Summarizing Static Graphs and Mining Dynamic Graphs

LIU, Zheng

A Thesis Submitted in Partial Fulfilment
of the Requirements for the Degree of
Doctor of Philosophy
in
Systems Engineering and Engineering Management

The Chinese University of Hong Kong
August 2011

Thesis/Assessment Committee

Professor Wai Lam (Chair)

Professor Jeffrey Xu Yu (Thesis Supervisor)

Professor Man-Cho So (Committee Member)

Professor Xuemin Lin (External Examiner)

This thesis is dedicated to my parents who brought me into this world,

Zengtao Liu and Ling Wu.

Abstract

Graph patterns are able to represent the complex structural relations among objects in many applications in various domains. Managing and mining graph data, on which we study in this thesis, are no doubt among the most important tasks. We focus on two challenging problems, namely, graph summarization and graph change detection.

The objective of graph summarization is to obtain a concise representation of a single large graph or a collection of graphs, which is interpretable and suitable for analysis. A good summary can reveal the hidden relationships between nodes in a graph. The key issue of summarizing a single graph is how to construct a high-quality and representative summary, which is in the form of a super-graph. We propose an entropy-based unified model for measuring the homogeneity of the super-graph. The best summary in terms of homogeneity could be too large to explore. By using the unified model, we relax three summarization criteria to obtain an approximate homogeneous summary of appropriate size. We propose both agglomerative and divisive algorithms for approximate summarization, as well as pruning techniques and heuristics for both algorithms to save computation cost. Experimental results confirm that our approaches can efficiently generate high-quality summaries.

In the area of summarizing a collection of graphs, we study the problem of summarizing frequent subgraphs, since it is not much necessary to summarize a collection of random graphs. The bottleneck for exploring and understanding frequent subgraphs is that they are numerous. A summary can be a solution to this

issue, so the goal of frequent subgraph summarization is to minimize the restoration errors of the structure and the frequency information. The unique challenge in frequent subgraph summarization comes from the fact that a subgraph can have multiple embeddings in a summarization template graph. We handle this issue by introducing a partial order between edges to allow accurate structure and frequency estimation based on an independence probabilistic model. The proposed algorithm discovers k summarization templates in a top-down fashion to control the restoration error of frequencies within σ. There is no restoration error of structures. Experiments on both real and synthetic graph datasets show that our framework can control the frequency restoration error within 10% by a compact summarization model.

The objective of graph change detection is to discover the changing areas on graphs when they evolves at a high speed. The most changing areas are those areas having the highest number of evolutions (additions/deletions) of nodes and edges, which is called burst areas. We study on finding the most burst areas in a stream of fast graph evolutions. We propose to use Haar wavelet tree to monitor the upper bound of the number of evolutions. Our approach monitors all potential changing areas of different sizes and computes incrementally the number of evolutions in those areas. The top-k burst areas are returned as soon as they are detected. Our solution is capable of handling a large amount of evolutions in a short time, which is consistent to the experimental results.

Besides finding changing areas based on the number of node and edge evolutions, a more interesting problem is to analyze the impact of these evolutions to graphs and find the regions that exhibit significant changes when these evolutions happen. The more different the relationship between nodes in a certain region is, the more significant this region is. This problem is challenging since it is hard to define the range of changing regions that is closely related to actual evolutions. We formalize the problem by using a similarity measure based on neighborhood random walks, and design an efficient algorithm which is able to identify the

significant changing regions without recomputing all similarities. Meaningful examples in experiments demonstrate the effectiveness of our algorithms.

中文摘要

在很多領域中，圖被用來表示對象之間的關系，因為它能夠描述對象之間複雜的結構信息。本論文研究了圖數據的管理和挖掘，重　放在關於圖數據的兩個具有挑戰性的問題：靜態圖的摘要，和動態圖的變化檢測。

靜態圖的摘要是指基於壹個大圖或是壹群圖，生成壹個易於用戶分析的并且大小適當的簡潔的表達形式。生成的摘要能夠揭示圖中的隱藏信息。單個圖摘要的難　在於如何生成高質量的，具有代表性的超圖。我們提出利用一个通用的熵框架去衡量超圖中　集的同質性。同質性最好的超圖的大小仍然很大，所以我們提出放鬆三個摘要標準以得到大小適當的摘要近似。我們提出了層次聚合和分裂兩個算法，並且在每個算法中，利用剪枝和啓發技巧節省計算量。實驗結果表明我們的方法可以快速生成高質量的摘要。

在概括壹群圖的研究方向上，因為生成隨機圖集合的摘要的意義不大，我們把重　放在概括壹系列的頻繁子圖。頻繁子圖是指在圖數據庫中頻繁出現的子圖，其最小頻率大於壹個給定的閥值。頻繁子圖分析的瓶頸在於子圖的數量，大量的頻繁子圖使得分析這些子圖變得很困難。生成頻繁子圖的摘要可以解決這個問題。頻繁子圖摘要的目標是降低結構信息和頻率信息的恢復誤差。我們提出利用獨立概率模型去生成頻繁子圖的摘要，使其沒有結構誤差，並且頻率的恢復誤差在可控制的範圍之內。基於真實和人造數據的實驗結果顯示我們的方法能有效的將頻率誤差控制在 10% 之內。

動態圖的變化檢測的目標是快速發現動態圖中變化的區域。變化最大的區域可以用　和邊的變化次數來衡量。我們研究如何在變化流中監控變化最大的區域。基於 Haar 小波，我們的框架能夠監控潛在變化區域的變化次數的上限。我们用增量方式計算不同大小的變化區域的總變化次數，所以能夠及時返回變

化最大的 k 個區域。我們的算法能夠在短時間內處理大量的變化流，實驗結果驗證了這壹　。

除了用　和邊的變化次數來衡量變化的程度，壹個更有趣的問題是分析這些　和邊的變化對圖的影響，尤其的是對　之間的關系的影響。哪個區域中　之間的關係變化越大，那麼這個區域就越重要。這個問題的難　在於如何確定變化區域的大小，使得它能夠真實反映實際的變化。我們提出基於鄰域隨機行走來衡量　之間的相似度，並基於這個相似度的變化來確定變化程度最大的區域。我們提出的算法避免了大部分　之間的相似度的重新計算。實驗結果給出了真實有意義的變化的例子。

ACKNOWLEDGEMENTS

I express my sincere gratitude to Professor Jeffrey Xu Yu. It might be the luckiest thing in my life to get to know Professor Yu, who opened the gate of research for me and is leading me on the road to a successful researcher. Professor Yu was my supervisor when I was an MPhil candidate in the Department of System Engineering and Engineering Management, and now he continues supervising me on my PhD study. Without his patient guidance, enthusing encouragement and considerable support, all the research projects I have done, as well as this thesis, would not be possible.

I am grateful to Professor Ruomin Jin, Professor Hong Cheng and Dr. Yiping Ke for all those cooperations in my research works. I would like to thank Professor Xuemin Lin for supporting my visiting to University of New South Wales and teaching me how to be a hardworking researcher.

It is my honor to have an outstanding thesis committee: Professor Wai Lam, Professor Xuemin Lin, and Professor Anthony So, to evaluate my research works and this thesis. Thank your all for your valuable time and constructive comments.

A lot of thanks to Doctor Jianjun Gao, Doctor Nan Tang, and my friends and colleagues in our database research group for all the interesting things we shared, all the joyful time we spent, and all the colorful life we lived. I also would like to take this opportunity to thank all the clerical staffs, all the technical staffs in the Department of System Engineering and Engineering Management for their wonderful support.

Finally, I want to say thank you to my parents and my fiancée Tina Chen for letting me fly freely in places far away. My parents never prevent me from any hobby, which makes me eventually be a man of wide interests. It was a luck coincidence for me to meet Tina, by whom the Chinese Music played is always enjoyable.

Contents

List of Tables

List of Figures

CHAPTER 1

INTRODUCTION

Graph patterns have the expressive ability to represent the complex structural relationship among objects in many applications in various domains, where graphs are the fundamental representation of data. In social networks, million of users are conducting numerous interactive activities (e.g., following, messaging, tagging, etc.) everyday. By modeling users as nodes and activities as edges, we can construct a huge social graph representing multiple relationships between uses. Photos users uploaded, music users listened, movies users watched, plus users themselves can also be modeled as a very large multipartite graph, where edges may indicate users' preferences. In World Wide Web, numerous web pages are hosted in different web sites. These web pages are connected by hyperlinks which allow users to go from one page to another page by clicking them. By viewing web pages as nodes and hyperlinks as edges, the whole web is a very large graph. In most search engines nowadays, search results are returned in an order which is partially determined by the structure of the graph. Machines hosting web sites can also be considered as nodes in graphs. In global computer networks, routers and hosts, plus the data links between them form a large graph. The particular characteristic of this graph is its location attribute, i.e., routers and hosts could be anywhere on the earth. In sensor networks, the communication range of a senor is usually limited. A sensor could interchange data with a certain number of nearby sensors. If we consider sensors as nodes, and add edges

between a sensor and all its neighbors within its communication range, this forms a graph. In areas other than engineering, graphs also exist. In biology, proteins, together with interactions between them, are viewed as graphs. Such protein-protein interaction networks are useful in revealing the relationships between the functions and the structures of proteins. In chemistry, chemical compounds are represented by graphs. A basic task in drug design is to find the active chemical compounds to a certain disease.

Due to the wide existences and the modeling abilities of graphs, researchers are attracted recently to put a lot of efforts in managing and mining graph data. The research in graph data management includes managing and indexing large amount of graph data for querying and searching. For example, given a query graph, (sub)graph matching is to find the matched (sub)graphs in a graph database. Given a graph and a node pair, reachability query is to determine whether there is at least a path between the node pair. The research in graph pattern mining includes discovering patterns, classes or clusters of graphs. For example, frequent subgraph mining is to find all subgraphs, whose number of embeddings is larger than a pre-defined threshold, in a collection of graphs. Clustering graph nodes is to discover the dense areas on graphs, in terms of the number of edges. These clusters can reveal the hidden relationships between nodes, because nodes in the same dense area are usually considered as similar to each other. Clustering graphs is to find the clusters of similar graphs or subgraphs, based on the underlying structures. Graph classification is to learn a classifier from labeled graphs, and class other graphs into different classes accurately.

Under the context of managing and mining graph data, we focus on two challenging problems in this thesis, namely, graph summarization and graph change detection. The difficulties of managing and exploring graph data lie in the large size of graphs themselves, and the huge number of graphs in a collection. The objective of graph summarization is to obtain an concise representation of a

single large graph or a collection of graphs for easy management and exploration.

In summarizing a single graph, a good summary can reveal the hidden relationships between nodes in a graph. The key task of summarizing a single graph is to construct a high-quality and representative summary, while keeping the summary size small. The summary is in form of a super-graph, where each node/edge in the super-graph represents a number of nodes/edges in the input graph. We propose the criteria of homogenous partition for summarization. The best summary is the one where all nodes and edges in the super-graph are homogenous. Unfortunately, it is almost as large as the input graph and still difficult for exploration. Then we propose an entropy-based unified model for measuring the homogeneity of the super-graph. Based on the unified model, we relax all the criteria of homogeneity in order to obtain an approximate homogeneous summary in appropriate size. We introduce both agglomerative and divisive algorithms for approximate summarization. In both algorithms, we present pruning techniques and heuristics for fast computation. Experimental results confirm that our approaches can efficiently generate high-quality summaries. This work is published in [39] and invited for publication in [38].

It is not much necessary to summarize a collection of random graphs, so we study the problem of summarizing frequent subgraphs in the task of summarizing a collection of graphs. The huge number of generated frequent subgraphs is the main bottleneck for users to explore and understand them. A compact summary which can represent both the structure and the frequency information of these subgraphs could be a possible solution to this issue. The objective of frequent subgraph summarization is to minimize the restoration error of the structure and the frequency information restored from summaries, that is, we can restore any frequent subgraph based on only compact summaries. We propose to use maximal frequent subgraph as summarization template graphs. To further reduce the size of summaries, we also use union of maximal frequent subgraphs as template graphs. The unique challenge here is that a subgraph can have multiple embed-

dings in a template graph, which can deteriorate the restoration accuracy. We solve this issue by introducing a partial order between edges. The restoration of structures and frequencies is based on an independence probabilistic model. We propose a top-down algorithm, which can discover k summarization template graphs by controlling restoration error of frequencies within σ. There is no restoration error of structures. Experiments on both real and synthetic graph datasets show that our framework can control the frequency restoration error within 10% by a compact summarization model. This work is the most recent work and submitted for publication.

