\frac{bmax}{m}$

  Compute *Actualminsup* using (5.1)

  for *each edge i* **do**

    if *(rowsum(i)* $\geq$ *Actualminsup)* **then**

      Add edge i to set *F*

Join k recurrent connected edges from F with common edge using neighboring list N and to get $k+1$ recurrent connected edges , f_l for all $k \leq$m.

Compute freq(f_l) by intersecting the bit vectors of k recurrent edges from DSMatrix and adding all the resultant non-zero bits

if *freq(f_l)* $\geq$ *Actualminsup* **then**

  f_l is frequent

else

  f_l is infrequent

partition are {(v1, v2), (v1, v4), (v2, v5)}. The frequent singleton edges for the second partition are {(v1, v4)}. The resultant frequent subgraphs include { <(v1, v2), (v1, v4)>, <(v1, v4), (v2, v5)>, <(v1, v2), (v1, v4), (v2, v5)> }.

5.4.4.3 Enhancement #2: Dynamic sliding window filtering technique

In the proposed dynamic sliding window technique (described in Section 5.4.3), we observed that with increase in relative support, we may miss some of the significant frequent subgraphs. Hence, we modified the proposed dynamic approach by storing the batch number in addition to frequency for the edges in a map. For the first batch, we start with relative support and find the frequent singleton edges. The relative support is iteratively increased with next batch if the edge obtained in the current batch is frequent in the previous batch. If the edge is not frequent in the previous batch, but is frequent in the current batch, then we compare the frequency of the edge in the current batch with the relative support. For the next batch, the relative support is iteratively incremented. Thus, this approach preserves the previous history information. In addition, our proposed approach described in Algorithm 12 considers batches with variable sizes. After the frequent singleton edges are calculated, the combination is formed based on the neighbouring information. DynamicVarFreqSubgraph algorithm computes AND operation on the neighbouring frequent singleton edges and calculates the final row sum. If the final row sum satisfies the relative frequency threshold, then it is frequent.

Illustration of Dynamic Sliding Window Filtering Technique For instance, in the graph streams of Fig. 5.1, let batch size=3. Then, batch 1 constitutes graph streams G1, G2 and G3 of Fig. 5.1. For batch 1, the frequent singleton edges along with edge count are stored in map $\boldsymbol{M}$ for each edge with its relative frequency in the batch satisfying 40% of batch size. Thus, the initial map $\boldsymbol{M}$ for Fig. 5.1 with relative support 2 includes all the edges with count greater than or equal to 2 denoted by $\diamond$ symbol in Table 5.8.

Algorithm 12: DynamicVarFreqSubgraph

Input : Edges in graph streams, Let *rel* be percentage for computing relative support

Output: Set of Frequent subgraphs

```
// n=Total number of graph streams in the entire sequence
// DSMatrix[m][n]= 2D matrix of m rows and n columns
// batchsize = variable number of graphs in a batch
// Let b be the number of batches in the graph
```

totalbatch= n / batchsize

k1=1

Compute DSMatrix for each batch of graph streams

repeat

- Calculate edge_count for each batch
- batchpointer=0
- *minsup*=⌈*rel**(batchsize)⌉
- **if** *existsEdge(i)* **then**
  - ec=edge_count[i]+ Edge count of i in the current batch
  - *minsup*=⌈*rel**(k1-batchno[i]+1)*batchsize⌉
  - **if** *(ec ≥ minsup)* **then**
    - `// Edge i is frequent`
    - Update edge count of i to ec in map M
- **if** *(!existsEdge(i) & edge_count[i] ≥ minsup)* **then**
  - Store < edge i, batchno[i], edge_count[i] > in a map M
- batchpointer= batchpointer + batchsize
- k1=k1+1

until *k1* ≤ *b*;

Join *k* frequent connected edges with common edge to *k+1* frequent connected edges by intersecting their bit vectors from DSMatrix, f_l for all $k \leq$m.

minsup=⌈*rel**n⌉

if *freq*(f_l)≥*minsup* **then**

- f_l is frequent

else

- f_l is infrequent.

This map is incrementally maintained as shown in Table 5.9 for the next batch, with relative support 3, if the edge is frequent in the previous batch. If the edge is not frequent in the previous batch, then the relative support for such edge is 2. For instance, in Table 5.9, the edge (v3, v4) is not frequent in batch 1, but is frequent in the current batch as the relative support for that edge is set to 2. Thus, the resultant frequent singleton edges by applying Algorithm 12 with the

Table (5.8) Frequent edges with relative count for Batch 1

edge	Batch No.	count
⋄(v1, v2)	1	3
⋄(v1, v4)	1	3
⋄(v1, v5)	1	2
⋄(v2, v3)	1	2
⋄(v2,v5)	1	3
(v3, v4)	1	1

Table (5.9) Frequent edges with relative count for Batch 1 and Batch 2

edge	Batch No.	count
⋄(v1, v2)	1	5
⋄(v1, v4)	1	6
⋄(v1, v5)	1	4
⋄(v2, v3)	1	3
⋄(v2, v5)	1	5
⋄(v3, v4)	2	2

relative frequency threshold are {(v1, v2), (v1, v4), (v1, v5), (v2, v5), (v2, v3), (v3, v4)}. The resultant frequent subgraphs with relative support 3 are { <(v1, v2), (v1, v4)>, <(v1, v2), (v1, v5)>, <(v2, v5), (v1, v5)>,<(v2, v3), (v2, v5)>, < (v1, v2), (v1, v4), (v1, v5) > }.

5.4.5 Finding frequent subpaths from sequence of paths

For a directed graph, given a sequence of reachable paths [20], we can extract the frequent subpaths by using our proposed static and dynamic techniques. For this problem, algorithm 9 and algorithm 10 can be modified by considering each path as a graph stream input and the resultant output are frequent subpaths. In addition, while extracting frequent subpaths, we consider the sequence of neighbouring vertices that form a subpath. Algorithm 13 describes our proposed static single window technique to extract the frequent subpaths from sequence of paths.

Algorithm 11 and Algorithm 12 can also be similarly applied to find the frequent subpaths from sequence of paths for the given directed graph. For instance, for the directed graph in Fig 5.2, let us assume that the edges (v3, v5) and (v4, v5)

are frequent. Then, we cannot join the edges (v3, v5) and (v4, v5) as they do not form a subpath. Thus, the group of edges that do not form a subpath are not included.

Application areas of frequent subpaths extraction include analysis of traffic sub-routes based on the routes taken by vehicles (stored as paths database) to extract congested sections. In IP routing, we can analyse the routes taken by messages to extract the hot-spots. For instance, for graph of figure 5.2, Table 5.2 shows some of the paths extracted from the graph. The following subsections illustrate the extraction of frequent subpaths from paths data using the proposed static and dynamic techniques.

Algorithm 13: StaticFreqSubpath

Input : Sequence of paths and *minsup* = minimum support threshold
Output: Set of frequent subpaths
`// m= sequence of paths, n= number of edges in the entire paths data`
`// Mat_A[m][n] = 2-D DSMatrix with m rows and n columns`
`// rowsum = count of 1s in a row of DSMatrix`
Create neighbouring list, N for the paths by finding common vertex for the stream of edges
for *each path* g_j **do**
  if *edge* $i \in g_j$ **then**
    Mat_A[i][j]=1
  else
    Mat_A[i][j]=0
Calculate *rowsum* for each row of Mat_A.
if *(rowsum(i)* $\geq$*minsup)* **then**
  Edge i is frequent
else
  Edge i is not frequent
Join k frequent connected edges with common connecting vertices using neighboring list N to get $k+1$ frequent subpaths, f_l for all $k \leq$m.
Compute freq(f_l) by AND of the bit vectors of k recurrent edges for the path streams in the DSMatrix
if *freq(f_l)*$\geq$*minsup* **then**
  f_l is frequent.
else
  f_l is infrequent.

5.4.5.1 Illustration of static single window technique to find frequent subpaths

By considering each path as input for static single window technique, we retrieve the frequent subpaths for the set of paths. To extract frequent subpaths, we construct DSMatrix for the unique edges present in the paths set. Then, we extract frequent singleton edges from DSMatrix as described in Algorithm 13. From the frequent edges and neighbouring information, we extract the frequent subpaths. For instance, for the set of paths of Table 5.2, with *minsup*=3, the frequent singleton edges extracted using proposed static single window technique are {(v1, v2), (v2, v4), (v2, v5), (v3, v5), (v4, v5), (v5, v6), (v6, v7)} and the resultant frequent subpaths are { (v3, v5, v6), (v5, v6, v7)}

5.4.5.2 Illustration of dynamic sliding window filtering technique to find frequent subpaths

By considering each path as input for dynamic sliding window filtering technique, we retrieve the frequent subpaths for the set of paths. Initially, we construct the DSMatrix for the data of paths for each batch. Then, we extract the frequent singleton edges by using dynamic variable sliding window filtering technique described in Algorithm 12. For instance, let batch size=6 for paths data of Table 5.2. By using dynamic sliding window filtering technique, the resultant frequent singleton edges for the first batch with relative support=3 for path 1 to path 6 are {(v1, v2), (v5, v6)}. The resultant frequent singleton edges for the second batch for the remaining paths include {(v1, v2), (v5, v6), (v6, v7)}. Finally, the resultant frequent edges are {(v1, v2), (v5, v6), (v6, v7)} and the resultant frequent subpaths are {(v5, v6, v7)}.

5.4.6 Analysis of proposed static and dynamic approaches

The proposed static single window approach can be used for small datasets as this approach efficiently stores all the frequent subgraphs. Thus, the useful history information is retained. In addition, our proposed static approach with actual minimum support can retrieve the required number of frequent subgraphs/subpaths based on user minimum support. However, for large streams of graph data, we

can use the partition based static approach by running each partition of data in parallel to find frequent singleton edges. For large streams of data with variable batch size, the proposed dynamic sliding window approach efficiently finds the frequent subgraphs as relative support computation involves the number of graph streams. With the increase in the number of graph streams, the relative support is also proportionately increased thus storing the significant frequent subgraphs as well as minimizing the duplicate calculations. In addition, to extract frequent subpaths from large sequence of paths, our proposed dynamic sliding window filtering technique can be applied for large directed graphs. Table 5.4 shows different characteristics of proposed static and dynamic techniques compared to the conventional approach [31].

5.5 Experimental Evaluation

The direct 1-step algorithm (*Conventional approach*) of Alfredo Cuzzocrea et al. [31] and its modified versions, the proposed static single window approach (*Static*), proposed static approach with actual minimum support using linear strategy (*StaticLinear*) and cubic strategy (*StaticCubic*), proposed partition based static approach in sequential and parallel environments, dynamic sliding window technique (*DynFixed*) and dynamic variable sliding window filtering techniques (*DynVar*) are implemented.

The *minsup* is varied from 10% to 80% of number of graph streams/paths. For conventional approach, we set window size=5 and batch size=100. In dynamic approach and its variations, the batchsize is 100 for fixed batches and for variable batches, the batch size is randomly varied such that total number of batches are set to 10. In the experiments, the actual minimum support is computed based on the user given minimum support using equation 5.1 with i=1 for linear strategy and i=3 for cubic strategy. The relative support for batches are varied from 10% to 80% for the proposed dynamic sliding window filtering technique.

Experiment #1: Finding frequent subgraphs from graph stream data with minimum support and actual minimum support To extract frequent subgraphs from graph stream data, the proposed algorithms are exper-

imented on real dataset, Connect-4 [4]. We have considered each record of Connect-4 dataset as graph stream and constructed 1000 graph instances. The Connect-4 dataset can be used to find the most frequent moves chosen by the winner of the game. In each record, there are 1s, -1s and 0s for each position of 6×7 matrix of Connect-4 dataset. During pre-processing of the dataset, each grid position of the matrix is rendered as vertex. Thus, there are 42 vertices. The adjacent 1s and -1s in those grids are taken as edges. The two adjacent 1s or -1s represent an edge denoted using their respective grid positions.