In many applications related to graphs, graphs are not static but evolve all the time. New nodes or edges can join graphs, while old nodes or edges can leave graphs. These addtions/deletions of nodes/edges are called evolutions on graphs. A natural resultant problem is to determine the changing areas on graphs. There are two meanings of changes. One is based on raw evolutions, i.e., whether a region changes dramatically is measured by the number of evolutions happened inside it. The other is based on the relationship change. When evolutions happen, relationships between nodes also change, so the degree of change in an area could be measured by the variation of relationships between nodes in the area. Graph change detection is to discover these changing areas on graphs when they evolve fast.

Measured by the number of raw evolutions, the most changing areas, which are called burst areas, could be the regions with the most evolutions. We study on monitoring top k burst areas in a stream of graph evolutions coming in at a high speed. Here the potential burst areas could be hop areas of different sizes of all nodes. Our proposed approach monitors all these potential changing areas by using a structure based on Haar wavelet, by which the upper bound of the number of evolutions could be computed fast. Due to the number of changing areas are large, we propose an algorithm to compute the number of evolutions in large hop areas from the ones in small hop areas incrementally. In this way,

the computation is fast enough to cooperate with the stream environment. Once the top k burst areas are detected, they are returned as soon as possible. Our solution is capable of handling a large amount of evolutions in short time, which is consistent to the experimental results. Examples of burst areas in social networks are presented to show that they are meaningful burst areas. This work is published in [37].

One more interesting task is to analyze the impact of raw evolutions to the relationships between nodes on graphs. The significant changing areas are defined as the changing regions in which the relationships between nodes vary dramatically. The difficulties of this task are: (1) how to measure the relationships between nodes; (2) how to identify the appropriate region range that is closely related to the actual evolutions. We propose the neighborhood random walks to measure the similarity between nodes based on an analysis of different possible candidates. Under the context of evolving graphs, we design an efficient algorithm that is able to update the similarities without recomputing all of them. Once we identify the top nodes whose relationships to other nodes change dramatically, we expand from them to obtain subgraphs as the significant changing areas. Experiments demonstrate the effectiveness of our algorithms by presenting real meaningful examples. This work is published in [40].

The key contributions of the thesis is summarized below. The more detailed contributions in each specific applications are presented in later chapters.

1. In the problem of summarizing graphs, we introduce the new criteria for controlling the quality. In the problem of graph change detection, we formalized the problems from abstract concepts into detailed definitions.

2. We propose efficient and effective algorithms for summarizing graphs. The generated summaries are of high quality. We propose fast algorithm to monitor burst areas in graph evolution streams. We propose incremental algorithms to save computation cost in solving the problems of graph change detection.

3. We conduct extensive experiments on many real and synthetic datasets to verify the efficiency and effectiveness of our proposed algorithms.

The reminder of this thesis is organized in two parts. We present our work on graph summarization in Part I. Part I starts with an introduction to graph summarization in Chapter 2, followed by the related works in Chapter 3. We explain the details of approximate homogenous graph summarization in Chapter 4, and the details of frequent subgraph summarization in Chapter 5. We address our work on graph change detection in Part II. Part II starts with an introduction to this problem in Chapter 6, followed by the related works in Chapter 7. We explain the details of detecting burst areas in fast evolving graphs is presented in Chapter 8, and the details of spotting significant changing subgraphs in evolving graphs are described in Chapter 9. Finally, we conclude the thesis in Chapter 10.

Part I

Summarizing Static Graphs

CHAPTER 2

Introduction to Graph Summarization

It is not an easy task for users to manage and explore graph data, due to the complex structures, and the increasing sizes of graphs themselves, as well as the huge number of graphs in a collection. Graph summarization is a potential solution to this problem. In this part, we study the graph summarization problem in two different contexts: a single large graph, and a collection of frequent subgraphs. The objective of graph summarization is easy management and exploration, so the generated summary must be a concise representation of input graph(s), which is interpretable and suitable for analysis.

2.1. Summarizing a Single Large Graph

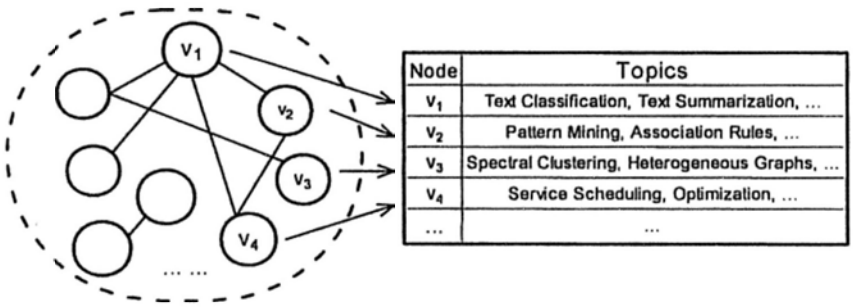
The goal of summarizing a large graph G is to obtain a concise graph representation G_S, which is smaller than G in size, for visualization or analysis. Although specific summarization representations can be various in different approaches, the main idea behind them is to construct a super-graph G_S with super-nodes and super-edges. The nodes in G are partitioned into several node sets and each node set is represented by a single super-node in G_S. Two super-nodes are connected by a super-edge in G_S if there exist edges in G between nodes from two corresponding node sets. The basic assumption is that nodes in

the same node set are similar to each other under the criteria of homogeneity, otherwise, using a single node to represent them will not be reasonable.

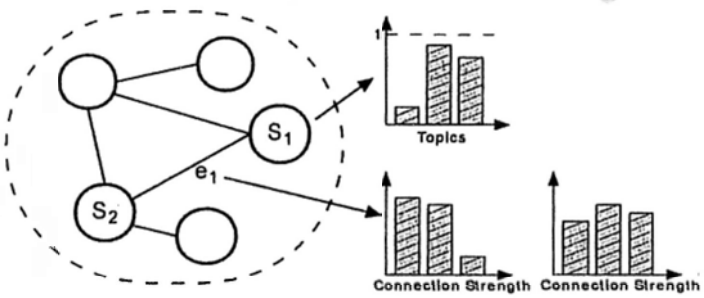
In the literature, there are two major approaches for super-graph construction, where the main difference lies in how to create super-edges between two super-nodes. A strict approach [42] requires that a super-edge exists between two super-nodes in G_S only if every pair of nodes residing in the two corresponding super-nodes is connected by an edge in G. A relaxed approach [50, 63] allows two super-nodes to be connected with a super-edge in G_S if there is at least one connected node pair in G among all the node pairs summarized by the two super-nodes. Here, each super-edge is associated with a participation ratio to indicate the percentage of connected nodes among all the nodes in the two super-nodes.

Unfortunately, both approaches have their disadvantages. In the strict approach, since only cliques or bipartite cliques can be represented by super-nodes according to the very rigorous requirement, the size of the summarized graph cannot be small in most cases, even when super-nodes are near-cliques, which makes the summarized super-graph still difficult to explore and access. In the relaxed approach, the issue lies in the quality of the summarization, which we will discuss soon in the later chapter. For example, if the participation ratio between two super-nodes is close to 1, it means almost all nodes in one super-node have neighbors in the other super-node. If the participation ratio is close to 0, it means almost no nodes have neighbors in the other super-node. So we can infer whether nodes in one corresponding node set may have edges connected to certain nodes in the other node set with high confidence. However, if the participation ratio is somewhat around 0.5, then the summarized super-nodes cannot provide much connection information of the neighborhood in the original graph. Because it implies that only partial nodes in one super-node have neighbors in the other super-node, and the chance of a node having neighbors almost equals the one of a random guess.

We focus on the information-preserving graph summarization for attribute



(a) DBLP Co-Author Network



(b) Our Proposed Graph Summary

Figure 2.1: An Example of Graph Summarization

graphs, which means the summarized representation must satisfy the quality criteria as much as possible. The summary for a graph in our solution consists of two parts: a super-graph and a list of probability distributions for each super-node and super-edge. Figure 2.1 shows a conceptual example. A DBLP co-author graph to be summarized is presented in Figure 2.1(a). Inside the dotted area is the structure information of the co-author graph, where nodes represent authors and edges represent collaborations between these authors. There is attribute information associated with authors possibly, for example, the table in Figure 2.1(a) associated with nodes shows the main research topics of each author. Figure 2.1(b) shows our proposed summarized representation, where the summarized super-graph is within the dotted area. Each super-node represents a number of authors, and is affiliated with a topic distribution indicating the research topics of the authors in the super-node, as well as the homogeneity of these research topics. Each super-edge has two connection-strength distributions indicating the homogeneity of the neighbor relationship between the nodes

in the two connected super-nodes in two different directions. We will have a careful analysis of the meaning of homogeneity in Chapter 4.

The major contributions of this research are summarized below.

- We focus on how to obtain an optimized approximate homogeneous partition on which a graph summarization can be constructed by relaxing both attribute requirement and structure requirements. Inspired by information theory, we propose a unified entropy model which unifies both attribute information and structural information.

- We propose a new lazy algorithm to compute the exact homogeneous partition by delaying the reconstruction of matrix, as well as two new approximate homogeneous algorithms aiming to find the optimized approximate partition.

- We conduct experiments on various real datasets and the results confirm that our proposed approaches can efficiently summarize a graph to achieve low average entropy.

2.2. Summarizing Numerous Frequent Subgraphs

Frequent subgraph mining has been an important research problem in the literature, with many efficient algorithms proposed [24, 29, 55, 59, 7, 22, 43]. Given a collection $\mathcal{D}$ of graphs, frequent subgraph mining is to discover all subgraphs whose frequencies are no less than a user-specified threshold f_{min}. Frequent subgraphs are useful in many applications, for example, as the active chemical structures in HIV-screening datasets, the spatial motifs in protein structural families, the discriminative features in chemical compound classification [15], and the index attributes [61] in graph databases to support graph queries.

One major issue of frequent subgraph mining is the difficulty of exploring and analyzing numerous patterns generated due to the exponential number of combinations. Given a graph with n edges, the total number of possible subgraphs could be 2^n. Tens of thousands of frequent subgraphs may be generated under a moderate minimum frequency threshold. This issue is inherited from frequent itemset mining, while on graph data, it is magnified much more due to the complex graph structure. A resulting solution for this issue is mining only closed or maximal frequent subgraphs [60, 23, 49], which generates fewer subgraph patterns. However, due to the structure complication and the rigid definition of maximal and closed subgraphs, maximal and closed graph patterns are still quite numerous. The difficulty to explore a large number of patterns still exists.

Frequent subgraphs may be utilized by machines or by users. For example, when frequent subgraphs serve as discriminative index features in graph databases, they are utilized by machines, where machines inspect individual frequent subgraph to find whether it is discriminative. In this case, the huge number of frequent subgraphs might not be the main issue. When frequent subgraphs serve as active chemical structures in HIV-screening datasets, they are mainly explored by users, where inspecting them one by one is almost an impossible task. Users focus on exploring frequent subgraphs as a whole set to obtain the comprehensive understanding of them. Then a concise representation of all frequent subgraphs is necessary for users in order to explore frequent subgraphs easily, and what is more, to make frequent subgraphs interpretable. It seems that sampling is a potential solution to this problem. The sampling approach is to select some frequent subgraphs as the representatives of all frequent subgraphs [13, 36, 64, 20, 5]. The representative subgraphs are similar to some frequent subgraphs, where the similarity measures may be maximum common subgraph, graph edit distance, etc. These representative subgraphs are dissimilar to each other based on a pre-defined threshold. While choosing a small number of rep-

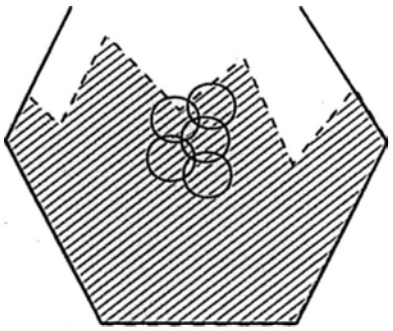


Figure 2.2: Summarization by Sampling

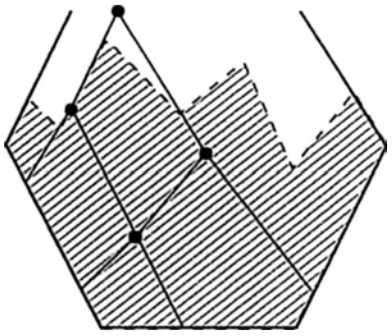


Figure 2.3: Summarization by Template Subgraphs

resentative subgraphs reduces significantly the number of output subgraphs, a problem is that it loses too much information about other unselected subgraphs, such as their structures and frequencies. A concept example is shown in Figure 2.2. Let the shaded areas denote a set of frequent subgraphs. The sampling approach uses rounded circles to cover all frequent subgraphs and only reports the centers of these circles as the representative subgraphs.

In frequent itemset mining, there have been several methods which use probabilistic models to summarize frequent itemsets [57, 56, 27]. These probabilistic models, as a concise summarization, are effective to restore the itemsets and their frequencies. In this paper, we aim to summarize frequent subgraphs by preserving the structure and frequency information of frequent subgraphs as much as possible. A concept example is shown in Figure 2.3. Let the shaded areas denote a set of frequent subgraphs. We are aiming to partition the whole set of frequent subgraphs into some subsets where the root of each subset, which is the black dots in Figure 2.3, is called template subgraph or union template subgraph. And all the frequent subgraphs in a subset are subgraphs of the (union) template subgraph of this subset. The black dot outside the shaded area means that it is a union template subgraph. This problem is more challenging than itemset summarization, due to two difficulties in subgraph mining: "multiple embeddings" (i.e., a subgraph can have multiple embeddings in a large graph.) and "topolog-

ical constraint" (i.e., the topological structure specifies the connectivity among nodes and edges.). To solve the problem, we make an independence assumption between edges in a frequent subgraph. We take a regression approach to estimate the parameters in the independence probabilistic model by least square estimation. To ensure a good summarization quality, we allow users to specify an error tolerance σ and our algorithms take a top-down approach to discover k template subgraphs. Multiple regression models will be built based on the k template subgraphs to control the frequency restoration error within σ.

The main contributions of this research are

- We introduce to summarize frequent subgraphs with an independence probabilistic model. Specifically, we propose to restore frequent subgraphs and their frequencies from template subgraphs by a regression approach to obtain a concise representation of all frequent subgraphs.

- We propose an efficient algorithm in a top-down fashion to discover a set of template subgraphs, together with the probabilistic models, as the summarization. Multiple regression models are built on these template subgraphs and the restoration errors are below a maximum error tolerance.

- We have evaluated our subgraph summarization approach on both real and synthetic graph datasets. Experimental results show that our method can achieve a concise summarization with high accuracy in terms of subgraph frequency restoration error.

CHAPTER 3

Related Works

In this chapter, we present an overview the related works to graph summarization, which are categorized into four parts.

3.1. Large Graph Summarization

There are a few existing works which are focusing on large graph summarization. Navlakha et al. [42] propose to substitute super-nodes for cliques in graphs without attribute information to generate summaries. Given a graph, each clique on the graph is represented by a super-node. The summary is a combination of the super-nodes and the original nodes that cannot be represented. If there is an edge between two super-nodes, or a super-node and a original node, then all the possible pairs of nodes must be connected by edges in the original graph. It is obvious that usually a graph cannot have many cliques, so they also use super-nodes to represent near-cliques or dense areas, with an extra table to record the edges that do not exist or need to be removed. The quality of a summary is measured by the size of the summary, which is based by Minimum Description Length (MDL) principle. MDL can find the best hypothesis leading to the best compression of data. Even with the help of the additional table, the compression ratio of a summary generated by the above method is still too large, which is almost one half of the size of the original graph. To further reduce the

summary size, an error bound ϵ is introduced for edges, that is, for an original node, if it or its super-node connects another super-node in a summary, then the number of missing edges is at most $(1 - \epsilon)$ times of the number of nodes in the other super-node. They propose both a greedy algorithm and a randomized algorithm to calculate the exact summary and the error-bounded summary. The greedy algorithm iteratively merges two nodes which introduce small extra space cost. The randomized algorithm randomly selects a node u, and finds a node v of small extra space cost in u's 2-hop neighborhood to merge with u.

Tian et al. [50] propose to summarize large attribute graphs by aggregating nodes into groups and use super-nodes to represent groups of nodes. The attributes are categorical. Two super-nodes are connected by a super-edge if there is at least one pair of nodes, one from each group, connected in the original graph. They require the nodes in each group having the same attribution information, so the total number of possible attribute values cannot be too many. Otherwise, the size of summaries will be too large for users to explore. On the super-graph, there is a participation ratio associated with each super-edge, which is the percentage of pairs of nodes that are connected among all potential possible pairs. They prove NP-completeness of this problem and present two heuristic aggregating algorithms in a bottom-up fashion and a top-down fashion, respectively. They design a merging distance mainly based on the similarity between participation ratio vectors. Two super-nodes have a small merging distance if their participation ratio vectors are similar. Given a graph, the bottom-up algorithm iteratively merges two super-nodes with the minimum merging distance until the number of super-nodes left is k. In the top-down algorithm, nodes in the graph are initially grouped into clusters and nodes in each cluster have the same attribute information. A super-node S_i is first selected to be split based on the number of the connection errors to its neighbors. Suppose S_j is a neighbor of S_i, and the number of the connection errors between S_i and S_j is the largest among all the neighbors of S_i. Then S_i is split into two super-nodes whose participation

ratios to S_j is 0 and 1. This procedure is repeatedly performed till there are k super-nodes. Their approach does not work well when the number of attribute values is not small and their criteria are not strict enough to obtain high-quality summaries.

Zhang et al. [63] extend Tian's approach [50] to summarize graph with two contributions. First, they propose to deal with numerical attribute values, not just categorical. Second, they recommend possible values of k, which is the number of super-nodes in summaries. Their algorithm for categorizing numerical values is agglomerative, which iteratively merges two value-adjacent super-nodes until no two super-nodes are value-adjacent. Then super-nodes of continuous values are cut into c groups of categories, where c is given by users. Next, they apply algorithms in [50] to generate summaries. During the splitting or merging process, their algorithm keeps tracking the interestingness measure of the current summary, and recommends the value of k. The interestingness measure is based on three characteristics: diversity, coverage and conciseness.

3.2. Graph Generation Models

Graph generating models can be considered as a summarization since they are able to partially reveal the hidden relationships between nodes in graphs. Chakrabarti et al. [12] study the problem from various points of views in physics, mathematics, sociology, and computer sciences. Based on the analysis of real social networks, the main characteristics they found for social graphs are power laws, small diameters and community effects. The characteristic of power laws indicates that most nodes in social graphs have few neighbors, while only a very small portion of nodes are of high degree. The characteristic of small diameters indicates that the distance between reachable pair of nodes is small, the effective diameter of the studied social graph is only 6. The characteristic of community effects indicates that nodes on graph can be grouped into clusters, whose clustering coefficients measure their clumpiness. Based on the above characteristics

of social graphs, they survey a lot of graph generators and suggest the possible solutions for each unique requirements.

Leskovec et al. [32] focus on the problem of generating a synthetic graph that has the same properties to a given one. The difficulty lies in that the parameters of the generating model must be consistent to the given graph. The authors utilize Kronecker product of matrix to achieve fast synthetic graph generation. They estimate the parameters of Kronecker model using maximum likelihood estimation. The estimation process is speeded up by the permutation distribution of the parameters. The same authors study the problem of evolving graph generator in [33]. Similar to [12], they first find the evolving rules from the sample graph data, including densification laws and shrinking diameters. Densification laws show that the average degree of nodes increases as time goes by, resulting in the smaller diameters of graphs. With these two observations, the forest fire model is introduced which simulates a burning fire of nodes and each node has a certain probability to link a new node which is found during the spread of the fire.

3.3. Frequent Subgraph Summarization

There are quite a number of works related to frequent subgraph summarization, but unfortunately, none of them could restore all frequent subgraphs within a certain restoration error. The main issue in frequent subgraph mining is the huge number of frequent subgraphs. Recently, researchers [13, 36, 64, 20, 5] have focused on selecting a small number of representative graph patterns to represent many similar subgraphs.

Chen et al. [13] select structural representatives from all frequent subgraphs based on clustering. Their proposed algorithm consists of two steps: smoothing and clustering. In the first step of smoothing, frequent subgraphs are grouped together if they have the same number of nodes, and the number of different edges is less than a threshold. After grouping, the support transactions of each frequent

graph are changed to the aggregation of transactions of frequent subgraphs in the same group. The support now is the number of the aggregated transactions. In the second step of clustering, these groups of subgraphs are further be partitioned into clusters based on the graph edit distance. The centroids of each clusters are selected as representative subgraphs.

Liu et al. [36] propose to select representative subgraphs based on two conditions. First, a selected frequent subgraph can cover a number of graphs. One graph can be covered by another graph if their support transactions are similar, measured by Jaccard distance. Second, the support transactions of the selected subgraphs must not be similar by the same measure. Three algorithms are introduced to find representative subgraphs. The first algorithm starts from the closed frequent subgraphs and find all subgraphs satisfied the second condition by subgraph matching. Then, for subgraphs cannot be represented by the current set of representative subgraphs, they are added to the set. This procedure is repeated until all the subgraphs are inspected. At last, the candidate representative set is further shrank to find the minimum representative set by removing subgraphs whose covered frequent subgraphs are subset of the covered frequent subgraphs of another representative subgraph. The other two algorithms search representative subgraphs directly from a graph collection. Both of them employ the framework of gSpan [59] to find frequent subgraphs. During the depth first search of all potential subgraphs, subgraphs which are covered by other mined frequent subgraphs are pruned directly.

Hasan et al. [20, 5] propose to discover representative frequent subgraphs by using random walks. The approach in [20] starts with finding all frequent edges. Then it chooses an edge randomly and repeats extending this edge by randomly selecting more edges from frequent edge set, as long as these edges are connected. In each step of extension, if it is a maximal frequent subgraph, a random walk is performed to exclude the nearby frequent subgraphs with similar structures. This procedure is performed iteratively until enough number of

frequent subgraphs are outputted. In [5], the authors select a small number of maximal frequent subgraphs through sampling without computing all the maximal frequent subgraphs. Once a frequent subgraph is identified, a subset of all its super and sub patterns are sampled to see if they are frequent subgraphs. Three sampling techniques are compared, which are uniform sampling, support-biased sampling and discriminatory sampling.

3.4. Frequent Subgraph Mining

The task before summarizing frequent subgraphs is to find all frequent subgraphs. Any summarization framework without pre-computing all frequent subgraphs could not restore all of them. Many algorithms have been proposed for finding frequent subgraphs in graph databases, where the frequency of a subgraph is the total number of graphs containing the subgraph in the database. Similar to the Apriori-based approaches in frequent itemset mining, Apriori-based algorithms for frequent subgraph mining are proposed in [24, 29, 55], where the search strategy follows a breadth-first manner in terms of number of edges. Subgraphs of small sizes are searched first. Once identified, a new larger candidate subgraphs are generated by joining two highly overlapping frequent subgraphs, which differ by one edge. So, in each iteration, the size of these candidate subgraphs is increased by one. Other algorithms [59, 7, 22, 43] employ a pattern-growth style. New candidate subgraphs are generated by adding a new edge to the current ones. It is possible that a candidate subgraph is extended from multiple frequent subgraphs. The gSpan algorithm in [59] constructs a depth-first search (DFS) tree for searching frequent subgraphs. Depending on the order of edges' addition, a DFS code is generated for each search tree. By using DFS codes, gSpan can prune the search space by duplicate removal without graph matching, which speeds up the computation. There are also research efforts on finding the frequent subgraphs in a single large graph [8, 34, 18, 31], where an important problem is how to define the frequency. A solution [31] considers the maximum

number of non-overlapping embeddings as the frequency.

CHAPTER 4

Approximate Homogeneous Graph Summarization

In this chapter, we present the details of our research work on approximate homogeneous graph summarization. This chapter is organized as follows. Section 4.1 analyzes the graph summarization problem carefully and Section 4.2 presents our concept of approximate homogenous partition based on information theory. We propose the summarization framework in Section 4.3 and report experimental results in Section 4.4.

4.1. Problem Statement

An attribute graph G is a triple (V, E, Γ), where V and E are the node set and the edge set of the graph, respectively. Γ is a finite set of attributes, and each node $v \in V$ or edge $(u, v) \in E$ is mapped to one or more attributes in Γ, denoted as $\Gamma(v)$ or $\Gamma(u, v)$. Given $\Gamma(v) = (A_1, A_2, \cdots, A_d)$, let $\gamma(v) = (a_1, a_2, \cdots, a_d)$ denote the attribute value vector of v, where a_i is the value of attribute A_i. In this work, we concentrate on categorical attributes. For a categorical attribute A_i with l distinct values, we can represent an attribute value using a l-bitmap, where all bits are zero except for the bit which corresponds to the attribute value. To simplify the presentation in this paper, we assume that the edges of the graph to be summarized are of the same attribute value, but our framework

can be extended to handle graphs which have multiple attributes associated with both nodes and edges.