Table (5.10) Number of frequent singleton edges (|FSE|) for graph stream data with varying *minsup* for proposed static approach compared to Conventional Approach

minsup	*|FSE| for Proposed Static approach*	*|FSE| for Conventional Approach*
0.1	121	104
0.2	101	72
0.3	87	47
0.4	71	23
0.5	59	0
0.6	45	0
0.7	34	0
0.8	23	0

Table 5.10 shows the number of frequent singleton edges with varying *minsup* for proposed static approach and conventional approach. From the table and its graph as shown in figure 5.3, we observe that with increase in *minsup*, the number of frequent singleton edges for conventional approach reduced to zero from 50% *minsup*, whereas our proposed static approach retained frequent singleton edges.

From Table 5.10 and Table 5.11 , we also observe that the actual minimum support computed using linear strategy ($Actualminsup_l$) is closer to the user given minimum support than that of cubic strategy ($Actualminsup_c$) for graph stream data. Fig. 5.4 shows the number of frequent singleton edges for graph stream data extracted using our proposed approaches, i.e., static approach (*Static*), static

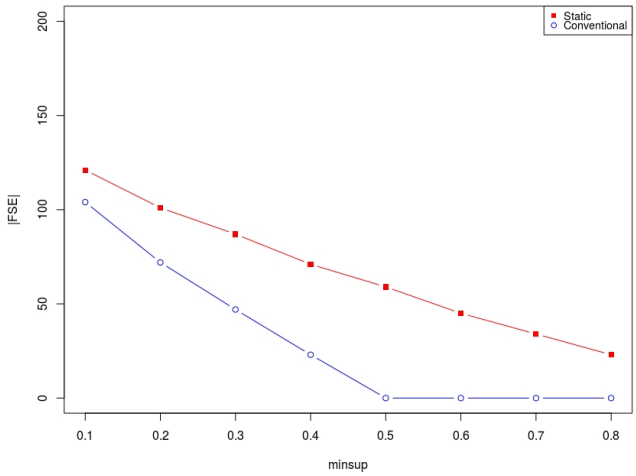


Figure (5.3) Number of frequent singleton edges (|FSE|) for proposed static approach compared to conventional approach for graph stream data

Table (5.11) Number of frequent singleton edges (|FSE|) for graph stream data with varying *minsup* for proposed static approach with actual minimum support using linear strategy with and cubic strategy

minsup	$Actualminsup_l$	$\|FSE\|\_linear$	$Actualminsup_c$	$\|FSE\|\_cubic$
0.1	0.099	121	0.450	64
0.2	0.198	101	0.576	49
0.3	0.296	87	0.661	40
0.4	0.390	73	0.726	30
0.5	0.490	61	0.783	25
0.6	0.590	46	0.832	20
0.7	0.690	37	0.875	12
0.8	0.789	24	0.915	10

approach with actual minimum support computed using linear strategy (*StaticLinear*) and cubic strategy (*StaticCubic*) compared to conventional approach. We observe that more number of frequent singleton edges are retained using our proposed static approach with linear strategy than the conventional approach with increase in minimum support.

Experiment #2: Finding frequent sub graphs through sequential and parallel approaches Table 5.12 shows the execution time of our proposed

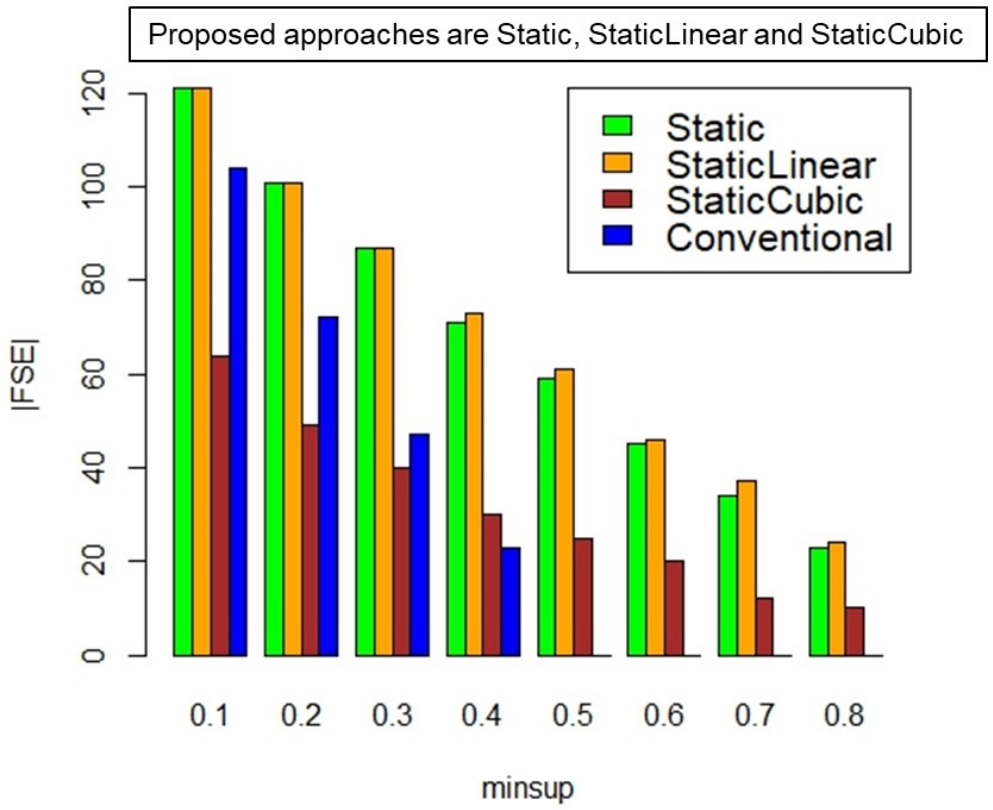


Figure (5.4) Number of frequent singleton edges (|FSE|) for proposed approaches with actual support using linear strategy and cubic strategy compared to conventional approach for graph stream data

partition based static approach (with number of partitions assumed to be 4) with actual minimum support computed using linear strategy in sequential and parallel environment. Table 5.13 shows the execution time of our proposed partition based static approach in sequential and parallel environment using cubic strategy, with the proposed partition based static approach in parallel environment executing faster. We implemented the parallel environment based on multi-threading, with each thread executing each partition while using the computed actual minimum support for every partition.

Table (5.12) Execution time (in *milliseconds*) to find frequent singleton edges for graph stream data using proposed static approach with actual minimum support through linear strategy in sequential and parallel environment

minsup	$Actualminsup_l$	*Execution time in sequential envt.*	*Execution time in parallel envt.*
0.1	0.102	0.417	0.262
0.2	0.199	0.454	0.260
0.3	0.296	0.388	0.245
0.4	0.394	0.414	0.271
0.5	0.49	0.430	0.270
0.6	0.59	0.381	0.215
0.7	0.687	0.355	0.209
0.8	0.785	0.394	0.203

Table (5.13) Execution time (in *milliseconds*) to find frequent singleton edges for graph stream data using proposed static approach with actual minimum support through cubic strategy in sequential and parallel environment

minsup	$Actualminsup_c$	*Execution time in sequential envt.*	*Execution time in parallel envt.*
0.1	0.455	0.391	0.187
0.2	0.573	0.369	0.167
0.3	0.656	0.391	0.253
0.4	0.722	0.377	0.218
0.5	0.078	0.360	0.198
0.6	0.827	0.334	0.177
0.7	0.870	0.363	0.187
0.8	0.909	0.34	0.148

Experiment #3: Finding frequent subgraphs through dynamic sliding window filtering approach Table 5.14 shows the number of frequent singleton edges extracted with relative support varying from 10% to 80% of batch size using our proposed dynamic sliding window technique (*DynFixed*) and dynamic variable sliding window filtering technique (*DynVar*). We observe that with increase in relative support, the number of frequent singleton edges extracted decreased. We also observe that we can consider the relative support from 20% upto 60%

to retrieve significant number of frequent singleton edges and thus the frequent subgraphs.

Table (5.14) Number of frequent singleton edges for graph stream data with varying % of relative support(*relsup*) for proposed dynamic approach with fixed batch size (*DynFixed*) and variable batch size (*DynVar*)

relsup	|FSE| for *DynFixed*	|FSE| for *DynVar*
0.1	127	124
0.2	108	108
0.3	90	89
0.4	76	75
0.5	64	64
0.6	51	49
0.7	39	39
0.8	25	25

Table (5.15) Number of frequent singleton edges (|FSE|) for paths data for proposed Static approach compared to Conventional approach

minsup	|FSE| for proposed Static Approach	|FSE| for Conventional Approach
0.1	31	7
0.2	4	0
0.3	0	0
0.4	0	0
0.5	0	0
0.6	0	0

Experiment #4: Finding frequent subpaths and their analysis with varying parameters For reachability path queries with constraints, the result of the query is sequence of paths. This database constitutes the reachable paths for reachability queries with constraints. These paths are stored in log file. We generate synthetic graph, i.e., E-R graph [53] with n=1000 and m=2000 edges. Next, we generate 800 reachability queries and retrieve 1000 possible paths satisfying the given constraints based on constrained BFS technique. We used the

same query generation process for constraint reachable paths addressed in [20]. Each path is considered as graph instance. We apply the conventional approach and the proposed static and dynamic techniques to find the frequent subpaths.

Table (5.16) Number of frequent singleton edges (|FSE|) for paths data using proposed static approach with actual minimum support using linear strategy and cubic strategy

$minsup$	$Actualminsup_l$	$\|FSE\|\_linear$	$Actualminsup_c$	$\|FSE\|\_cubic$
0.1	0.024	178	0.105	24
0.2	0.047	105	0.133	14
0.3	0.068	61	0.152	12
0.4	0.091	38	0.167	7
0.5	0.114	21	0.180	7
0.6	0.137	14	0.191	5
0.7	0.159	9	0.202	4
0.8	0.182	7	0.21	4

Table 5.15 shows the number of frequent singleton edges extracted from the sequence of paths using our proposed static approaches compared to conventional approach. We observe that as the paths data is sparse, the number of frequent singleton edges extracted for conventional approach is very less and is zero from 20% of minimum support. Our proposed static approach has zero frequent singleton edges with the user given minimum support from 30%.

From Table 5.16 and Table 5.15, we observe that the proposed static approach with actual minimum support using linear strategy and cubic strategy retained the significant frequent singleton edges than the conventional approach and proposed static approach with minimum support. Fig. 5.5 shows the number of frequent singleton edges extracted for paths data using our proposed approaches, i.e., static approach (*Static*), static approach with actual minimum support computed using linear strategy (*StaticLinear*) and cubic strategy (*StaticCubic*) compared to conventional approach. We observe that the number of frequent singleton edges are retained more using our proposed static approach using linear strategy with increase in minimum support for the generated paths data.