Given an attribute graph, we aim to find a concise and interpretable summary which is friendly for users to explore and analyze. This can be done by partitioning all nodes V in a graph G into k homogeneous non-overlapping node sets $\{V_1, V_2, \cdots, V_k\}$, where the criterion of homogeneity is discussed later. Here, each V_i represents a non-empty subset of node set V. Let $\mathcal{P}$ denote the node partition $\{V_1, V_2, \cdots, V_k\}$, and let $\mathcal{P}(v)$ denote the unique node set V_i that a node v belongs. Furthermore, because a node v in a node set V_i has edges to link other nodes in another node set V_j, we use $N(v) = \{\mathcal{P}(v_k) | (v, v_k) \in E(G)\}$ to denote the set of V_j. In addition, for $V_j \in N(v)$, we use $|V_j|_v$ to denote the number of edges from v to any nodes in V_j.

Based on the homogeneous partition $\mathcal{P}$, a graph summarization G_S can be constructed as follows. A super-node S_i represents a node set V_i, for all node sets in $\mathcal{P}$, and all nodes of G summarized by a super-node in G_S have the same attribute values. The super-edges among super-nodes in G_S imply that every node of G summarized by a super-node has the same pattern of connecting nodes to other nodes summarized by other super-nodes. For example, suppose that S_i has super-edges to S_j, S_k, and S_l. It shows that every node of G summarized by S_i has edges to some nodes of G summarized by S_j, S_k, and S_l. In the following of this chapter, we use V_i and S_i interchangeably.

Now the question is what is a homogeneous partition. In a homogeneous partition $\mathcal{P}$, every node set V_i in $\mathcal{P}$ is considered to be homogeneous, which consists of the following three criteria: First, nodes are homogenous according to the attribute information, i.e., nodes in the same node set must have the same attribute value vectors. Second, nodes are homogenous according to the neighbor information, i.e., if a node $v \in V_i$ connects to V_j, then all the nodes in V_i must connect to V_j. Third, nodes are homogenous according to the connection strength, which is measured in terms of edges. If V_i and V_j are connected, all

nodes in V_i have the same number of edges to nodes in V_j. With these three criteria, we present the definition of exact homogeneous partition below.

Definition 4.1. Exact Homogeneous Partition. An exact homogeneous partition $\mathcal{P} = \{V_1, V_2, \cdots, V_k\}$ of a graph $G = (V, E, \Gamma)$ satisfies the following three criteria for every node $v \in V$: (1) $\gamma(v) = \gamma(V_i)$; (2) $N(v) = N(V_i)$; and (3) $|V_j|_v = |V_j|_{V_i}$, for every $V_j \in N(v)$. Here, $\gamma(V_i)$ denotes the common attribute values of nodes in V_i under the assumption that all nodes in V_i have the same attribute values. $N(V_i)$ denotes the common node sets for every node in the node set V_i, and $|V_j|_{V_i}$ denotes the common number of edges from every node in V_i linking to nodes in node set V_j.

The above definition of exact homogeneous partition extends the definition of exact grouping in [50]. The difference is that the exact grouping in [50] only considers the first two criteria but not the third one. Without the third one, nodes in a certain node set having more edges connecting to another node set, are considered to be the same as the one having less edges, which is obviously not reasonable. For example, in a DBLP co-author work, authors with more collaborations to a certain research group are more important than authors having few collaborations. It is not reasonable to place them together into the same node set apparently.

A summary G_S of a graph G constructed by an exact homogeneous partition can be considered as the best summarization with respect to the homogeneity criterion, since nodes in the same node sets are exactly the same in terms of attribute and structure information. Unfortunately, due to the high complexity of graph attributes and structures, as well as the increasing size of graph itself, such exact homogeneous partition cannot achieve a high compression ratio. The size of G_S based on the exact homogeneous partition is too large to serve as a graph summarization, which makes it beyond possible for users to handle. As we will see later in the experimental results, the size of G_S based on exact homogeneous partition can be almost as large as G. To solve this issue, we need

to relax partial or all the criteria in Definition 4.1. The approach in [50] loosens only the second criterion, by allowing nodes in V_i connect to similar node sets of V_j but not necessarily to be the same. But it still requires that all attribute values of nodes in the same node set V_i must be exactly the same vector.

It is questionable if it is sufficient to relax only the second criterion in Definition 4.1 due to the following issues: (1) Keeping the same attribute vector in each node set makes it very difficult to handle a graph with multiple attributes, in particular, when the number of attributes is not small. Suppose a node has m attributes and each attribute has d possible values, there are total d^m possible combinations of these values. Though the real existing combinations may not be so many, it is still impossible to find a partition of a relative small size, say k, such that all nodes in the same node set V_i have the same attributes, when k is less than the number of existing combinations. (2) In the third criterion in Definition 4.1, it requests that all nodes in the same node set V_i should have the same number of edges connecting to nodes in any other node set. Due to the various possibility of neighborhood structures, this can also lead to a graph summary which is not much smaller than the original graph G.

To achieve compact summarization G_S, we propose to relax all the criteria in Definition 4.1. In order to relax these criteria, a quality function for each criterion is needed to control the quality of relaxation. Let us first give a high level definition for approximate homogeneous partition, and explain it later.

Definition 4.2. Approximate Homogeneous Partition. Given a graph $G = (V, E, \Gamma)$, a number k, a graph node partition $\mathcal{P}$ is called approximate homogeneous partition, if it satisfies the following three criteria for every $V_i \in \mathcal{P}$. (1) $Q_\gamma(V_i) \leq \epsilon_1$; (2) $Q_{N(v_k)}(V_i) \leq \epsilon_2$; and (3) $Q_{|V_j|_{v_k}}(V_i) \leq \epsilon_3$, $\forall V_k \in (N(v_i) \cup N(v_j))$. Here, let $v_k \in V_i$, $Q_\gamma(\cdot)$, $Q_{N(v_k)}(\cdot)$, $Q_{|V_j|_{v_k}}(\cdot)$ are three quality measure functions, and ϵ_1, ϵ_2, and ϵ_3 are three thresholds to control the quality of the partition.

In an approximate homogeneous partition, nodes in the same node set are

considered to be homogeneous as long as their attributes and neighborhood relationship patterns to other node sets are similar to each other. Besides, a overall ranking function is necessary to rank the partitions based on the overall summarization quality to obtain the best one. Suppose $R(\cdot)$ is the function that reflects the three criteria in Definition 4.2 to measure the quality of approximate homogeneous partition, we study how to compute an approximate homogeneous partition $\mathcal{P}$ of size k for a graph $G = (V, E, \Gamma)$ by minimizing the ranking function $R(\mathcal{P})$. The key issues are as follows. What quality measure and the function $R(\mathcal{P})$ should we use? Can we make it threshold free (without ϵ_1, ϵ_2, and ϵ_3)? We address these issues in the following sections.

4.2. An Approximate Homogeneous Partition Based on Information Theory

In this paper, we propose an information-preserving criterion, based on information theory. We first review some background knowledge, followed by detailed discussions about how to utilize a unified entropy model to measure the quality of the three relaxations in Definition 4.2.

Let x_i be a boolean random binary variable and $p(x_i)$ be its Bernoulli distribution function, $\mathbf{p(x)} = [p(x_1), \cdots, p(x_d)]$ is a Bernoulli distribution vector [58] over d independent Boolean random variables $x_1, \cdots, x_d$. Let $\mathbf{b_j}$ denote a binary d-element vector. Given a set of binary vectors $D = \{\mathbf{b_1}, \cdots, \mathbf{b_n}\}$, under the assumption of independence, the probability by which they are generated by a distribution vector θ is estimated as

$$P(D|\theta) = \prod_{\mathbf{b_j} \in D} \prod_{i=1}^{d} p(x_i = \mathbf{b_j^i}), \tag{4.1}$$

where $\mathbf{b_j^i}$ is the ith element of the binary vector $\mathbf{b_j}$. The best θ, which fits the model, is

$$\hat{\theta} = arg\max_{\theta} \log(P(D|\theta)). \tag{4.2}$$

The well-known solution based on the maximum likelihood estimation is

$$p(x_i = 1) = \frac{\sum_{\mathbf{b_j} \in D}^{n} \mathbf{b_j^i}}{|D|}. \tag{4.3}$$

We use information theory to measure the quality of these distribution vectors. Recall that in information theory, entropy [14] is a measure of the uncertainty (randomness) associated with a random variable X, which is defined as

$$H(X) = -\sum_{x \in X} p(x) \log_2 p(x). \tag{4.4}$$

Consider a random variable x_i whose value domain is $\{0, 1\}$, the probability of x_i equals 0 or 1 is $p(x_i = 0)$ or $p(x_i = 1)$. The entropy of an unknown sample of the random variable x_i is maximized when $p(x_i = 0) = p(x_i = 1) = 1/2$, which is the most difficult situation to predict the value of an unknown sample. When $p(x_i = 0) \neq p(x_i = 1)$, we know that the value of the unknown sample is more likely to be either 0 or 1 accordingly, which is quantified in a lower entropy. The entropy is zero when $p(x_i = 0) = 1$ or $p(x_i = 1) = 1$. For a Bernoulli distribution vector $\mathbf{p(x)}$, assuming the contained random variables are independent of each other, the total entropy of a Bernoulli distribution vector is

$$H(\mathbf{p(x)}) = -\sum_{i=1}^{d} \sum_{x_i=0}^{1} p(x_i) \log_2 p(x_i). \tag{4.5}$$

If binary vectors within the set D are similar to each other, or homogeneous, then for each random variable x_i, most of its values should be similar, resulting in a low $H(\mathbf{p(x)})$.

4.2.1. Entropy-Based Relaxations of Criteria

In the following part, we discuss the three relaxations in Definition 4.2. Based on these observations, we can measure the quality of the three relaxations in a unified model inspired by information theory.

Observation for Q_γ: For each node $v_i \in V$, $\gamma(v_i) = (a_1, ..., a_d)$ is the attribute vector of v_i, where a_i is the value of attribute A_i. As mentioned, we repre-

	Node	a_1	a_2	a_3	a_4
S_1	v_1	1	1	1	0
	v_2	1	1	1	1
	v_3	0	1	1	1
S_2	v_4	1	1	0	0
	v_5	1	0	0	1
	v_6	0	0	1	1

	Entropy	$p(a_1=1)$	$p(a_2=1)$	$p(a_3=1)$	$p(a_4=1)$
S_1	1.85	0.66	1	1	0.66
S_2	3.70	0.66	0.33	0.33	0.66

Figure 4.1: Entropy-Based Attribute Homogeneity

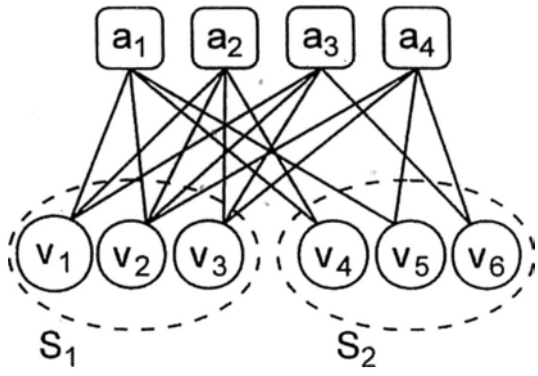


Figure 4.2: The Conversion from Attributes to Nodes

sent categorical attribute values as bitmaps, so we also use a_i to indicate the bitmap when there is no confusion. For a certain node set, V_j, in an approximate homogeneous partition, the attribute information of each node $v_i \in V_j$ is in form of a binary vector by concatenating these bitmaps together, denoted as $\mathbf{a} = (a_1, ..., a_d)$. The attribute information of a node set is homogeneous if the corresponding binary vectors are similar to each other. A binary Bernoulli distribution vector can be estimated from these vectors by Eq. (4.2). When the majority of nodes in a node sets share a same attribute value, the corresponding bit in the Bernoulli distribution vectors approaches to 1. When the majority of nodes do not have a certain attribute value, the corresponding bit approaches to 0. In this case, we can infer from the Bernoulli distribution where a node has or has not a certain attribute value by the expected value of the corresponding bit. Then it is better if each column in the Bernoulli distribution vector approaches to 1 or 0. When the value is 0.5, it is the worst case that we are uncertain to infer any useful attribute information, since the confidence of the expected value is like the one of a random guess. Entropy is an excellent quality measure in this case, and low entropy means high confidence based on Eq. (4.5).

As shown in Figure 4.1, each row in the top table represents a node in the graph. For each node, there are four attributes: (a_1, a_2, a_3, a_4). The first three

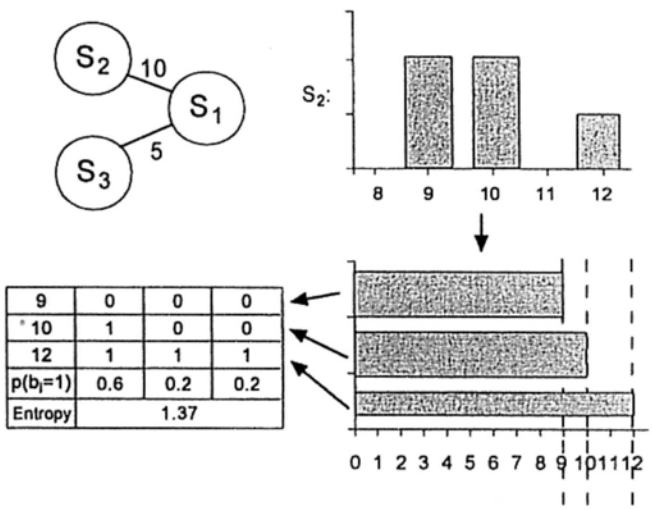


Figure 4.3: Entropy-Based Homogeneity of Connection Strength

rows belong to the super-node S_1 (or node set V_1), while the remaining belong to the super-node S_2 (or node set V_2). It is easy to see that nodes in S_1 are more similar to each other than nodes in S_2. The corresponding Bernoulli distribution vectors for S_1 and S_2, are represented in the lower table, as well as their entropy values. As we can see, the entropy value of S_1 is much lower than that of S_2, which is consistent to that nodes in S_1 are more similar to each other.

Observation for $Q_{N(v_k)}$: Nodes in the same homogeneous node set V_i should have similar neighborhood relationship in the super-graph. Note that if a node set has good quality according to the third criterion, it must be also good by the second one, since the second criterion is in fact a special case of the third one. If there is only one neighbor for nodes in a node set, then the second criterion and the third one are the same. Obviously, $Q_{|V_j|_{v_k}}$ is a stronger condition than $Q_{N(v_k)}$, because $Q_{|V_j|_{v_k}}$ measures the quality based on not only whether there are connections between nodes in V_i and V_j, but also the number of connections for $v_k \in V_i$. Therefore, we can ignore $Q_{N(v_k)}$, and concentrate on $Q_{|V_j|_{v_k}}$ which we will discuss next.

Observation for $Q_{|V_j|_{v_k}}$: Consider a super-graph G_S, and we use V_i (node set) and S_i (super-node) in G_S interchangeably. If there is a super-edge between

super-nodes S_i and S_j, then nodes in S_i should have similar total number of edges to nodes in S_j. As discussed, it is not appropriate to put two nodes together, whose connection strengths to a certain node set differ a lot, because their importance to the node set is not in the same level. We can keep two histograms for each super-edge (S_i, S_j), namely, S_i-to-S_j and S_j-to-S_i, to record the distributions of neighbors in S_j (S_i) of nodes in S_i (S_j). We explain it by using an example as shown in Figure 4.3. There are three node sets (super-nodes) in the partition: S_1, S_2, and S_3. At the upper left corner in Figure 4.3, it shows how these super-nodes are connected by super-edges. For example, it indicates that every node in S_2 has 10 neighbors in S_1 on average. The histogram of S_2-to-S_1 is drawn on upper right corner, where the x-axis indicates the number of neighbors in S_1 for a node in S_2. The y-axis indicates the number of nodes in S_2 corresponding to each value on x-axis. The histogram shows that there are 2 nodes within S_2 which have 9 edges connected to nodes in S_1, 2 nodes which have 10 edges and 1 node which has 12 edges. Intuitively, a homogeneous node set should have a tight spread range on x-axis in the histogram. Again, entropy is a good measure to show how homogeneous inside each node set. To do so, we present the histogram in another way as shown in the bottom right corner. The x-axis still indicates the number of neighbors in S_1 for a node in S_2, while the thickness of each bar indicates the number of nodes in S_1 corresponding to each value on x-axis. Based on this intuition, we transform each bar in the bottom histogram to a binary vector of all 1's. For example, for bar indicating the number of neighbors is 9, a binary vector of length 9 is constructed. We first concatenate 0's at the end of each binary vector to make them of the same length. Then we remove the common 1's in the suffix of these vectors, because the entropy on these columns are all zero and we focus on only the difference in these binary vectors. The remaining binary vectors are shown in the bottom left table in Figure 4.3. Similar to Q_γ, a Bernoulli distribution vector is learned from these binary vectors. The more similar these vectors are, the lower entropy

of the distribution vector is, as shown in the table.

In summary, the homogeneity of a node set can be measured by the concept of entropy of these Bernoulli Vectors. Let $\mathbf{e} = (S_i, S_j)$ denote a super-edge between two super-nodes S_i and S_j, and let v_i denote a node in super-node S_i. The entropy of super-node S_i consists of two parts: the attribute part and the neighborhood connection strength part. We propose to convert the attribute homogeneity into neighbor relationship homogeneity to unify the two parts. Figure 4.2 shows our conversion. For each attribute value, we add an additional node in the original graph G. Here we have four attribute values $\{a_1, a_2, a_3, a_4\}$, so we add four corresponding nodes in G. For each node, we add edges between it and those nodes corresponding to its attribute values. For example, in Figure 4.2, node v_1 has attribute values $\{a_1, a_2, a_3\}$, so we add edges between v_1 and nodes representing a_1, a_2, and a_3. In this way, we convert the attribute homogeneity into neighbor relationship homogeneity. Then, we apply the same approach as we have discussed in Observation for $Q_{|V_j|v_k}$ to calculate the attribute homogeneity. The entropy for S_i is

$$Entropy(S_i) = \sum_{j=1}^{k+l} H(\mathbf{p}(\mathbf{b_j^m} = \mathbf{1})), \tag{4.6}$$

where k is a user-given parameter for controlling the number of node subsets in the partition $\mathcal{P}$ and l is total number of distinct attribute values, $\mathbf{b^m}$ is the mth element in $\mathbf{b}$, and $\mathbf{p}(\mathbf{b_j^m} = \mathbf{1})$ is the Bernoulli distribution vector estimated by Eq. (4.3) for S_i to S_j or a_j, depending on whether the connections are to a super-node or attribute value node. As we can see from the above analysis, the total entropy for every super-node in exact homogeneous partition is zero.

Users might prefer attribute homogeneity over connection strength homogeneity or vice versa. To achieve this, we allow users to assign weights during the entropy calculation as follows in Eq. (4.7).

$$WeightedEntropy(S_i) = \lambda \sum_{j=1}^{l} H(\mathbf{p}(\mathbf{a_j^m} = \mathbf{1})) + (1-\lambda) \sum_{j=1}^{k} H(\mathbf{p}(\mathbf{b_j^m} = \mathbf{1})). \tag{4.7}$$

Now, $\mathbf{p}(\mathbf{a}_\mathbf{j}^\mathbf{m} = \mathbf{1})$ is the Bernoulli distribution vector estimated by Eq. (4.3) for S_i to node a_j, and $\mathbf{p}(\mathbf{b}_\mathbf{j}^\mathbf{m} = \mathbf{1})$ is the Bernoulli distribution vector estimated by Eq. (4.3) for S_i to super-node S_j. When λ equals 1/2, the entropy score is one half of the entropy computed by Eq. (4.6). And the weighted entropy for every super-node in the exact homogeneous partition is still zero.

The optimized approximate homogeneous partition is the partition that minimizes the ranking score of the super-graph, which is the total weighted entropy of all nodes:

$$R(\mathcal{P}) = \sum_{S_i \in \mathcal{P}} |S_i| \times WeightedEntropy(S_i), \tag{4.8}$$

where $|S_i|$ is the number of nodes contained in S_i. What we study next is how to find the optimized approximate homogeneous partition $\mathcal{P}$ for a given graph G. Based on $\mathcal{P}$, the graph summarization G_S can be constructed.

4.3. Homogeneous Graph Summarization

In this section, we present the algorithms for exact homogeneous partition and approximate homogeneous partition.

4.3.1. A Lazy Algorithm for Exact Homogeneous Partition

Exact homogeneous partition is the best summary in terms of homogeneity, and we extend the algorithm in [50] to compute exact homogeneous partition using a simple but effective approach.

Algorithm 4.1 outlines the procedures to compute the exact homogeneous partition based on Definition 4.1. Recall that $\mathbf{a}$ is the concatenated attribute vector for nodes. Suppose there are m distinct attribute vectors, the nodes in graph G are partitioned into m groups first according to the distinct vectors. Then the algorithm constructs an $n \times m$ node-to-group matrix M, where $M(i, j)$ is the number of v_i's neighbors in S_j. One thing worth noting is that nodes

Algorithm 4.1 A Lazy Algorithm for Exact Homogeneous Partition

Input: A graph $G = (V, E, \Gamma)$

Output: The exact homogeneous partition $\mathcal{P}$.

1: Partition V into m node sets based on distinct attribute value vectors $\mathbf{a}$;
2: Construct an $n \times m$ node-to-group matrix M;
3: **while** True **do**
4: Sort rows within each group;
5: Let $splitflag$ be an all-zero binary vector of length n;
6: **for** each cell $M(i, j)$ in M **do**
7: **if** $M(i, j) \neq M(i + 1, j)$ **then**
8: $splitflag(i)$ = True;
9: **end if**
10: **end for**
11: **if** $splitflag$ is all False **then**
12: break;
13: **end if**
14: Split each node sets according to $splitflag$ to form m' new node subsets;
15: Reconstruct the $n \times m'$ node-to-group matrix M;
16: **end while**
17: Output the exact homogeneous grouping $\mathcal{P}$.

belonging to the same group are stored adjacently in M and the order of groups in rows is the same as the order of groups in columns. At line 5, Algorithm 4.1 marks the split positions using a binary vector of length n. After inspecting all the groups, Algorithm 4.1 reconstructs the node-to-group matrix M based on the marked split positions.

Because the matrix reconstruction is costly, we do not reconstruct M immediately after a split position is found. There are many unnecessary reconstructions during the split operations. An example is shown in Figure 4.4. Suppose the left matrix is the initial node-to-group matrix after sorting, and we find S_1

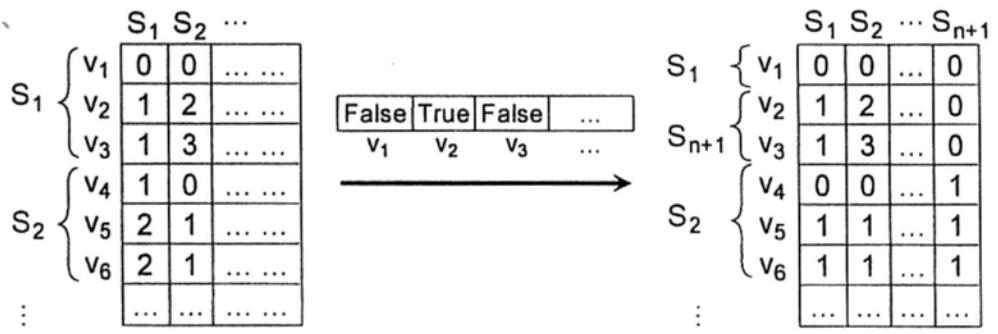


Figure 4.4: Data Structure for Lazy Exact Homogeneous Partition

should be split into two subsets. If we reconstruct the matrix in each loop, the matrix will be like the one on the right. As we can see that the next node set to be split is S_{n+1}, the last reconstruction of matrix is not necessary. Instead, we mark these split positions using a binary array and reconstruct only once after we check all the possible positions. We call it lazy exact homogeneous partition.

Next we will present two algorithms for approximate homogeneous partition: an agglomerative merging algorithm and a divisive k-means algorithm.