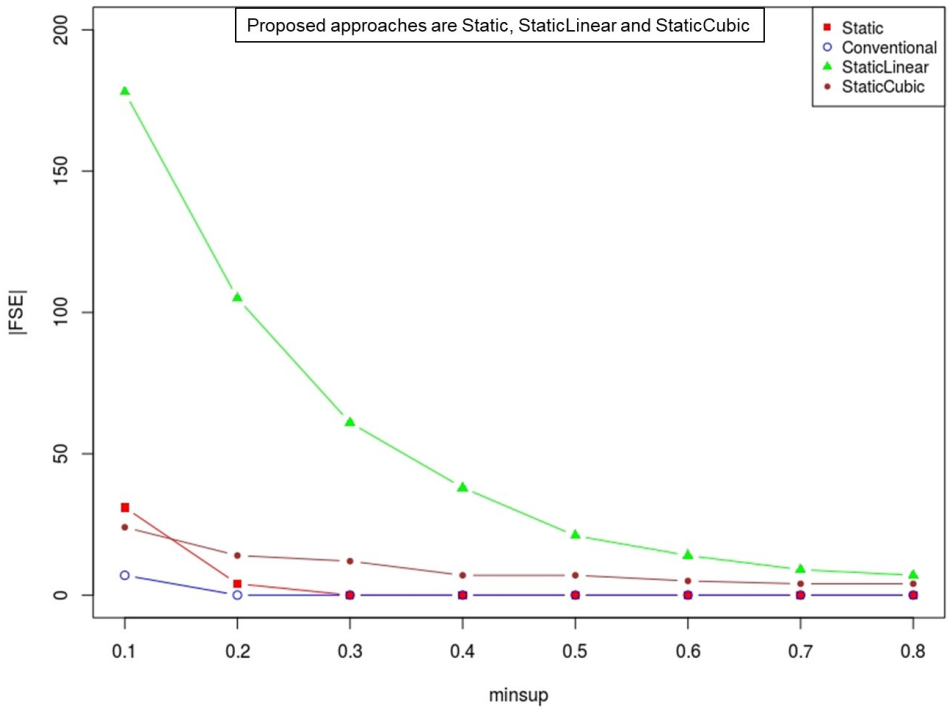


Figure (5.5) Number of frequent singleton edges (|FSE|) for proposed approaches with actual support using linear strategy and cubic strategy compared to conventional approach for sequence of paths

Table 5.17 and Table 5.18 shows the number of frequent singleton edges and frequent subpaths respectively that are extracted using our proposed dynamic approach with fixed batch size and variable batch size. Since the paths data is sparse, we observe that the relative support of 0.5% upto 1% retrieves significant number of frequent subpaths.

From the experiments and results, we observe that our proposed static approach with linear strategy extracted significant number of frequent singleton edges than that of cubic strategy. From these observations, we can conclude that our proposed techniques can efficiently extract frequent subpaths from sequence of paths.

Table (5.17) Number of frequent singleton edges (|FSE|) for paths data with varying relative support(*relsup*) for proposed dynamic approach with fixed batch size (*DynFixed*) and variable batch size (*DynVar*)

relsup	|FSE| for *DynFixed*	|FSE| for *DynVar*
0.005	599	529
0.01	488	414
0.02	329	267
0.03	240	210
0.04	193	167
0.05	159	139
0.06	133	119
0.07	108	100
0.08	98	84
0.09	88	75
0.1	73	100
0.2	14	12
0.3	4	2

Table (5.18) Number of frequent subpaths (|FSP|) for paths data with varying relative support (*relsup*) for proposed dynamic approach with fixed batch size (*DynFixed*) and variable batch size (*DynVar*)

relsup	|FSP| for *DynFixed*	|FSP| for *DynVar*
0.005	75	75
0.01	35	35
0.02	9	9
0.03	9	7
0.04	5	5
0.05	3	3
0.06	2	2

5.6 Conclusions

To discover the collections of frequent edges, we proposed two approaches the static single window approach and dynamic approach of sliding window. The proposed static approach finds frequent subgraphs by considering the entire graph streams as a single window and applying the given minimum support for them. In addition, we adopt polynomial strategy to compute the actual minimum support from the user given minimum support and apply it to our proposed static

approach that retained more number of frequent subgraphs. We also propose partition based static approach with actual minimum support that can be executed in parallel environment. In the dynamic approach, for each batch with variable size, we incrementally compute relative support and extract frequent edges. Finally, we join the frequent edges that share the common edge to extract the frequent subgraphs above the relative support computed for the entire graph stream data.

In addition, we also solve the problem of finding frequent subpaths from the sequence of paths by using our proposed techniques. From experiments, we observe that our proposed static approach with linear strategy retrieved significant number of frequent subpaths. The intention to propose these techniques is to solve similar type of queries of constraint reachability which will be discussed in Chapter 6.

Chapter 6

Query Processing Framework

A framework is defined as a generic combination of data and processes, where sub-components may be substituted. This chapter integrates our contributions and we propose a novel query processing framework to find paths for constrained reachability queries. Section 6.1 gives a brief introduction about the framework and its functionality. In section 6.2, we define the terminologies and discuss the problem statement by identifying important research questions. Section 6.3 describes the proposed query processing framework and its flow with examples. We perform experiments and analyze the usefulness of integrating every module of the proposed framework in section 6.4.

6.1 Introduction

One of the fundamental operations to manage graph data is to find the reachability from one vertex to another vertex in the graph. In real-time, the vertices and edges of a graph consist of attributes. These attributes give information about the type of vertices, type of relationship, and strength of the relationship between vertices. These real-time constraints motivate us to find reachability between the given vertices with vertex constraints and edge constraints.

To find the reachability for heterogeneous types of queries, the procesing structure is desired to fulfil the requirements of the user. Hence, query processing plays a vital role. The query processing would mean the entire process (module) or activity which involves query optimization, evaluation of query and extraction of

resultant information from respective components. In this chapter, the query processing starts when constraint reachability query is routed to a specific module. We studied different constrained reachability techniques described in Chapter 2. Our objective in this chapter is to propose a new query processing framework that can find paths for different variants of constrained reachability queries efficiently. Thus, we can apply our proposed techniques described in chapter 3 and chapter 4 for solving constraint reachability queries.

A framework [90] allows flexibility in choosing the most appropriate or available sub-modules as long as they perform the same specified functions. Integrated framework is used to place all the components required to implement systems and applications. We observe that collation of knowledge from different components or modules is a complex task. This chapter integrates our contributions and proposes a novel query processing framework to find paths for constrained reachability queries.

Our proposed query processing framework can find paths for variants of constrained reachability queries. Using the proposed query processing framework, we can store the resultant paths for queries in the Query Path Log (QPL) and extract frequent subpaths from them. These frequent subpaths can be used to handle similar queries. The results of the queries in QPL are used to solve the same queries.

One of the applications of the query processing framework is in social networks. Social networks may include information in the form of multiple vertex attributes values and edge attributes values. The constraints can be on the values of vertex attributes and edge attributes. In real-time, large number of queries can be invoked by many users to extract useful and unknown information from these social networks. These queries can also be repeated or similar to the previous queries. Our proposed framework can efficiently solve such queries. Our main contributions in this chapter are described as follows:

- We proposed Query Path Log and updation to handle same queries.
- We proposed a novel query processing framework to efficiently find paths for new queries, same queries and similar queries by integrating the proposed techniques for constrained reachability queries in the framework.

- We evaluated the usefulness and efficiency of our proposed query processing framework on real and synthetic datasets.

6.2 Problem Description

6.2.1 Problem statement

Consider Q={q_1, q_2, ..., q_n} are the different types of queries with respect to constraint reachability of a directed graph, for each $q_i \in Q$, owned by different modules M= {m_1, m_2, ..., m_r } where $m_i \neq \phi$, we find paths for the constrained reachability queries efficiently.

Given a source vertex, destination vertex, and a set of vertex constraints or edge constraints for the directed graph, the problem of finding paths for constrained reachability queries is to find the path between the given source vertex and destination vertex satisfying the given constraints.

It includes finding paths for MCR queries for attributed graphs and bounded LCR queries in case of weighted directed graphs. Besides, we identify techniques to handle the same queries and similar queries. We also address the usage of QPL and updating the paths and frequent subpaths in the log to handle such constraint reachability queries.

By analyzing the constraint reachability queries, we answer the following questions:

- Which techniques are used to compute reachable paths for constraint reachability queries based on constraints?
- How can we extract frequent subpaths from resultant paths?
- How can we update QPL?
- How can we handle new queries, same queries and similar queries?

For instance, let us consider a new MCR query q1(‘a’, ‘j’, ‘I:H’, ‘xml’), for the attributed graph of Figure 6.1. The resultant MCR path for the query q1 is {‘a’, ‘c’, ‘j’}. Now let us consider the bounded LCR query q2(‘P1’, ‘P7’, ‘SH’,

200) for the graph of the figure 6.3. The resultant bounded LCR path is {'P1', 'P6', 'P7'}. Consider bounded LCR query q3('P1', 'P6', 'SH', 200). Since q3 is a similar query to that of q1, the resultant stored path is {'P1', 'P6'}. Let us consider another bounded LCR query q4('P1', 'P7', 'SH', 200). Since q4 is the same query as that of q2, the resultant stored path is {'P1', 'P6', 'P7'}.

6.2.2 Definitions

Label-Constraint Reachability: *Given two vertices, 'vs' and 'vd' in the edge labeled directed graph G, and a label set A, where 'vs', 'vd'* $\in V$ *and* $A \subseteq T_l$*, if there is a path p from vertex 'vs' to 'vd' whose path label L(p) is a subset of A, i.e.,* $L(p) \subseteq A$*, then we say 'vs' can reach 'vd' with label-constraint A. We also refer to path p as an A-path from 'vs' to 'vd'.*[45]

LCR Query: Given two vertices '*vs*' and '*vd*', and a label set A, the label-constraint reachability (LCR) query asks if there exists an A-path from '*vs*' to '*vd*'.[45]

Path: "*Given a graph G(V, E), a path p of length k from a vertex u to u′ is a sequence* $(v_0, v_1, \ldots, v_k)$ *of vertices such that* $v_i \in V$*,* $v_0 = u$ *and* $v_k = u'$ *and*

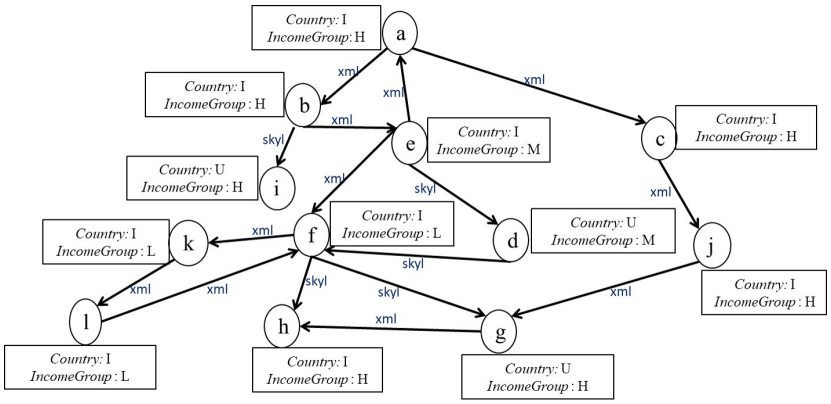


Figure (6.1) An instance of attributed graph for a toy email network

(v_{i-1}, v_i) $\in$ E for i = 1, 2, . . . , k." [41]

Subpath: *"A path Q in G is said to be a subpath of P if Q = (w_0, w_1, . . ., $w_{k'}$), where (w_0, w_1, . . . ,$w_{k'}$) is a contiguous sub-sequence of path P(v_0, v_1, . . . , v_k), i.e., if, for some i such that 0 $\leq$ i $\leq$ i+k' $\leq$ k, we have $w_0 = v_i$, $w_1 = v_{i+1}$, . . . , $w_{k'} = v_{i+k'}$."* [41]

Bounded Label Constrained Reachable Paths: Given two vertices '*vs*' and '*vd*', the label set $A \subseteq T_l$ and bound for the path-weight $\delta \in \mathbb{R}^+$ in an edge-labeled weighted directed graph G', if there is an A-path lp_i from *'vs'* to *'vd'* such that the path weight C(lp_i) $\leq \delta$, then we say *'vs'* can reach *'vd'* with label-constraint A and the path weight bound δ. In other words, it can also be referred as follows: Given two vertices *'vs'* and *'vd'*, a label set A and bound δ, the bounded label constrained reachable paths are the A-paths lp_i, between *'vs'* and *'vd'* that satisfy the bounded path weight constraint C(lp_i) $\leq \delta$.
We referred and termed the Bounded Label Constrained Reachable Paths as BLCRP.

Multidimensional Constraint Reachability: Given an attributed graph G, a source vertex s, a destination vertex t, vertex constraint CV_a, and edge constraint CE_a, the multidimensional constraint reachability query on attributed graph verifies whether s can reach t under vertex and edge constraint CV_a, CE_a [88].

6.3 Proposed Query Processing Framework

In this section, we propose a novel query processing framework to find paths for constraint reachability queries. The siginificant contributions are as follows:

- We extract the paths information for bounded LCR queries based on our proposed query processing with landmark-based path indexing.