4.3.2. An Agglomerative Algorithm for Approximate Homogeneous Partition

As discussed, though exact homogeneous partition is of the highest quality based on homogeneity, its size is almost as large as the original graph. To further reduce the size of a summary of exact partition, we propose an agglomerative algorithm which is presented in Algorithm 4.2, which takes the exact homogeneous partition $\mathcal{P}$ as the input. The main idea of the agglomerative algorithm is to maintain a matrix to record the change in total weighted entropy for each pair of node sets if they are merged, and merges the pair with the minimum value repeatedly. Each merging will decrease the total number of node sets by one.

In the loop from line 3 to line 6, Algorithm 4.2 calculates the initial value of total weighted entropy of the exact partition after merging each possible node pair (V_i, V_j). Recall that the input is the exact partition, whose total weighted

Algorithm 4.2 The Agglomerative Algorithm for Approximate Partition

Input: The exact homogeneous partition $\mathcal{P} = \{V_1, \cdots, V_m\}$; a number k.

Output: The approximate homogeneous partition $\mathcal{P}_A$.

1: $\mathcal{P}_A = \mathcal{P}$;
2: Let Δ be an $m \times m$ empty matrix;
3: **for** each subset pair V_i and V_j in $\mathcal{P}$ **do**
4: $\quad \mathcal{P}_{ij} = \mathcal{P}_A \cup \{V_i \cup V_j\} \setminus \{V_i, V_j\}$;
5: $\quad \Delta_{ij} = R(\mathcal{P}_{ij})$ /* Eq. (4.8) */
6: **end for**
7: **while** $|\mathcal{P}_A| > k$ **do**
8: $\quad$ Let (V_l, V_m) be the pair of node sets with the minimum Δ_{ij};
9: $\quad \mathcal{P}_A = \mathcal{P}_A \cup \{V_l \cup V_m\} \setminus \{V_l, V_m\}$;
10: $\quad$ Update Δ based on V_l and V_m;
11: **end while**
12: Output the approximate homogeneous partition $\mathcal{P}_A$;

entropy is zero. At the end of line 6, Matrix $\Delta(i, j)$ stores the change of total weighted entropy if we merge node set V_i and V_j. Note that we only use the upper half of $\Delta(i, j)$ since $\Delta(i, j) = \Delta(j, i)$. In each iteration from line 7 to line 11, the algorithm merges the pair of node sets with the minimum change in total weighted entropy to generate new partition, and update matrix Δ.

Now, the problem is how to update matrix Δ. A naive way is to recompute the whole Δ based on the current partition $\mathcal{P}_A$, which is slow and not necessary, since merging one pair of nodes only affects partial values in Δ. Suppose (V_i, V_j) is the pair to merge, and $i < j$. The merging is done by adding all nodes in V_j to V_i and deleting V_j. This operation only affects the values in two types of cells in Δ. The first type is the cells for pairs involving V_i, which is easy to understand, since V_i is now changed to $V_i \cup V_j$. Thus, we have to recompute the change in total weighted entropy for these pairs of node sets.

The other type is the pairs of node sets involving the neighbors of (V_i, V_j).

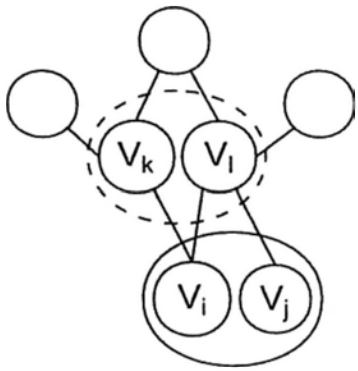


Figure 4.5: An Example of Updating Matrix Δ

An example is shown in Figure 4.5. Suppose V_k and V_l are neighbors of (V_i, V_j). It does not matter whether V_k and V_l are both neighbors of (V_i, V_j), or just one of them is. Before the merging of (V_i, V_j), $\Delta(k, l)$ stores the change in total weighted entropy if we merge V_k and V_l, while V_i and V_j are still separated. Once V_i and V_j are merged, the change of $\Delta(k, l)$ consists of three parts:

1. $|V_k \cup V_l| \times WeightedEntropy_{\{V_i,V_j\}}(V_k \cup V_l)$
   $-|V_k| \times WeightedEntropy_{\{V_i,V_j\}}(V_k) - |V_l| \times WeightedEntropy_{\{V_i,V_j\}}(V_l)$;

2. $|V_i| \times WeightedEntropy_{\{V_k \cup V_l\}}(V_i) + |V_j| \times WeightedEntropy_{\{V_k \cup V_l\}}(V_j)$
   $-|V_i| \times WeightedEntropy_{\{V_k,V_l\}}(V_i) - |V_j| \times WeightedEntropy_{\{V_k,V_l\}}(V_j)$;

3. The change of neighbors of (V_l, V_k) except V_i and V_j.

The subscript of $WeightedEntropy$ means the portion of the total weighted entropy related to node sets in the subscript. As we can see, after the merging of (V_i, V_j), the third part does not change, so we only need to recompute the first part and the second part.

4.3.3. A Divisive k-Means Algorithm for Approximate Homogeneous Partition

In this section, we present a divisive k-means based approximate algorithm to find the optimized approximate homogeneous partition using the Kullback-Leibler (KL) divergence. The Kullback-Leibler divergence [58] is a measure of the difference between two distribution vectors $\mathbf{p}$ and $\mathbf{q}$, which is defined as follows,

$$KL(\mathbf{p} \parallel \mathbf{q}) = \sum_{i=1}^{d} \sum_{x_i=0}^{1} p(x_i) \log \frac{p(x_i)}{q(x_i)}. \tag{4.9}$$

In the view of information theory, KL divergence measures the expected number of extra bits required to encode samples from $\mathbf{p}$ when using a code based on $\mathbf{q}$, rather than using a code based on $\mathbf{p}$. We assume the Bernoulli distribution vector for a certain node group S_i is $\mathbf{p}$. For each node in group S_i, let $\mathbf{q}$ be the Bernoulli distribution vector for a node $v_i \in S_i$. Both $\mathbf{p}$ and $\mathbf{q}$ are the concatenated vector of $\mathbf{a_j}$ and $\mathbf{b_j}$ in Eq. (4.7). Then we have

$$\begin{aligned}
&- \sum_{v_i \in S_i} KL(\mathbf{p}(\mathbf{x}) \parallel \mathbf{q}(\mathbf{x})) \\
=&- \sum_{v_i \in S_i} \sum_{i=1}^{d} (p(x_i = 1) \log \frac{p(x_i = 1)}{q(x_i = 1)} - p(x_i = 0) \log \frac{p(x_i = 0)}{q(x_i = 0)}) \\
=&- \sum_{v_i \in S_i} \sum_{i=1}^{d} (p(x_i = 1)(\log p(x_i = 1) - \log q(x_i = 1)) \\
&+ p(x_i = 0)(\log p(x_i = 0) - \log q(x_i = 0))) \\
=&\sum_{v_i \in S_i} \sum_{i=1}^{d} (-p(x_i = 1) \log q(x_i = 1) - p(x_i = 0) \log q(x_i = 0) - H(\mathbf{p})) \\
=&n(s_i) \sum_{i=1}^{d} (-\frac{\sum_{v_i \in S_i} p(x_i = 1)}{n(s_i)} \log q(x_i = 1) - \frac{\sum_{v_i \in S_i} p(x_i = 0)}{n(s_i)} \log q(x_i = 0)) \\
=&n(s_i) \sum_{i=1}^{d} (-q(x_i = 1) \log q(x_i = 1) - q(x_i = 0) \log q(x_i = 0)) \\
=&n(s_i) * H(\mathbf{p}(\mathbf{x})).
\end{aligned}$$

Thus, the optimized approximate homogeneous partition that minimizes $R(\mathcal{P}_A)$ is the partition that minimizes the sum of $KL(\mathbf{p}(\mathbf{x}) \parallel \mathbf{q}(\mathbf{x}))$, which leads to the following divisive k-means approximate algorithm presented in Algorithm 4.3.

Algorithm 4.3 starts from one node set by putting all the nodes in G together. In each loop from line 2 to 20, Algorithm 4.3 first splits the node set with the maximum total weighted entropy, and then applies k-means clustering method based on KL-divergence. The split procedure is from line 3 to line 14. First, a random perturbation of nodes in the node sets with the maximum weighted entropy is performed. Then we inspect these nodes one by one according to the order in the perturbation. If moving the node from the old node set to a new node set decreases the total weighted entropy, we move it, otherwise, it stays in the old node set. Once the split is finished, Algorithm 4.3 performs k-means clustering from line 16 to line 20, to minimize the sum of KL-divergence. When the number of node sets equals k, the approximate homogeneous partition $\mathcal{P}_A$ is returned.

4.4. Experimental Evaluation

In this section, we report the experimental results of our proposed summarization framework on various real datasets from DBLP Bibliography [1]. The algorithms are implemented by using matlab and C++. All the experiments were run on a PC with Intel Core-2 Quad processor and 3GB RAM, running Windows XP. One thing worthy noting is that we did not optimize our sources for multiple core environment.

Algorithm 4.3 The Divisive k-Means Algorithm for Approximate Partition

Input: A graph $G = (V, E, \Gamma)$, a number k;

Output: The approximate homogeneous partition $\mathcal{P}_A$.

1: $\mathcal{P}_A = \{V\}$;
2: **while** $|\mathcal{P}_A| < k$ **do**
3: Let V_m be the node set with the maximum $|V_m| \times WeightedEntropy(V_m)$;
4: Generate a random perturbation L of nodes in V_m,
5: $V_i = V_m$;
6: $V_j = \emptyset$;
7: **for** $v \in L$ **do**
8: $we = |V_i| \times WeightedEntropy(V_i) + |V_j| \times WeightedEntropy(V_j)$;
9: $we' = (|V_i| - 1) \times WeightedEntropy(V_i \setminus \{v\})$
$+(|V_j| + 1) \times WeightedEntropy(V_j \cup \{v\})$;
10: **if** $we' < we$ **then**
11: $V_i = V_i \setminus \{v\}$;
12: $V_j = V_j \cup \{v\}$;
13: **end if**
14: **end for**
15: $\mathcal{P}_A = \mathcal{P}_A \cup \{V_i, V_j\} \setminus V_m$
16: **repeat**
17: Evaluate the Bernoulli distribution vectors $\mathbf{a_j}$'s and $\mathbf{b_j}$'s for $V_k \in \mathcal{P}$;
18: Concatenate $\mathbf{a_j}$'s and $\mathbf{b_j}$'s together for $V_k \in \mathcal{P}_A$;
19: Assign each node $v \in V(G)$ to a new cluster V_k according to the Kullback-Leibler divergence in Eq. (4.9);
20: **until** the change of $R(\mathcal{P}_A)$ is small or no more changes of the cluster assignment
21: **end while**
22: Output the approximate homogeneous partition $\mathcal{P}_A$.

Table 4.1: The DBLP Bibliography Datasets

	Datasets	# of Nodes	# of Edges	Average Degree
D1	DM	1695	2282	1.35
D2	DB	3328	11379	3.42
D3	DB+DM	5023	15262	3.03
D4	DB+DM+IR	6184	18710	3.02

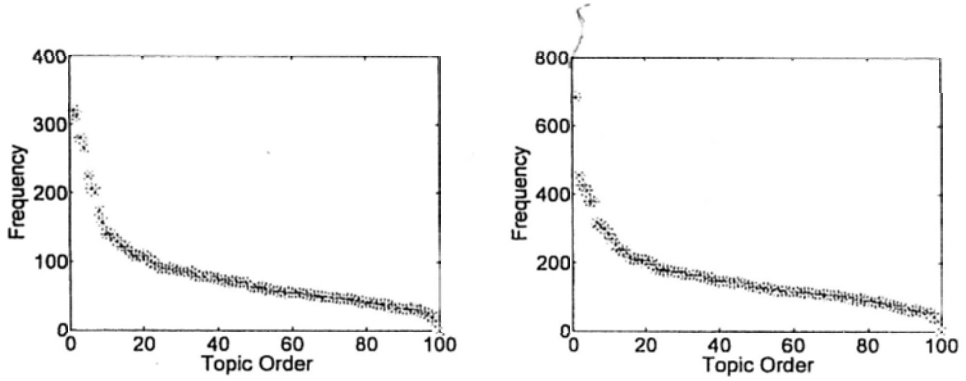
Table 4.2: The Keywords of Topics

Topics #	Keywords
32	text, classification, vector, categorization
66	mining, patterns, frequent, sequential...
76	service, scheduling, extending, media
80	clustering, matrix, density, spectral

4.4.1. Datasets

We construct a co-author graph with top authors and their co-author relationships, where the authors are from three research areas: database (DB), data mining (DM) and information retrieval (IR). Based on the publication titles of the selected authors, we use a topic modeling approach [21, 62] to extract 100 research topics. Each extracted topic consists of a probability distribution of keywords which are most representative for the topic.