- We extract the paths information for MCR queries using the proposed hashing based heuristic search technique.
- We update these queries and paths information in our proposed Query Path Log (QPL). We use this QPL to particularly handle the same queries and similar queries. We propose a novel technique to extract frequent subpaths from the QPL and update the log with the frequent subpaths information.

6.3.1 Query Path Log (QPL)

We construct QPL from new constraint reachability queries by storing the source vertex, destination vertex, constraints, and resultant paths. For each new query, we update the log by appending query information and resultant paths. The QPL is retrieved when the new queries, same queries and similar queries appear. QPL decides the type of query and subsequently identifies the respective modules for further processing to obtain the resultant path. We update the log with new queries/similar queries and the resultant paths. We extract frequent subpaths from the log after reaching a certain number of paths to solve similar queries. We maintain separate log files for BLCRP queries and MCR queries.

6.3.1.1 Query description

New Queries We identify a query as New query (N), if the query is not found in the QPL. Based on the type of constraints, the new query is executed as shown in query processing framework of figure 6.2 to extract the resultant paths.

If the constraints are edge label and bound on path weights, we compute the resultant constraint reachable paths using implicit landmark path indexing and query processing [15] explained in Chapter 3. If the constraints are on vertex attribute and edge attribute values, then the proposed hashing based heuristic search technique explained in Chapter 4 is invoked. For instance, consider a new bounded LCR query q1 (‘P1’, ‘P4’, ‘T’, 110). We invoke our proposed BLCRP technique and retrieve the resultant path of {‘P1’, ‘P2’, ‘P3’, ‘P4’}. This query and path information is added to the QPL.

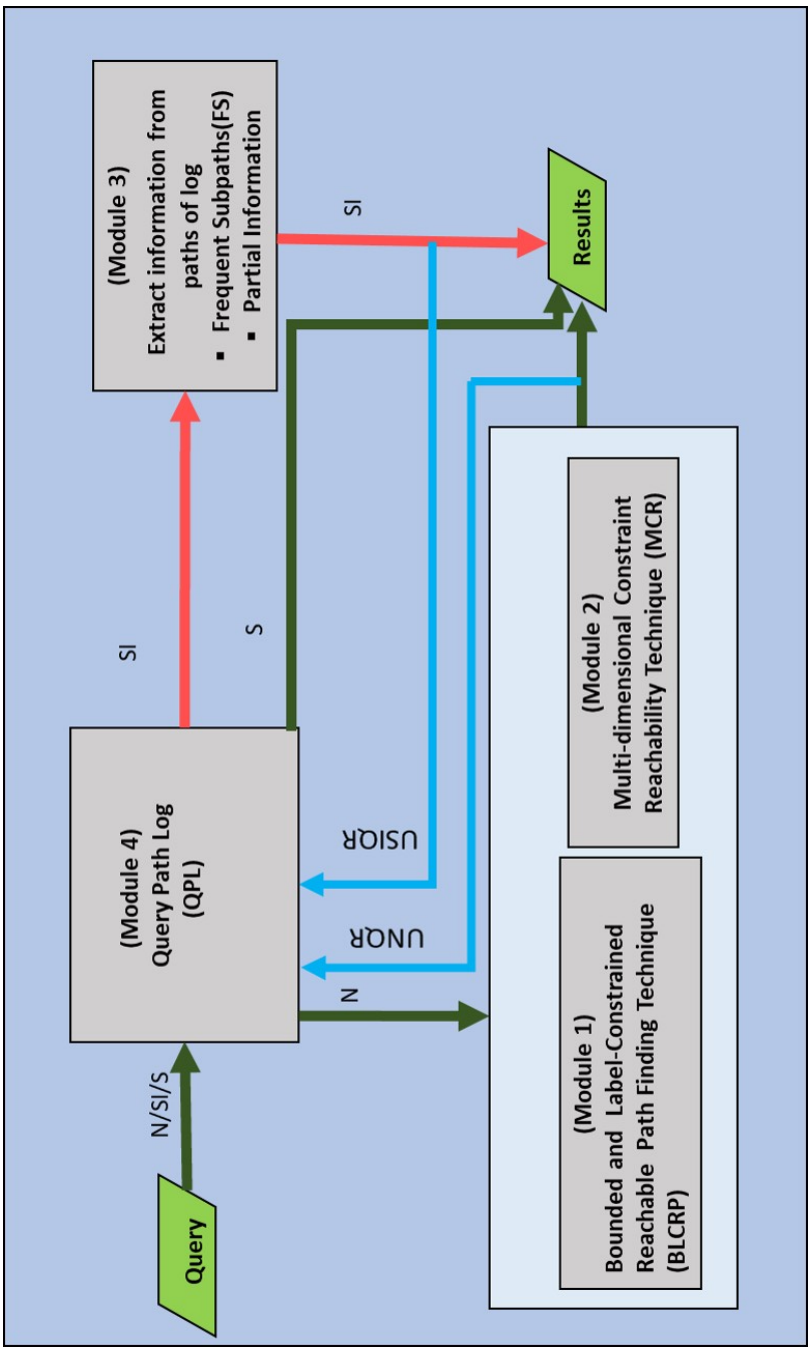


Figure (6.2) Query Processing Framework for Constrained Reachability Queries

Same Queries The same queries constitute the queries which are previously executed using proposed techniques. Thus, the query information such as given source vertex, destination vertex and constraints should match with at least one of the query information from the records of QPL.

If the queries are same (S), the results are directly extracted from the QPL as shown in figure 6.2 without executing the respective modules leading to faster query processing. For instance, let us consider a bounded LCR query q2 ('P1', 'P4', 'T', 110). Since q2 is same query as that of q1, the resultant path is retrieved from QPL (Table 6.1) is {'P1', 'P2', 'P3', 'P4'}.

Similar Queries We identify the similar (SI) queries as those queries whose source vertex or destination vertex are same, but constraints may differ. The constraints can be members or subset of the constraints present in QPL. We also consider similar queries whose source vertex and destination vertex can be different but constraints are same.

We solve similar queries by extracting query and path information from frequent subpaths of the QPL as shown in figure 6.2. For instance, let us consider a bounded LCR query q3 ('P3', 'P4', 'T', 110). We observe that in QPL file, q3 is similiar query to queries with ids QT1 and QT3 (Table 6.1) as q3 has same destination vertex as that of QT1 and QT3. Thus, the resultant path extracted for solving q3 is {'P3', 'P4'}.

True Queries The true queries are the queries that return atleast one path. For instance, these queries are denoted with query id QT1, QT2,... in QPL file as shown in Table 6.1. For example, the query q1 ('P1', 'P4', 'T', 110) is a true query with query id QT1 (Table 6.1) and resultant path {'P1', 'P2', 'P3', 'P4'} .

False Queries The false queries are the queries that have no paths. For instance, these queries are denoted with query id QF1, QF2,... in QPL as shown in Table 6.1. For example, the query q4 ('P9', 'P6', 'H', 150) is a false query with result *"No path"* as there doesn't exist any path between 'P9' and 'P6' satisfying the given constraints. This query is denoted as QF1 in QPL as shown in the Table 6.1.

6.3.1.2 Path index construction

We initialize the log file by inserting the query information and results whenever we invoke a new query. The query information includes source vertex, destination vertex, and constraints. The constraints can be vertex constraints or edge constraints, or both. For instance, Table 6.1 shows the QPL file for the constraint reachability queries of graph of Figure 6.1.

Table (6.1) Query Path Log

Query ID	Source	Destination	Constraint$_e$	Path
QT1	'P1'	'P4'	T,110	{'P1', 'P2', 'P3', 'P4'}
QF1	'P9'	'P6'	H,150	*"No Path"*
QT2	'P9'	'P4'	SH, 250	{'P9', 'P5', 'P4' }
QT3	'P2'	'P4'	T,100	{'P2', 'P3', 'P4'}
QT4	'P2'	'P10'	NHT,210	{'P2', 'P5', 'P6', 'P10'}
QT5	'P1'	'P10'	NHT,300	{'P1', 'P5', 'P6', 'P10'}
QF2	'P3'	'P6'	NH,100	*"No Path"*
...	..	..	..	...

6.3.1.3 Path index updation

Whenever a new query or similar query is executed, the QPL is updated as follows:

- When a **new** query is encountered, based on type of constraints, either BLCRP query or MCR query is identified. Then, module 1 of Chapter 3 or module 2 of Chapter 4 is invoked that consists of our proposed techniques to solve the query. Then, the resultant path information is updated along with query information in QPL.
- When a **similar** query is encountered, the frequent subpath information can be used to solve the similar query. If the similar query information does not match the extracted frequent subpaths, then that query is considered as new query.

Frequent Subpaths Extraction Once the log is updated with previous queries and their corresponding paths, the frequent subpaths are extracted from paths of QPL by using our proposed techniques described in chapter 5. The constraints of these subpaths are extracted and preserved to compare with the queries.

To extract constraints for the frequent subpaths of MCR queries, the constraints of corresponding paths in QPL are directly considered. This is because the constraints of every subpath of the path in QPL remains same as the constraints of the corresponding path. But, for bounded LCR paths, the constraints of the resultant frequent subpaths are to be verified before preserving, as the constraints on edge labels for the corresponding paths are the subset of the given set of edge labels. Hence, we extracted the constraints of subpaths for bounded LCR paths from the given graph instead of QPL file.

For instance, we observe that, in Table 6.1, the frequent subpaths extracted using our proposed technique are { ('P3', 'P4'), ('P5', 'P6', 'P10') }. The resultant constraints for the frequent subpaths extracted from the graph of figure 6.3 are {'T', 100} and {'HS', 150 } respectively.

6.3.2 Integrated framework

We integrate our contributions by combining four modules and developing novel query processing framework as shown in Figure 6.2. Given source vertex 's', destination vertex 't' and constraints as a query, the resultant path from *'s'* to *'t'* satisfying the constraints is found using the following modules:

Module 1: In this module, we compute the Bounded Label Constrained Reachable Paths (BLCRP) using our proposed technique. It involves the new bounded label constrained reachability query (N) execution. It is solved using the proposed implicit landmark path indexing and query processing technique (described in chapter 3). For instance, let us consider a new BLCRP query q1 ('P1', 'P4', 'T', 110) for the graph of figure 6.3. The resultant path retrieved using this module is {'P1', 'P2', 'P3', 'P4'}.

Module 2: In this module, we compute paths for Multidimensional Constraint Reachability (MCR) queries. When new multidimensional constraint reachability query (N) is to be executed, we use our proposed hashing based heuristic search

technique (described in chapter 4). For instance, let us consider a new MCR query q7 ('a', 'j', 'I:H', 'xml') for the graph shown in figure 6.1. The resultant path retrieved using this module is {'a', 'c', 'j'}.

Module 3: In this module, we can extract the paths for similar queries (SI) from the partial information of paths in the QPL. The partial information extraction involves computing frequent subpaths from the paths in the log. The frequent subpaths can be found based on our proposed static technique with linear strategy (described in chapter 5). For example, from the paths in Table 6.1, the frequent subpaths extracted using our proposed technique are {('P3', 'P4'), ('P5', 'P6', 'P10')}.

Module 4: The structure of QPL is shown in Table 6.1. Each record of QPL constitutes the query and its resultant path information. In this module, we extract the paths for same queries (S) from the QPL. QPL also stores the Updated New Queries Results (UNQR) and Updated Similar Queries Results (USIQR) from the respective modules.

6.3.2.1 Types of queries

We demonstrate the types of constrained reachability queries through the following cases:

Case 1: The bounded LCR query for which at least one path exists. For instance, let us consider LCR query q1('P1', 'P4', 'T', 100) for the graph of the Figure 6.3. The resultant bounded LCR path is {'P1', 'P2, 'P3', 'P4'}.

Case 2: The bounded LCR query for which there doesn't exist any path satisfying the given label-set constraint. For instance, let us consider the query q2('P2', 'P10', 'HT', 170). The result for q2 is no path exists between 'P2' and 'P10'.

Case 3: The same bounded LCR query, for which at least one path exists. For instance, let us consider bounded LCR query q3('P1', 'P4', 'T', 100) for the graph of the Figure 6.3. q3 is same query as that of q1 and thus, its resultant LCR path is {'P1', 'P2', 'P3', 'P4'}.

Case 4: The multidimensional constraint reachability query with vertex constraints or edge constraints or both for which at least one path exists. Let us

consider a new constraint reachability query q4('a', 'j', 'I:H', 'xml'), for the graph of Figure 6.1. For the given query q4, the source vertex 'a' can reach the destination vertex 'j' via vertex 'c' while satisfying the given vertex constraints 'I:H' and the edge constraint 'xml' along the path. The resultant constraint reachable path is {'a', 'c', 'j'}.

Case 5: There can exist a multidimensional constraint reachability query for which there does not exist any path in the entire graph even without satisfying the given label set constraint. For instance, let us consider MCR query q5('k', 'e', 'I:L', 'xml'). The result for q5 is no path exists between 'k' and 'e'.

Case 6: The same multidimensional constraint reachability query for which at least one path exists. For example, let us consider the constraint reachability query q6('a', 'j', 'I:H', 'xml'). q6 is same query as that of q4 and thus, its resultant path is {'a', 'c', 'j'}.

Case 7: There can exist a reachability query for which there exists a path p whose edge labels along the path are not the subset of the given label set. For instance, let us consider bounded LCR query q7('P2', 'P10', 'NT', 150). There exists a path {'P2', 'P5', 'P6', 'P10'} with the path label 'NHT' which is not subset of the given edge-label constraint 'NT'.

We work on finding paths for the reachability queries with constraints for the cases 1-5. Cases 1-3 are queries related to bounded label constrained reachable

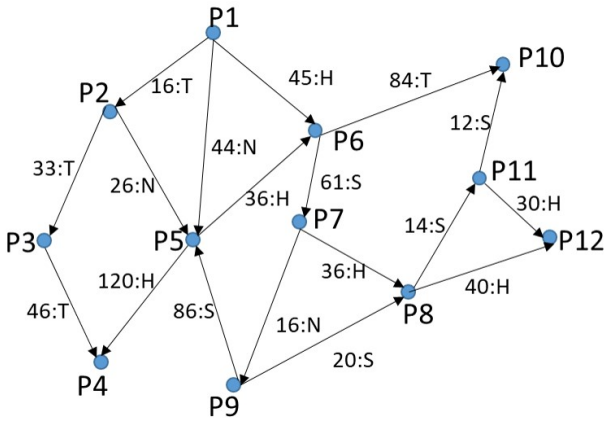


Figure (6.3) Road network

paths. Cases 4-6 are queries related to multidimensional constraint reachable paths. Case 7 is a why-not reachability query which is beyond the scope of the thesis and can be considered for future research directions. Currently, Case 7 type of queries are considered as new queries in our proposed framework.

6.3.3 Flow and functionality of modules

Algorithm **QPModules** (Query Processing Modules) describes the flow of framework and functionality of modules. For each constrained reachability query q_i of the given graph G, the resultant path information rp_i is retrieved. First, the QPL of module 4 is invoked for verifying the query q_i in which the given source and destination vertices and constraints are checked for matching. If match exists, the resultant path information is retrieved.

Algorithm 14: QPModules

Input : Constrained Reachability Queries Q={q_1, q_2, ...q_i}, graph G, *minsup* = minimum support threshold
Output: Path Information rp_i

```
// QPL storing query information and its path information
for each query q_i of graph G do
    if query q_i ∈ QPL then
        return rp_i from QPL of Module 4
    else
        if isSimilarQuery(q_i) then
            return rp_i from FS with the given minsup of Module 3
            Update QPL with q_i and rp_i in Module 4
        else
            if isBLCRPQuery(q_i) then
                return rp_i from BLCRP technique of Module 1
                Update QPL with q_i and rp_i in Module 4
            if isMCRQuery(q_i) then
                return rp_i from MCR technique of Module 2
                Update QPL with q_i and rp_i in Module 4
```

If the match is not found in QPL, module 3 is invoked. In module 3, the similar queries are solved by extracting frequent subpaths from paths of the QPL

in which the given source and destination vertices and constraints are checked for matching and resultant paths are retrieved. Here, the frequent subpaths are extracted from paths using our proposed techniques described in Chapter 5.

If the query is a new query and bounded LCR query, then module 1 is invoked in which the proposed implicit landmark path indexing and query processing technique (described in Chapter 3) is executed. If the query is a new query and multidimensional constraint reachability query, then module 2 is invoked in which proposed hashing based heuristic search technique (described in Chapter 4) is executed. After the query is processed, module 4 updates the new query information or similar query information and their respective resultant paths in the QPL as shown in figure 6.2.

6.4 Experimental Evaluation

During experimental evaluation of our proposed framework, murmur hash function [2] is used for hashing. We constructed the supergraph by using the existing ANCA clustering [37]. We assumed the number of super vertices to 15 and constructed the supergraph for all the datasets. While constructing landmark path index, we considered k=$\lceil sqrt(n) \rceil$ and b=20 based on the parameter values set in [79] for the proposed approach.

6.4.1 Datasets description

Table 6.2 summarizes the real and synthetic datasets used for experiments. We generated synthetic graphs from SNAP [53]. We assigned randomly vertex attribute values for the vertices and edge attribute values for the edges. Table 6.3 states the synthetic vertex attributes that are assigned randomly to the datasets.

6.4.1.1 Robots

Robots is a real trust network [1] with edge labels that denote the level of trust interaction between the users. We pre-process the dataset by assigning unique

Table (6.2) Datasets Overview

Real Graph	**|V|**	**|E|**
Robots [1]	1724	3596
Synthetic Graph	**|V|**	**|E|**
Erdos-Renyi [53]	1000	2000

identifier to the vertices, thus, resulting in 1724 vertices and 3596 edges. Each vertex has synthetic attributes whose values are randomly assigned (Table 6.3). Each edge has *Trustlevel* as the real attribute whose value is derived from the data set. The trust level can be Master (M), Apprentice (A), Journeyer (J) or Observer (O).

6.4.1.2 Erdos-Renyi Graph

Erdos-Renyi (E-R) graphs are the synthetic graphs that follow power law distribution [13]. These graphs have their degree near uniformly distributed. We generate E-R graph using SNAP [53] with number of vertices set to 1000 and maximum degree for each vertex set to 2. Besides, we assign two attributes (as described in Table 6.3) for each vertex with randomly assigned values within the domain. For BLCRP queries, we randomly assigned edge weights in the range [10, 120]. In addition, we assigned one of 8 unique edge labels for every edge.

Table (6.3) Vertex attributes and edge attributes

Vertex Attribute	**Domain Size, Distribution**
Country	5, *uniform*
Region	3, *uniform*

6.4.2 Query generation

We generated true BLCRP queries through BFS based query generation process [79] with same queries for the datasets. Besides, we also generated true MCR queries based on constrained BFS for both the real and synthetic datasets.

6.4.3 Experiments and result analysis

We performed extensive experiments to evaluate our proposed query processing framework on real and synthetic datasets. We compared the average execution time of our proposed techniques with and without QPL for BLCRP queries and MCR queries. Besides, we extracted frequent subpaths and updated it in the QPL.

We use our proposed Implicit Path LandMark indexing and query processing technique (*IPLM*) [15] described in Chapter 3 to solve BLCRP queries. We denote *IPLM_QPL* as the proposed technique that uses landmark path indexing and query processing for new queries and the resultant paths are stored in the QPL. If the query is same query, the resultant paths are retrieved from QPL. We denote *IPLM_QPLFS* as our proposed technique, in which we extract frequent subpaths and update with their constraints information into QPL. We extracted the frequent subpaths from the resultant paths by using our proposed static single window technique with linear strategy [16]. If the query is same query or similar query, we can invoke QPL with frequent subpaths to extract the resultant paths.

Similarly, for MCR queries, we evaluate the query processing framework by finding average execution time using our proposed Extended Heuristic search technique with Matrix Factorization technique (*EHMF*) described in Chapter 4. We denote *EHMF_QPL* to denote the usage of proposed technique and QPL for new queries and same queries. We denote *EHMF_QPLFS* to denote the updation of QPL with frequent subpaths information.

Table (6.4) Average query execution time in *milliseconds(ms)* of proposed techniques on BLCRP queries for Robots dataset

Technique	Average Execution Time(ms)
IPLM	0.095
IPLM_QPL	0.00775
IPLM_QPLFS	0.085

6.4.3.1 Experiment # 1: Testing on the real and synthetic datasets with fixed number of queries

We generated 100 BLCRP true queries and 100 MCR true queries randomly for the real and synthetic datasets with atleast 20% of same queries. Table 6.4 shows the average execution time for BLCRP queries of Robots dataset. From the table, we observe that our proposed technique with QPL, i.e., IPLM_QPL has lesser execution time than the IPLM technique.

Table (6.5) Average query execution time of proposed techniques on BLCRP queries for E-R graphs

Technique	Average Execution Time(ms)
IPLM	0.005
IPLM_QPL	0.000211
IPLM_QPLFS	0.00023

Table 6.7 shows the average query execution time of MCR queries with only vertex constraints for Robots dataset. We observe that our proposed technique with QPL and frequent subpaths, i.e. EHMF_QPLFS has faster execution time than the EHMF technique. Furthermore, our proposed technique with only QPL, i.e. EHMF_QPL has relatively lesser average execution time than EHMF, revealing the usefulness of QPL. Table 6.5 and Table 6.6 show the average query execution time of BLCRP queries and MCR queries respectively for E-R graphs. For MCR queries, we observe that EHMF_QPLFS has lesser execution time than only EHMF and EHMF_QPL. Besides, for BLCRP queries, IPLM_QPL and IPLM_QPLFS executed faster than only IPLM technique. These results reveal the usefulness of our proposed query processing framework.

Table (6.6) Average query execution time of proposed techniques on MCR queries for E-R graphs

Technique	Average Execution Time(ms)
EHMF	14.322
EHMF_QPL	3.681
EHMF_QPLFS	0.148

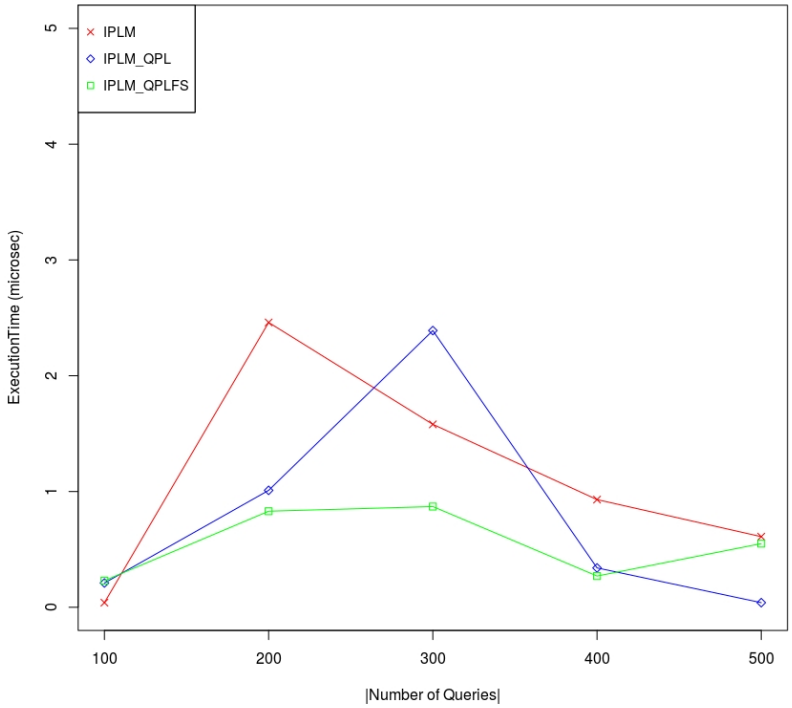


Figure (6.4) Average execution time with varying number of BLCRP queries for E-R graphs

Table (6.7) Average query execution time of proposed techniques on MCR queries for Robots dataset

Technique	Average Query Execution Time (ms)
EHMF	359.39
EHMF_QPL	206.04
EHMF_QPLFS	0.29025

6.4.3.2 Experiment # 2: Varying number of queries

In this experiment, we generated true BLCRP queries varying from 100 to 500 for E-R graphs and computed average query execution time of our proposed techniques. Figure 6.4 shows the average query execution time with varying number of queries. We observe by using our proposed technique with QPL, the average query execution time is lesser thus revealing the efficiency of our proposed query processing framework.