By using authors from partial or all areas, we construct four real datasets in our experiments. The basic statistics of the four datasets are presented in Table 4.1, including the number of nodes, the number of edges and the average degree of nodes. There are total 100 topics in the original datasets and in the experiments, we remove the topics from authors, whose probabilities are extremely small. Each author is related to several topics whose probabilities are larger than 5%. Example of the topics are shown in Table 4.2, as well as the top keywords in



(a) D1 (DM) (b) D2 (DB)

Figure 4.6: Topic Frequency in Dataset D1 and Dataset D2

Table 4.3: The Exact Partition of The DBLP Bibliography Datasets

	D1	D2	D3	D4
# of Nodes	1695	3328	5023	6184
# of Distinct Attribute Vectors	1492	2931	4401	5409
# of Exact Partitions	1604	3219	4829	5912

each topic.

All these topics are not of equal importance. We present the frequency distribution of topics in datasets D1 (DM) and D2 (DB) in Figure 4.6 in descending order. The x-axis is the topic order and the y-axis the frequency of a topic which is defined as the number authors doing research on the topic. For dataset D1 in Figure 4.6(a), the majority of topics appear less than 100 times, while only less than ten topics are very hot among authors. For dataset D2, the frequencies of most topics are below 200.

4.4.2. Exact Homogeneous Partition

Table 4.3 presents a comparison between the number of groups and the nodes in the original graphs. The number of distinct attribute vectors and the number of exact groups are quite close to the number of nodes. Therefore, the exact homogeneous partition cannot obtain a graph summary of a reasonable

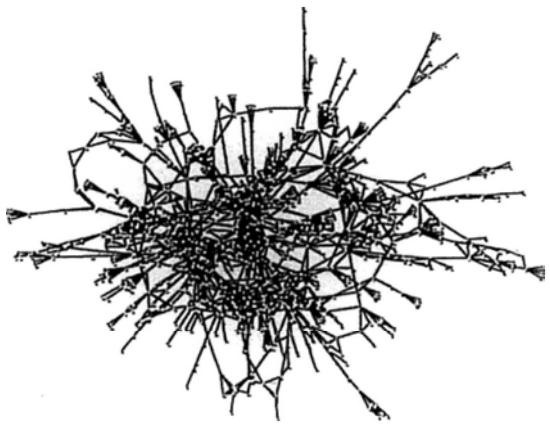


Figure 4.7: Exact Homogeneous Summarization of Dataset DM

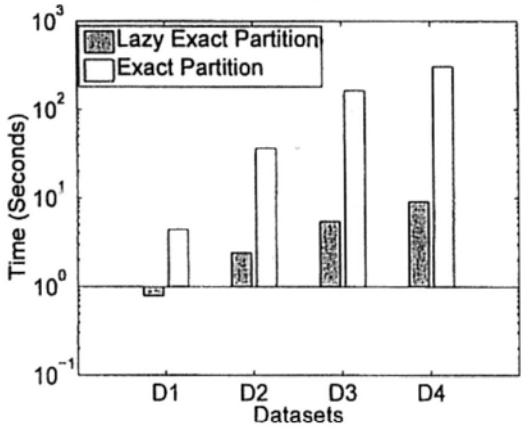
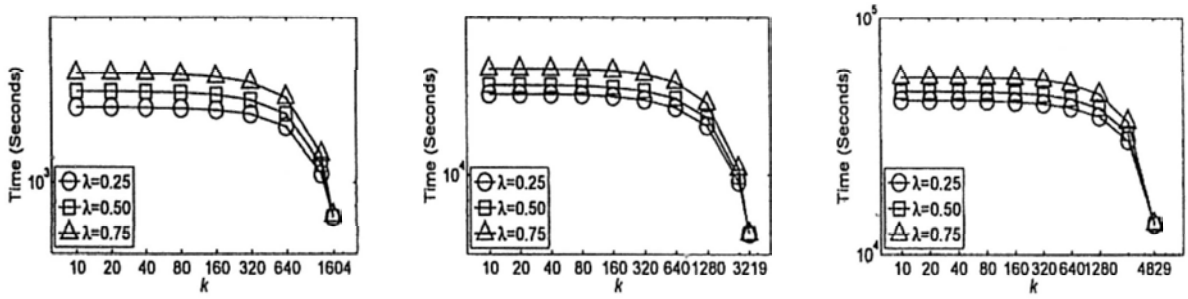


Figure 4.8: The Running Time of Exact Algorithms

size. Figure 4.7 shows the graph structure of the main connected component generated by exact partition algorithm on dataset DM, which is very large and not possible for users to explore. In Figure 4.8, we compare the running time of our lazy exact homogeneous partition algorithm with the exact partition algorithm, denoted as exact partition. Unlike the lazy partition algorithm, the exact partition algorithm reconstructs the matrix M in Algorithm 4.1 immediately after discovering a split position. The lazy exact partition algorithm is more than 10 times faster than the exact partition algorithm due to the saved time of matrix construction.

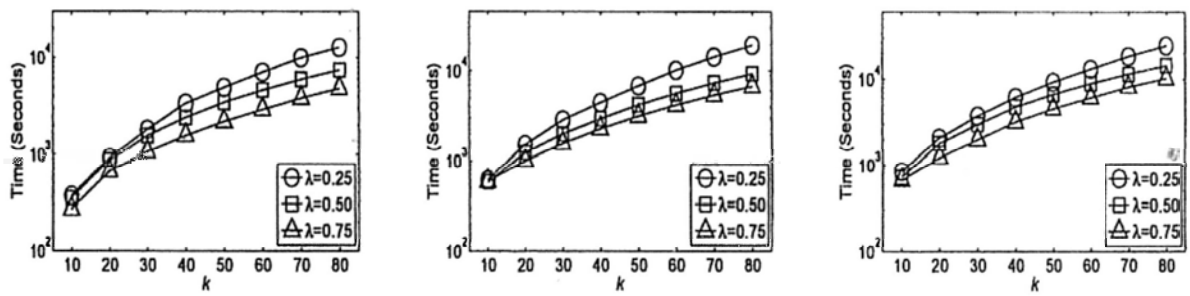
4.4.3. Approximate Homogeneous Partition

We performed our approximate homogeneous algorithms using three values of λ: 0.25, 0.5 and 0.75. Due to the high time complexity, we only apply our agglomerative algorithm on datasets D1, D2 and D3. Figure 4.9 shows the running time of the agglomerative algorithm performed on these three datasets. Both the x-axis and the y-axis are in log scale. We performed our divisive k-means algorithm on all the four datasets and report the results for datasets D2, D3 and D4 in Figure 4.10. k-means algorithm is almost 10x times faster than the



(a) D1 (DM) (b) D2 (DB) (c) D3 (DB+DM)

Figure 4.9: The Running Time of The Agglomerative Algorithm

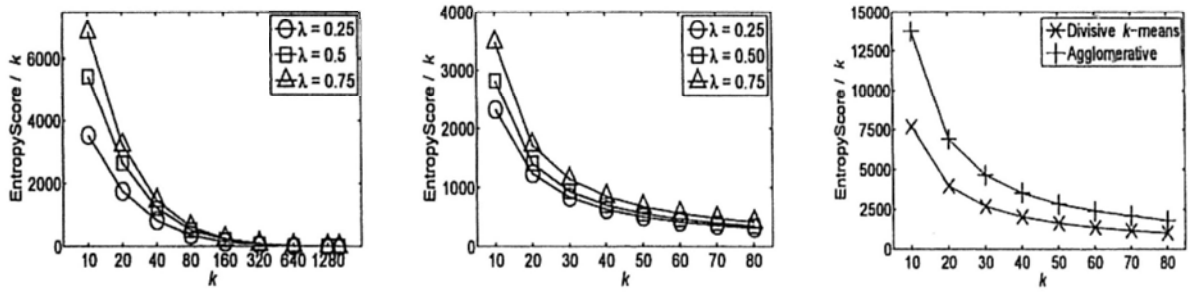


(a) D2 (DB) (b) D3 (DB+DM) (c) D4 (DB+DM+IR)

Figure 4.10: The Running Time of The Divisive k-Means Algorithm

agglomerative algorithm when k is small, which common in real applications. An interesting phenomenon is that the running time of small λ in the agglomerative algorithm is less, while the running time of large λ in the divisive k-means algorithm is less. The reason is that the agglomerative algorithm starts from the exact partition, and a large λ cannot boost the affect of attribute too much. In the divisive k-means algorithm, a large λ usually means less iterations in the k-means clustering algorithm, as we observed during the experiments.

Figure 4.11(a) shows the average entropy of the approximate homogeneous partition by the agglomerative algorithm on dataset D1, where we present the average entropy for different values of group number and λ. As the group number shrinks, the average entropy increases. Since the input of the bottom-up approximate algorithm is the exact homogeneous partition, the average entropy is 0 at the beginning. Figure 4.11(b) shows the average entropy of the approx-



(a) The agglomerative algorithm on D1 (DM)

(b) The divisive k-means algorithm on D1 (DM)

(c) D2 (DB), $\lambda = 0.5$

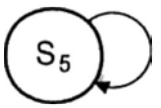
Figure 4.11: $R(\mathcal{P}_A)/k$

imate homogeneous partition by the divisive k-means algorithm on dataset D1. As we can see, when k is in the range from 10 to 80, the summary generated by the divisive k-means algorithm is much better that one generated by the agglomerative algorithm, in terms of the average entropy. Figure 4.11(c) reports the results on dataset D2 by these two algorithms when λ is 0.5, which once again shows that the divisive k-means algorithm performs better than the agglomerative algorithm, when k is small.

We present some interesting examples from summary of dataset D2 (DB), generated by the agglomerative algorithm when the group number is 60. For ease of presentation, we remove the distribution on edges, while the values of the entropy for these distributions are small. Each node in Figure 4.12 represents a group of researchers. The tables in Figure 4.12 present the topic number and the main keywords of each topic. Figure 4.12(a) shows that a group of researchers in time series domain tend to cooperate with themselves, where the size of node S_9 is 25. Figure 4.12(b) shows that researchers working on three different topics cooperate a lot, where the size of node S_5 is 28. We can infer from these keywords that these researchers are working on the core database technology. Figure 4.12(c) shows three groups of researchers cooperate a lot, where two of them mainly work on knowledge representation, while the third group mainly works on decision tree. The size of node S_9, S_{16} and S_{20} are 26, 12

Topic	Keywords:
12	time, series, real, subsequence

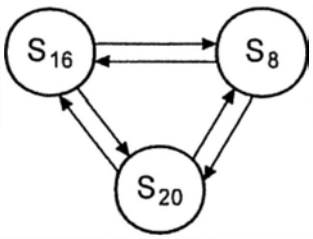
(a) Example 1



Topic	Keywords
54	implementation, db2, advanced, universal
90	database, object, oriented, serve, sql
75	control, concurrency, transaction, recovery

(b) Example 2

Topic	Keywords:
66	knowledge, base, representation, acquisition, bases

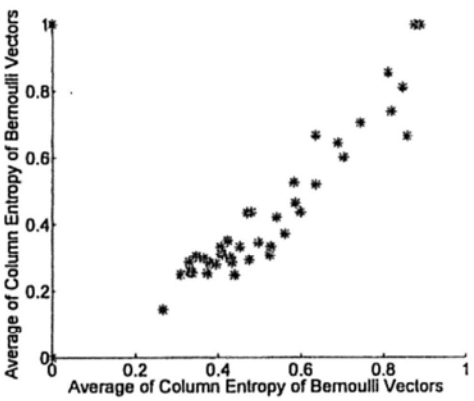


Topic	Keywords:
66	knowledge, base, representation, acquisition, bases

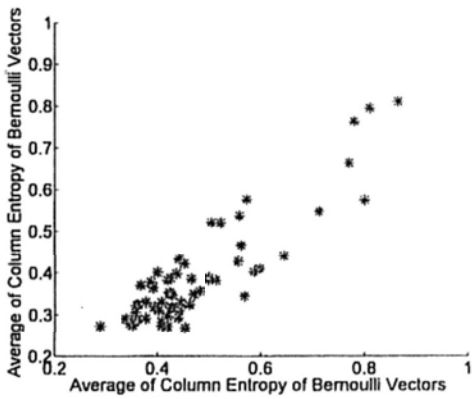
Topic	Keywords:
18	decision, trees, tree, induction

(c) Example 3

Figure 4.12: Real Examples from Summaries



(a) D1 (DM), $k = 40$



(b) D2 (DB), $k = 60$

Figure 4.13: Outliers Found by The Divisive k-Means Algorithm

and 35, respectively.