6.5 Conclusions

We proposed a novel query processing framework to find paths for constrained reachability queries. We proposed an efficient QPL based technique to solve the same queries or similar queries efficiently. We identified the use of extracting frequent subpaths for handling similar queries. Using our proposed query processing framework, we integrated our contributions to get the resultant constrained reachable paths. We evaluated the integration of modules of our proposed query processing framework on real and synthetic datasets. We observed that by using our proposed technique with QPL, the average query execution time is lesser thus revealing the efficiency of our proposed query processing framework.

Chapter 7

Conclusions and Future Scope

7.1 Conclusion

In this thesis, we focused on finding paths for constrained reachability queries through efficient techniques. The constraints include membership based edge-label constraints, bound on path weight and multidimensional vertex constraints and edge constraints.

In Chapter 3 of the thesis, we introduced a novel problem of finding bounded paths for label constrained reachable paths. Given source vertex, destination vertex, edge label constraint and bound on path weight, the bounded label constrained reachable query finds the path between the vertices satisfying the constraints. We proposed an efficient landmark path indexing and query processing technique to solve the bounded label constrained reachable paths in edge labeled weighted directed graphs. In our proposed indexing technique, we identified different combinations of path labels and path weights for which minimality of label sets and Dijkstra's relaxation property need to be considered. We evaluated our proposed technique on real and synthetic graphs. We also evaluated the accuracy of our proposed techniques using measures such as precision, recall and performed statistical analysis.

We investigated the problem of finding the existence of paths for MCR queries in Chapter 4 of the thesis. We observed that indexing multiple vertex attribute combinations or edge attribute combinations can be efficiently performed through hashing. We observed and identified that matrix factorization based clustering

that considers both graph topology and attributes is an efficient clustering technique that can be used to solve multidimensional constrained reachability queries. We proposed an extended heuristic search by using the clustering and evaluated our proposed techniques on real and synthetic benchmark datasets.

We also discussed the problem of finding frequent subgraphs from graph streams in Chapter 5 of the thesis. We observed that the state-of-the-art techniques in the literature do not consider historical information which may lead to loss of significant frequent subgraphs. We proposed static single window technique and also applied actual threshold to retrieve the frequent subgraphs. Besides, we proposed dynamic window filtering techniques with relative support to incrementally find frequent edges and thus frequent subgraphs. One of the significant contribution of the thesis is that we related our proposed static and dynamic window filtering techniques to find frequent subpaths from the sequence of paths.

In chapter 6 of the thesis, we proposed a novel query processing framework integrating our contributions to find paths for constrained reachability queries. In the proposed framework, we solved new queries by applying our proposed techniques. The query information and resultant paths are stored in QPL. Besides, frequent subpaths are also extracted using our proposed techniques and updated in the log. We handled same queries and similar queries using the log to retrieve relevant constrained reachable paths. QPL plays an important role by handling all types of queries efficiently and triggering the respective modules of framework. We evaluated the efficiency of our proposed query processing framework on real and synthetic datasets.

7.2 Future Scope

In future, we can optimize the proposed technique for finding bounded constrained reachable paths by using path compression techniques to compress the intermediate paths. We can investigate further to find an efficient maintenance algorithm for adding and removing edges during landmark path indexing. The impact of such edges during the indexing can measured and evaluated in case of dynamic graphs [67]. The index construction time is still relatively high for the large

datasets. It can be reduced further only with trade-off in terms of memory or speed-up.

We can extend our research by applying our proposed techniques to find paths for all LCR queries in edge-labeled directed graphs. We can also extend our research to find the distance between the vertices satisfying the given edge label constraints. Besides, we can do finer analysis of the impact of graph topology on the performance of queries. We can investigate the implementation of landmark path indexing in multi-core or distributed environments [43]. Further, we can apply the landmark path index to practical query languages like OpenCypher queries and validate its use.

In addition, we can extend our work on MCR queries by finding an efficient index or computing extra hash values for membership based constraint reachability queries or reachability queries with set attribute constraints. We can also extend our research by constructing hash-based index for dynamic graphs.

Moreover, we can perform extensive study to perform attributed clustering for dynamic graphs. We can further optimize the super graph construction by constructing directed acyclic graph from the super graph and analyze its impact on solving multidimensional constrained reachability queries. We can also use our proposed technique to solve constrained reachability queries for single source vertex and multiple destination vertices. We can further perform research to identify techniques to find reachability between the given vertices with constraints specified on only some of the vertex attributes or edge attributes which is beyond the scope of the thesis.

In this thesis, we assumed that the vertex and edge attribute values are independent of each other and are discrete. In future, we can investigate the functional dependencies between attributes and their impact during clustering. Besides, we can extend our research to validate attribute values, to update the attribute values and remove outdated attributes while solving multidimensional constrained reachability queries.

We can further extend our proposed techniques to find frequent subgraphs by applying distributed techniques to partition the large dataset. We can then apply the proposed techniques to each partition and then group the resultant frequent subgraphs of each partition to get the final frequent subgraphs. We

can also extend by extracting frequent subgraphs from graph streams arriving from different sources. We observed from the experiments that we can compute the actual minimum support for the relative support of our proposed dynamic sliding window filtering technique which is part of our future work. In addition, we can extend by computing the actual minimum support to extract the frequent sub patterns based on the user minimum support using fuzzy membership based approach and analyse its efficiency. Besides, we can use our proposed techniques that extract frequent subpaths from the sequence of paths to compress the stored paths while indexing paths.

We can further optimize the proposed query processing framework by finding the partial information from stored paths and applying algebra to constraints of similar queries. Thus, we can identify and analyse the special cases for solving similar constrained reachability queries. Besides, we can investigate on the efficient storage and retrieval of query logs with queries and resultant paths. Furthermore, we can extend our proposed QPL and framework to find constrained reachable paths for undirected graphs. We can extend our proposed framework to identify the techniques that consider other real-time constraints like bound on edge weights. We can also apply the proposed framework on specific scenarios like social networks and transportation networks. Our proposed techniques cannot be applied on hierarchical graphs or layered graphs where each layer has unique labels. The study and implementation of constrained reachability techniques on such graphs is beyond the scope of the thesis.

References

[1] **Robots**. http://tinyurl.com/gnexfoy/, 2017. (58, 59, 90, 139, 140)

[2] **Murmur Hash**. https://sites.google.com/site/murmurhash/, 2011. (25, 79, 88, 139)

[3] **Tweet Statistics**, 2020. (4, 96)

[4] **Connect4**. https://www.kaggle.com/tbrewer/connect-4. (117)

[5] **Neo4j**. https://neo4j.com/. (33)

[6] **InfiniteGraph**. https://www.objectivity.com/products/infinitegraph/, 2020. (33)

[7] **Twitter FlockDB**. https://github.com/twitter/flockdb. (33)

[8] Charu C Aggarwal, Yao Li, Philip S Yu, and Ruoming Jin. **On dense pattern mining in graph streams**. *Proceedings of the VLDB Endowment*, **3**(1-2):975–984, 2010. (15, 28, 98)

[9] Charu C. Aggarwal and Haixun Wang. *Graph Data Management and Mining: A Survey of Algorithms and Applications*. Springer US, 2010. (12, 14, 41, 75)

[10] 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*, SIGMOD '89, pages 253–262, New York, NY, USA, 1989. ACM. (14, 41)

[11] Takuya Akiba, Yoichi Iwata, and Yuichi Yoshida. **Fast exact shortest-path distance queries on large networks by pruned landmark labeling**. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, pages 349–360. ACM, 2013. (17, 43)

[12] M. T. Al Amin, C. Aggarwal, S. Yao, T. Abdelzaher, and L. Kaplan. **Unveiling polarization in social networks: A matrix factorization approach**. In *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*, pages 1–9, 2017. (15, 27, 71, 77)

[13] Rka Albert and Albert lszl Barabsi. **Statistical mechanics of complex networks**. *Rev. Mod. Phys*, pages 47–94, 2002. (59, 90, 140)

[14] Thomas Timm Angela Bonifati, Wim Martens. **An Analytical Study of large SPARQL query logs**. *VLDB Journal*, **29**(2-3):655–679, 2020. (30)

[15] Bhargavi B. and Swarupa Rani K. **Implicit Landmark Path Indexing for Bounded Label Constrained Reachable Paths**. *International Journal of Recent Technology and Engineering (IJRTE)*, **8**(4):10, 2019. (131, 141)

[16] Bhargavi B. and K. Swarupa Rani. **Finding Frequent Subgraphs and Subpaths through Static and Dynamic Window Filtering Techniques**. *EAI Endorsed Transactions on Scalable Information Systems*, **7**(27), 4 2020. (141)

[17] Chris Barrett, Riko Jacob, and Madhav Marathe. **Formal-Language-Constrained Path Problems**. *SIAM J. Comput.*, **30**(3):809–837, May 2000. (16, 43)

[18] Hannah Bast, Daniel Delling, Andrew Goldberg, Matthias Müller-Hannemann, Thomas Pajor, Peter Sanders, Dorothea Wagner, and Renato F Werneck. **Route planning in transportation networks**. In *Algorithm engineering*, pages 19–80. Springer, 2016. (18)

[19] Scott Beamer, Krste Asanovic, and David Patterson. **Direction-optimizing breadth-first search**. In *SC'12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, pages 1–10. IEEE, 2012. (14)

[20] B. Bhargavi and K. Swarupa Rani. **Bounded Paths for LCR Queries in Labeled Weighted Directed Graphs**. In Mayank Singh, P. K. Gupta, Vipin Tyagi, Jan Flusser, and Tuncer Ören, editors, *Advances in Computing and Data Sciences*, pages 124–133, Singapore, 2018. Springer Singapore. (36, 58, 75, 113, 122)

[21] B Bhargavi and KP Supreethi. **Graph pattern mining: A survey of issues and approaches**. *International Journal of Information Technology*, **5**(2):401–407, 2012. (13)

[22] Albert Bifet, Geoff Holmes, Bernhard Pfahringer, and Ricard Gavaldà. **Mining Frequent Closed Graphs on Evolving Data Streams**. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '11, pages 591–599, New York, NY, USA, 2011. ACM. (96)

[23] Kyoungsoo Bok, Jaeyun Jeong, Dojin Choi, and Jaesoo Yoo. **Detecting Incremental Frequent Subgraph Patterns in IoT Environments**. *Sensors*, **18**(11):4020, Nov 2018. (28, 97, 102)

[24] Francesco Bonchi, Aristides Gionis, Francesco Gullo, and Antti Ukkonen. **Distance oracles in edge-labeled graphs**. In *Proceedings of the 17th International Conference on Extending Database Technology, EDBT 2014, Athens, Greece, March 24-28, 2014.*, pages 547–558, 2014. (21, 43)

[25] Li Chen, Amarnath Gupta, and M. Erdem Kurul. **Stack-based Algorithms for Pattern Matching on DAGs**. In *Proceedings of the 31st International Conference on Very Large Data Bases*, VLDB '05, pages 493–504. VLDB Endowment, 2005. (14, 41)

[26] Minghan Chen, Yu Gu, Yubin Bao, and Ge Yu. **Label and Distance-Constraint Reachability Queries in Uncertain Graphs**. In *Database Systems for Advanced Applications*, pages 188–202, Cham, 2014. Springer International Publishing. (15, 16, 23, 42, 43)

[27] Yangjun Chen and Yibin Chen. **An Efficient Algorithm for Answering Graph Reachability Queries**. In *Proceedings of the 2008 IEEE 24th International Conference on Data Engineering*, ICDE '08, pages 893–902, Washington, DC, USA, 2008. IEEE Computer Society. (14, 41)

[28] Edith Cohen, Eran Halperin, Haim Kaplan, and Uri Zwick. **Reachability and Distance Queries via 2-hop Labels**. In *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '02, pages 937–946, Philadelphia, PA, USA, 2002. Society for Industrial and Applied Mathematics. (14, 41, 75)

[29] Diane J. Cook and Lawrence B. Holder. *Mining Graph Data*. John Wiley & Sons, Inc., USA, 2006. (1, 33)

[30] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009. (22, 24, 89)

[31] Alfredo Cuzzocrea, Zhao Han, Fan Jiang, Carson K. Leung, and Hao Zhang. **Edge-based Mining of Frequent Subgraphs from Graph Streams**. *Procedia Computer Science*, **60**:573–582, 2015. Knowledge-Based and Intelligent Information & Engineering Systems 19th Annual Conference, KES-2015, Singapore, September 2015 Proceedings. (4, 12, 15, 28, 29, 96, 97, 101, 102, 104, 116)

[32] Daniel Delling, Andrew V Goldberg, Thomas Pajor, and Renato F Werneck. **Robust distance queries on massive networks**. In *European Symposium on Algorithms*, pages 321–333. Springer, 2014. (17, 43)

[33] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification (2nd Edition)*. Wiley-Interscience, USA, 2000. (26, 77, 81)

[34] Jack Edmonds. **Optimum branchings**. *Journal of Research of the national Bureau of Standards B*, **71**(4):233–240, 1967. (19)

[35] Amit Erez and Alexander Nadel. **Finding Bounded Path in Graph Using SMT for Automatic Clock Routing**. In *CAV (2)*, **9207** of *Lecture Notes in Computer Science*, pages 20–36. Springer, 2015. (18, 24)

[36] Petros Petrou et al. **ARGO: A Big Data Framework for Online Trajectory Prediction**. In *Proceedings of the Sixteenth International Symposium on Spatial and Temporal Databases*, SSTD '19, pages 194–197. ACM, 2019. (30)

[37] Issam Falih, Nistor Grozavu, Rushed Kanawati, and Younès Bennani. **ANCA : Attributed Network Clustering Algorithm**. In *Complex Networks & Their Applications VI*, pages 241–252, Cham, 2018. Springer International Publishing. (4, 9, 12, 15, 26, 71, 76, 77, 78, 81, 139)

[38] Wenfei Fan, Jianzhong Li, Shuai Ma, Nan Tang, and Yinghui Wu. **Adding regular expressions to graph reachability and pattern queries.** In *ICDE Proceedings*, pages 39–50. IEEE Computer Society, 2011. (13, 20, 42, 43)

[39] Michael J. Folk, Greg Riccardi, and Bill Zoellick. *File Structures: An Object-Oriented Approach with C++*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 3rd edition, 1997. (24, 44)

[40] Sudipto Guha, Andrew McGregor, and David Tench. **Vertex and Hyperedge Connectivity in Dynamic Graph Streams**. In *Proceedings of the 34th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, PODS '15, pages 241–247, New York, NY, USA, 2015. ACM. (28, 101)

[41] Sumanta Guha. **Finding frequent subpaths in a graph**. *International Journal of Data Mining & Knowledge Management Process*, **4**(5):35, 2014. (29, 98, 102, 130)

[42] Monika R. Henzinger, Prabhakar Raghavan, and Sridhar Rajagopalan. **Computing on Data Streams**. In James M. Abello and Jeffrey Scott Vitter, editors, *External Memory Algorithms*, pages 107–118. American Mathematical Society, kston, MA, USA, 1999. (27, 101)

[43] Li-Yung Ho, Jan-Jan Wu, and Pangfeng Liu. **Workload prediction and balance for distributed reachability processing for large-scale attribute graphs**. *Concurrency and Computation: Practice and Experience*, **30**(6):1–19, 2018. (23, 24, 43, 147)

[44] Zengfeng Huang and Pan Peng. **Dynamic Graph Stream Algorithms in O(n) Space**. *Algorithmica*, **81**(5):1965–1987, May 2019. (28, 96, 101)

[45] Ruoming Jin, Hui Hong, Haixun Wang, Ning Ruan, and Yang Xiang. **Computing Label-constraint Reachability in Graph Databases**. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, SIGMOD '10, pages 123–134, New York, NY, USA, 2010. ACM. (2, 3, 15, 19, 22, 33, 34, 37, 39, 42, 43, 75, 129)

[46] Ruoming Jin, Lin Liu, Bolin Ding, and Haixun Wang. **Distance-constraint reachability computation in uncertain graphs**. *Proceedings of the VLDB Endowment*, **4**(9):551–562, 2011. (22, 23)

[47] Ruoming Jin, Yang Xiang, Ning Ruan, and David Fuhry. **3-HOP: A High-compression Indexing Scheme for Reachability Query**. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data*, SIGMOD '09, pages 813–826, New York, NY, USA, 2009. ACM. (14, 41, 75)

[48] Ruoming Jin, Yang Xiang, Ning Ruan, and Haixun Wang. **Efficiently Answering Reachability Queries on Very Large Directed Graphs**. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, SIGMOD '08, pages 595–608, New York, NY, USA, 2008. ACM. (14, 41)

[49] Martin Junghanns, André Petermann, Martin Neumann, and Erhard Rahm. **Management and Analysis of Big Graph Data: Current Systems and Open Challenges**. *Handbook of Big Data Technologies*, page 457, 2017. (4)

[50] Rohit Kumar, Sanmeet Kaur, and K. Swarupa Rani (Supervisor). *Finding Frequent Subgraphs using Direct Vertical Mining from Graph Streams*. Master's thesis, (MCA), School of Computer and Information Sciences, University of Hyderabad, India, 2019. (103)

[51] Max R. Land and Linyuan Lu. **An Upper Bound on the Burning Number of Graphs**. In *WAW*, **10088** of *Lecture Notes in Computer Science*, pages 1–8, 2016. (44)

[52] Chang-Hung Lee, Cheng-Ru Lin, and Ming-Syan Chen. **Sliding window filtering: an efficient method for incremental mining on a time-variant database**. *Information Systems*, **30**(3):227–244, 2005. (15, 29, 99, 103)

[53] Jure Lescovec. **SNAP: A general purpose network analysis and graph mining library in C++**. <http://snap.stanford.edu/snap>., 2016. (58, 59, 90, 91, 121, 139, 140)

[54] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. **Graph Evolution: Densification and Shrinking Diameters**. *ACM Trans. Knowl. Discov. Data*, **1**(1), March 2007. (91)

[55] Carson K. Leung and Alfredo Cuzzocrea. **Frequent Subgraph Mining from Streams of Uncertain Data**. In *Proceedings of the Eighth International C* Conference on Computer Science & Software Engineering*, C3S2E '15, pages 18–27, New York, NY, USA, 2008. ACM. (5, 28)

[56] Carson Kai-Sang Leung, Fan Jiang, Adam G. M. Pazdor, and Aaron M. Peddle. **Parallel Social Network Mining for Interesting 'following' Patterns**. *Concurr. Comput.: Pract. Exper.*, **28**(15):3994–4012, October 2016. (96)

[57] Ankita Likhyani and Srikanta Bedathur. **Label Constrained Shortest Path Estimation**. In *Proceedings of the 22Nd ACM International Conference on Information & Knowledge Management*, CIKM '13, pages 1177–1180, New York, NY, USA, 2013. ACM. (16, 35, 43, 91)

[58] Bingqing Lyu, Lu Qin, Xuemin Lin, Lijun Chang, and Jeffrey Xu Yu. **Scalable supergraph search in large graph databases**. In *2016 IEEE 32nd International Conference on Data Engineering (ICDE)*, pages 157–168. IEEE, 2016. (12, 13)

[59] Shuai Ma, Jia Li, Chunming Hu, Xuelian Lin, and Jinpeng Huai. **Big graph search: challenges and techniques**. *Frontiers of Computer Science*, **10**(3):387–398, 2016. (4)

[60] Grzegorz Malewicz, Matthew H. Austern, Aart J. C Bik, James C. Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. **Pregel: A System for Large-Scale Graph Processing**. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, pages 135–146, NewYork, NY, USA, 2010. ACM. (33)

[61] Andrew McGregor. **Graph Stream Algorithms: A Survey**. *SIGMOD Rec.*, **43**(1):9–20, May 2014. (28, 96, 101)

[62] Pasquale De Meo, Katarzyna Musial-Gabrys, Domenico Rosaci, Giuseppe ML Sarnè, and Lora Aroyo. **Using centrality measures to predict helpfulness-based reputation in trust networks**. *ACM Transactions on Internet Technology (TOIT)*, **17**(1):1–20, 2017. (19)

[63] Christian von Mering, Martijn Huynen, Daniel Jaeggi, Steffen Schmidt, Peer Bork, and Berend Snel. **STRING: a database of predicted functional associations between proteins**. *Nucleic acids research*, **31**(1):258–261, 2003. (35)

[64] M. E. J. Newman. *Networks: an introduction*. Oxford University Press, 2010. (19)

[65] G. Qi, C. C. Aggarwal, and T. Huang. **Community Detection with Edge Content in Social Media Networks**. In *2012 IEEE 28th International Conference on Data Engineering*, pages 534–545, 2012. (15, 27, 71, 77)

[66] Miao Qiao, Hong Cheng, Lu Qin, Jeffrey Xu Yu, S Yu Philip, and Lijun Chang. **Computing weight constraint reachability in large networks**. *The VLDB journal*, **22**(3):275–294, 2013. (22, 42, 43)

[67] Yongrui Qin, Quan Z Sheng, Simon Parkinson, and Nickolas JG Falkner. **Edge Influence Computation in Dynamic Graphs**. In *International Conference on Database Systems for Advanced Applications*, pages 649–660. Springer, 2017. (19, 146)

[68] Sherif Sakr, Sameh Elnikety, and Yuxiong He. **G-SPARQL: A Hybrid Engine for Querying Large Attributed Graphs**. Technical Report MSR-TR-2011-138, Microsoft Research, December 2011. (25, 75)

[69] Stephan Schwartz, Leonardo Balestrieri, and Ralf Borndörfer. **On Finding Subpaths With High Demand**. In *Operations Research Proceedings 2017*, pages 355–360, Cham, 2018. Springer International Publishing. (29, 102)

[70] Yingxia Shao, Bin Cui, and Lin Ma. **Page: a partition aware engine for parallel graph computation**. *IEEE Transactions on Knowledge and Data Engineering*, **27**(2):518–530, 2014. (12)

[71] Henry H Su and P Eter A. L Achenbruch. **Paired t-test**. In *Wiley Encyclopedia of Clinical Trials*. John Wiley & Sons Inc., 2008. (64)

[72] Zhixiong Su, Jianxun Qi, and Hanying Wei. **Path problem simplification with desired bounded lengths in acyclic networks**. *Journal of Systems Science and Systems Engineering*, **24**(4):500–519, 2015. (17, 24)

[73] Jonathan M Sumrall. *Path Indexing for Efficient Path Query Processing in Graph Databases*. Master's thesis, Department of Mathematics and Computer Science, Eindhoven University of Technology, 2015. (19, 43)

[74] Jonathan M Sumrall, George HL Fletcher, Alexandra Poulovassilis, Johan Svensson, Magnus Vejlstrup, Chris Vest, and Jim Webber. **Investigations on Path Indexing for Graph Databases**. In *European Conference on Parallel Processing*, pages 532–544. Springer, 2017. (19, 43)

[75] Nan Tang, Qing Chen, and Prasenjit Mitra. **Graph Stream Summarization: From Big Bang to Big Crunch**. In *Proceedings of the 2016 International Conference on Management of Data*, SIGMOD '16, pages 1481–1496, New York, NY, USA, 2016. ACM. (28, 96, 101)

[76] Mikkel Thorup and Uri Zwick. **Approximate distance oracles**. *Journal of the ACM (JACM)*, **52**(1):1–24, 2005. (16)

[77] Robert Tibshirani, Guenther Walther, and Trevor Hastie. **Estimating the Number of Clusters in a Data set via the Gap Statistic**. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, **63**(2):411–423, 2001. (71, 81, 82, 88, 93)

[78] Silke Trissl and Ulf Leser. **Fast and practical indexing and querying of very large graphs**. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 845–856, 2007. (41)

[79] Lucien D. J. Valstar, George H. L. Fletcher, and Yuichi Yoshida. **Landmark Indexing for Evaluation of Label-Constrained Reachability Queries**. In *Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD Conference 2017, Chicago, IL, USA, May 14-19, 2017*, pages 345–358, 2017. (4, 9, 15, 20, 21, 24, 36, 40, 42, 43, 44, 45, 47, 49, 57, 60, 139, 140)

[80] Haixun Wang, Hao He2, Jun Yang, Philip S. Yu, Jeffrey Xu Yu, and Jeffrey Xu Yu. **Dual Labeling: Answering Graph Reachability Queries in Constant Time**. In *Proceedings of the 22Nd International Conference on Data Engineering*, ICDE '06, page 75, Washington, DC, USA, 2006. IEEE Computer Society. (14, 41, 75)

[81] Zhengkui Wang, Qi Fan, Huiju Wang, Kian Lee Tan, Divyakant Agrawal, and Amr El Abbadi. **Pagrol: Parallel Graph OLAP over large-scale attributed graphs**. In *Proceedings - International Conference on Data Engineering*, pages 496–507. IEEE Computer Society, 1 2014. (24, 68, 72, 75)

[82] Hao Wei, Jeffrey Xu Yu, Can Lu, and Ruoming Jin. **Reachability Querying: An Independent Permutation Labeling Approach**. *The VLDB Journal*, **27**(1):1–26, February 2018. (14, 41)

[83] Zhonggang Wu, Zhao Lu, and Shan-Yuan Ho. **Community Detection with Topological Structure and Attributes in Information Networks**. *ACM Trans. Intell. Syst. Technol.*, **8**(2):33:1–33:17, November 2016. (4, 26, 71, 76)

[84] Zhiqiang Xu, Yiping Ke, Yi Wang, Hong Cheng, and James Cheng. **A Model-Based Approach to Attributed Graph Clustering**. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, SIGMOD 12, pages 505–516, New York, NY, USA, 2012. Association for Computing Machinery. (4, 26, 71, 76)

[85] Jaewon Yang and Jure Leskovec. **Overlapping Community Detection at Scale: A Nonnegative Matrix Factorization Approach**. In *Proceedings of the Sixth ACM International Conference on Web Search and Data Mining*, WSDM 13, pages 587–596, New York, NY, USA, 2013. Association for Computing Machinery. (15, 27, 71, 77)

[86] Frank Yates and P Michael Grundy. **Selection without replacement from within strata with probability proportional to size**. *Journal of the Royal Statistical Society: Series B (Methodological)*, **15**(2):253–261, 1953. (23)

[87] Cheng J. Yu J.X. *Graph Reachability Queries: A Survey*, **40**. Springer, Boston, MA, 2010. (14, 41)

[88] Duncan Yung and Shi-Kuo Chang. **Fast reachability query computation on big attributed graphs**. In *Big Data (Big Data), 2016 IEEE International Conference on*, pages 3370–3380. IEEE, 2016. (4, 9, 12, 15, 24, 25, 27, 42, 43, 44, 68, 70, 71, 72, 73, 75, 77, 88, 89, 91, 92, 93, 94, 130)

[89] Ka Wai (Duncan) Yung. *Query Processing on Attributed Graphs*. PhD thesis, University of PittsBurgh, 2017. (4, 77, 78)

[90] Zaharin Yusoff. **On Architectures, Frameworks, and Models in Thesis Writing for Computer Science**. *Journal of IT in Asia*, **6**(1):1–10, 2016. (30, 127)

[91] Shichao Zhang, Xindong Wu, Chengqi Zhang, and Jingli Lu. **Computing the minimum-support for mining frequent patterns**. *Knowledge and Information Systems*, **15**(2):233–257, 2008. (4, 15, 29, 99, 103, 106, 109)

[92] Yang Zhou, Hong Cheng, and Jeffrey Xu Yu. **Graph Clustering Based on Structural/Attribute Similarities**. *Proc. VLDB Endow.*, **2**(1):718–729, August 2009. (4, 20, 26, 71, 76)

[93] Lei Zou, Kun Xu, Jeffrey Xu Yu, Lei Chen, Yanghua Xiao, and Dongyan Zhao. **Efficient Processing of Label-constraint Reachability Queries in Large Graphs**. *Inf. Syst.*, **40**:47–66, March 2014. (3, 20, 42, 43)

A Study of Constrained Reachability Query Processing in Directed Graphs

by Bhargavi B

A Study of Constrained Reachability Query Processing in Directed Graphs

ORIGINALITY REPORT

41%	21%	23%	7%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

Overall Similarity Index = 41 - (18+16+3) = 04% (FOUR)

1	eudl.eu Internet Source https://eudl.eu/contents?q=finding+frequent+subgraphs (enclosed at the end of this thesis) This publication is by the student	18%
2	"Implicit Landmark Path Indexing for Bounded Label Constrained Reachable Paths", International Journal of Recent Technology and Engineering, 2019 Publication This publication is by the student	16%
3	B. Bhargavi, K. Swarupa Rani. "Chapter 13 Bounded Paths for LCR Queries in Labeled Weighted Directed Graphs", Springer Science and Business Media LLC, 2018 Publication This publication is by the student	3%
4	www.scribd.com Internet Source	<1%
5	d-scholarship.pitt.edu Internet Source	<1%
6	Submitted to National Institute of Technology, Silchar Student Paper	<1%

Dr. K. Swarupa Rani
Associate Professor
School of CIS
Prof. C.R. Rao Road,
Central University
Hyderabad (India)

Similarity Screening Done @ IGM Library
Sub ID: 1476629944
Date: 16.12.2020
Librarian / ~~DLIAL~~
IGM Library (UOH)

7	pure.tue.nl Internet Source	<1%
8	Duncan Yung, Shi-Kuo Chang. "Fast reachability query computation on big attributed graphs", 2016 IEEE International Conference on Big Data (Big Data), 2016 Publication	<1%
9	Lecture Notes in Computer Science, 2014. Publication	<1%
10	Hamilton Wilfried Yves Adoni, Tarik Nahhal, Moez Krichen, Brahim Aghezzaf, Abdeltif Elbyed. "A survey of current challenges in partitioning and processing of graph-structured data in parallel and distributed systems", Distributed and Parallel Databases, 2019 Publication	<1%
11	Zhixiong Su, Jianxun Qi, Hanying Wei. "Path problem simplification with desired bounded lengths in acyclic networks", Journal of Systems Science and Systems Engineering, 2015 Publication	<1%
12	Lecture Notes in Computer Science, 2015. Publication	<1%
13	www.springerprofessional.de Internet Source	<1%
	Miao Qiao, Hong Cheng, Lu Qin, Jeffrey Xu Yu,	

14	Philip S. Yu, Lijun Chang. "Computing weight constraint reachability in large networks", The VLDB Journal, 2012 Publication	<1%
15	Submitted to Carnegie Mellon University Student Paper	<1%
16	Jacek Czekaj. "Comparison of the Exact and Approximate Algorithms in the Random Shortest Path Problem", IFIP Advances in Information and Communication Technology, 2009 Publication	<1%
17	Zhonggang Wu, Zhao Lu, Shan-Yuan Ho. "Community Detection with Topological Structure and Attributes in Information Networks", ACM Transactions on Intelligent Systems and Technology, 2017 Publication	<1%
18	Carson K. Leung, Alfredo Cuzzocrea. "Frequent Subgraph Mining from Streams of Uncertain Data", Proceedings of the Eighth International C* Conference on Computer Science & Software Engineering - C3S2E '15, 2008 Publication	<1%
19	"Handbook of Big Data Technologies", Springer Science and Business Media LLC, 2017 Publication	<1%
	"Algorithms - ESA 2014", Springer Science and	

20	Business Media LLC, 2014 Publication	<1%
21	Luo Chen, Ye Wu, Zhinong Zhong, Wei Xiong, Ning Jing. "A hierarchical model for label constraint reachability computation", Neurocomputing, 2015 Publication	<1%
22	docplayer.net Internet Source	<1%
23	Lucien D.J. Valstar, George H.L. Fletcher, Yuichi Yoshida. "Landmark Indexing for Evaluation of Label-Constrained Reachability Queries", Proceedings of the 2017 ACM International Conference on Management of Data - SIGMOD '17, 2017 Publication	<1%
24	Zhiwei Zhang, Jeffrey Xu Yu, Lu Qin, Qing Zhu, Xiaofang Zhou. "I/O cost minimization", Proceedings of the 15th International Conference on Extending Database Technology - EDBT '12, 2012 Publication	<1%
25	physiolgenomics.physiology.org Internet Source	<1%
26	"Statistical Methods for Global Health and Epidemiology", Springer Science and Business	<1%

Media LLC, 2020
Publication

27	Zou, Lei, Kun Xu, Jeffrey Xu Yu, Lei Chen, Yanghua Xiao, and Dongyan Zhao. "Efficient processing of label-constraint reachability queries in large graphs", Information Systems, 2013. Publication	<1%
28	www.learningace.com Internet Source	<1%
29	"Databases Theory and Applications", Springer Science and Business Media LLC, 2016 Publication	<1%
30	link.springer.com Internet Source	<1%
31	Duncan Yung, Shi-Kuo Chang. "Answering How-to-Reach Query in Big Attributed Graph Databases", 2017 IEEE Third International Conference on Big Data Computing Service and Applications (BigDataService), 2017 Publication	<1%
32	Kamal Taha, Ramez Elmasri. "CXLEngine", Proceedings of the 2008 EDBT workshop on Database technologies for handling XML information on the web - DataX '08, 2008 Publication	<1%

33	scholarcommons.usf.edu Internet Source	<1%

Exclude quotes	On	Exclude matches	< 14 words
Exclude bibliography	On		

finding frequent subgraphs

Page 1 of 1 (3 results)

Ordered by title or year

Finding Frequent Subgraphs and Subpaths through Static and Dynamic Window Filtering Techniques

Research Article in EAI Endorsed Transactions on Scalable Information Systems

Authors: Bhargavi B., K. Swarupa Rani

Abstract:

Big data era has large volumes of data generated at high velocity from different data sources. Finding frequent subgraphs from the graph streams can be a challenging task as streams are non-uniformly distributed and continuously processed. Its applications include finding strongly interacting gro...

more »

Prefix Tree Based MapReduce Approach for Mining Frequent Subgraphs

Research Article in Ubiquitous Communications and Network Computing. Second EAI International Conference, Bangalore, India, February 8–10, 2019, Proceedings

Authors: Supriya Movva, Saketh Prata, Sai Sampath, R. Gayathri

Abstract:

The frequent subgraphs are the subgraphs which appear in a number, more than or equal to a user-defined threshold. Many algorithms assume that the apriori based approach yields an efficient result for finding frequent subgraphs, but in our research, we found out that Apriori algorithm lacks scalabi...

more »

Target Gene Mining Algorithm Based on gSpan

Research Article in Collaborative Computing: Networking, Applications and Worksharing. 14th EAI International Conference, CollaborateCom 2018, Shanghai, China, December 1-3, 2018, Proceedings

Authors: Liangfu Lu, Xiaoxu Ren, Lianyong Qi, Chenming Cui, Yichen Jiao

Abstract:

In recent years, the focus of bioinformatics research has turned to biological data processing and information extraction. New minir algorithm was designed to mine target gene fragment efficiently from a huge amount of gene data and to study specific gene expression in this paper. The extracted ge...

more »