Figure 4.13 plots the average entropy of all columns in all the Bernoulli distribution vectors from datasets D1 and D2. The x-axis is the average entropy of attributes and the y-axis is the average entropy of connection strength. Figure

4.13(a) shows results from dataset D1 (DM) when $k = 40$. As we can see, most points are close to (0.3, 0.4) indicating a good confidence, while a few points are closed to (1, 1), which are considered as outliers in the summary. There are also outliers at (0, 1), which means these outliers have the same attribute information but not the neighborhood relationships.

Figure 4.13(b) shows results from dataset D2 (DB) when $k = 60$. Most points are close to (0.35, 0.4), while a few outliers are far away from the main cluster.

CHAPTER 5

Frequent Subgraph Summarization with Error Control

In this chapter, we address our research work on frequent subgraph summarization. This chapter is organized as follows. We formally define the problem of frequent subgraph summarization in Section 5.1 and propose the summarization algorithms in Section 5.2. Section 5.3 describes how to query the summarization to restore a subgraph and its frequency. We report the experimental results in Section 5.4.

5.1. Problem Statement

A graph G is a triple (V, E, Γ), where V and E are the node set and the edge set, respectively. Γ is a finite set of labels, and each node $v \in V$ or edge $(u, v) \in E$ is mapped to one or more labels in Γ, denoted as $\Gamma(v)$ or $\Gamma(u, v)$. A graph g is a subgraph of a graph G if there exists a subgraph isomorphism from g to G, denoted as $g \subset G$. G is called a supergraph of g.

Definition 5.1. Subgraph Isomorphism. For two graphs g and G, G contains a subgraph that is isomorphic to g, if there is an injective function $h : V_g \rightarrow V_G$, such that $\forall v \in V_g, \Gamma_g(v) = \Gamma_G(h(v))$, and $\forall (u, v) \in E_g, (h(u), h(v)) \in E_G$

and $\Gamma_g(u,v) = \Gamma_G(h(u),h(v))$, where Γ_g and Γ_G are the label set of g and G, respectively.

Definition 5.2. Frequent Subgraph. Given a collection $\mathcal{D}$ of graphs, a graph g is frequent if $f(g) \geq f_{min}$, where $f(g)$ is the number of graphs in $\mathcal{D}$ containing g, and f_{min} is a user-specified minimum frequency threshold.

A frequent subgraph g is a maximal one if and only if there does not exist another frequent subgraph g' and $g \subset g'$. A frequent subgraph g is a closed one if and only if there does not exist another frequent subgraph g', $g \subset g'$ and $f(g') = f(g)$. Anti-monotonicity holds for frequent subgraphs in a graph database, which means the subgraphs of a frequent subgraph are also frequent.

5.1.1. Subgraph Summarization and Restoration Error

Given a set $\mathcal{F}$ of frequent subgraphs, we aim to find a concise and interpretable summarization of frequent subgraphs which is friendly for users to explore and analyze. We decompose the meaning of friendly into two aspects: descriptive and informative. By descriptive, we can identify that a particular subgraph is a member from the summarization. By informative, the concise representation should maintain as much information of the frequent subgraphs as possible, to be specific, structures and frequencies. That is, users can restore the structure and the frequency of a certain subgraph accurately, based on only the concise summarization itself. If there is no error between the restored subgraphs and the original subgraphs, this summarization is lossless. While due to the high complexity of graph structures, the lossless summarization may not achieve a high compression ratio. In our proposed framework, there is no information loss in structures, while frequencies are summarized by probabilistic models. We define the relative restoration error as follows.

Definition 5.3. Average Relative Restoration Error. Let $\mathcal{F}$ denote the set of frequent subgraphs. For each subgraph $g \in \mathcal{F}$, $f(g)$ and $r(g)$ are the true

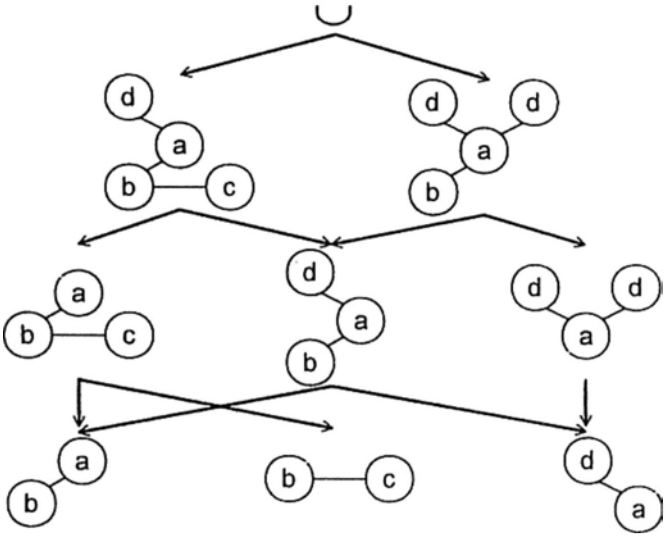


Figure 5.1: Partial Order Graph of Frequent Subgraphs Based on Containment Relationship

frequency and the restored frequency of g, respectively. The relative frequency restoration error of g is $|r(g) - f(g)|/f(g)$, denoted as $\delta(g)$. The average relative restoration error of the frequent subgraph set $\mathcal{F}$ is

$$\delta_{avg}(\mathcal{F}) = \frac{1}{|\mathcal{F}|} \sum_{g \in \mathcal{F}} \frac{|r(g) - f(g)|}{f(g)}. \tag{5.1}$$

Given the average relative restoration error as the summarization quality measure, the optimal frequent subgraph summarization can be defined in two ways. One is given a fixed integer as the number of partitioned subsets in summarization S, the total restoration error should be minimal. The other is given a maximum tolerance σ of the average relative restoration error, the number of partitioned subsets in summarization should be minimal. We adopt the latter one, because we aim to summarize frequent subgraphs with preserved frequencies.

5.1.2. Regression for Subgraph Summarization

Before we formally define the problem, let us first review the concept of regression models and how it adapts well in the context of frequent subgraph summarization. As mentioned, a frequent subgraph indicates all its subgraphs are frequent, which is called anti-monotonicity. Based on the subgraph containment relationship, a partial order graph (POG) is composed with all frequent subgraphs. Figure 5.1 shows an example. Each node in the POG is a frequent subgraph and subgraphs in the same level are of the same size, measured by the number of edges. In the POG, two subgraphs are connected by a directed edge from the larger one to the smaller one, if one is a subgraph of the other and differs by one edge. Suppose g is a subgraph in the POG, we use connected children to represent its connected neighbors smaller than g, and g is called connected parent. We use reachable children to represent all the nodes that can be explored by traveling along with the edges in the POG, starting from g. A maximal subgraph does not have a connected parent in POG. If there is more than one maximal frequent subgraph, we add a union of all maximal frequent subgraphs as the root of the POG. If there is more than one connected component in a POG, we handle each component one by one.

Suppose g and g' are two connected subgraphs in the POG, where g and g' differ by only one edge e, that is, $g \cup \{e\} = g'$. Let $p(g'|g)$ denote the conditional probability that a graph G from $\mathcal{D}$ containing g also contains g'. Since $g \cup \{e\} = g'$, we denote $p(g'|g)$ as $p(e|g)$, the conditional probability that a graph G from a graph database $\mathcal{D}$ containing g also contains edge e. Let $f(g)$ denote the frequency of a frequent subgraph g, then

$$p(g'|g) = p(e|g) = \frac{f(g')}{f(g)}. \tag{5.2}$$

Given any two frequent subgraphs g_1 and g_l that differ by $l-1$ edges $\{e_1, e_2, \cdots, e_{l-1}\}$, then

$$f(g_l) = f(g_1) \times p(e_1, e_2, \cdots, e_{l-1}|g_1). \tag{5.3}$$

Let g_i denote the graph $g_1 \cup \{e_1, \cdots, e_{i-1}\}$. Following the chain rule of conditional probabilities, we have

$$f(g_l) = f(g_1) \times \prod_{i=1}^{l-1} p(e_i|g_i). \tag{5.4}$$

In order to model the joint probability distribution, we apply the following independence assumption: whether a frequent subgraph g contains an edge e is independent of the structure of $g \setminus \{e\}$. Without loss of generality, we use $p(e)$ to denote $p(e|*)$, where $*$ denotes an arbitrary subgraph g that $g \cup \{e\}$ is frequent. Under this assumption, we can rewrite Eq. (5.4) for subgraph g_l and g_1 as follows:

$$f(g_l) = f(g_1) \times \prod_{i=1}^{l-1} p(e_i). \tag{5.5}$$

Given a frequent subgraph g, let $\mathcal{G} = \{g_1, g_2, \cdots, g_n\}$ be the frequent subgraphs reachable from g in the POG. Suppose we know the frequency $f(g)$ of g, as well as all the probabilities $p(e_j)$ of edges in g, with the independent assumption, we can estimate the frequency of each graph $g_i \in \mathcal{G}$ according to Eq. (5.5). By applying the logarithmic transformation on both sides of the equation, we have

$$\log f(g) = \log f(g_i) + \sum_{j=1}^{i-1} \log p(e_j). \tag{5.6}$$

Similar to the regression approach in [27], we can build a regression model $Y = X\beta + E$ for $\mathcal{G}$, where E is the matrix of error terms,

$$Y = \begin{bmatrix} \log f(g) - \log f(g_1) \\ \cdots \\ \log f(g) - \log f(g_n) \end{bmatrix}, X = \begin{bmatrix} \mathbf{1}_{e_1 \in g_1} & \cdots & \mathbf{1}_{e_l \in g_1} \\ \cdots & \cdots & \cdots \\ \mathbf{1}_{e_1 \in g_n} & \cdots & \mathbf{1}_{e_l \in g_n} \end{bmatrix} \text{ and } \beta = \begin{bmatrix} \log p(e_1) \\ \cdots \\ \log p(e_l) \end{bmatrix}. \tag{5.7}$$

Here, $\mathbf{1}_{e_i \in g_j}$ is an indicator that edge e_i belongs to graph g_j. $\mathbf{1}_{e_i \in g_j} = 1$ if $e_i \in g_j$, and $\mathbf{1}_{e_i \in g_j} = 0$, otherwise. The least square estimation [46] of the above regression model is to minimize the sum of squares of the errors (residues), which is

$$\delta = \min_{\beta} \Big\{ (Y - X\beta)'(Y - X\beta) \Big\}. \tag{5.8}$$

Then the solution is

$$\hat{\beta} = \arg\min_{\beta} \left\{ (Y - X\beta)'(Y - X\beta) \right\} = (X'X)^{-1}X'Y. \tag{5.9}$$

Union of Maximal Frequent Subgraphs: By applying the above regression approach, we are able to summarize any frequent subgraph g in the POG, together with all its reachable children, as a regression model. We call g a template subgraph, or a template for brief. Recall that in the POG of frequent subgraphs, each node represents a frequent subgraph, which is either itself a maximal frequent subgraph or a subgraph of a maximal frequent subgraph. In order to maintain the information of all frequent subgraphs, each maximal frequent subgraph needs to be represented by a regression model. Therefore, the total number of such models can be as large as the number of maximal frequent subgraphs, which is too many. To solve this issue, we introduce a union of maximal frequent subgraphs as a template, called union template. For example, given two maximal frequent subgraphs g_1 and g_2, if there are common sub-structures between them, we consider to merge them into a compact union by eliminating the duplicated sub-structures as much as possible. It is worth noting that two maximal frequent subgraphs without any common sub-structure can not be merged into a union template, because if they are merged, the union template is not a connected graph. A single regression model built on a union template that is not a connected graph is of no difference from a set of regression models built on each connected component (maximal frequent subgraph) separately. Because first, the number of parameters in the single regression model and the total number of parameters in the set of regression models are the same. Second, the average relative restoration error of the single regression model and the average relative restoration error of the set of regression models are the same. Details of how to construct a union template is discussed in Section 5.2.2.

5.1.3. Problem Definition

It is obvious that the accuracy of the estimated regression models depends on whether the independence assumption is valid among the reachable children of a template subgraph, which is not common in mostly graph databases. So, we divide all frequent subgraphs into groups and apply the independence assumption locally on each group to make sure that the restoration error can be controlled. We present the formal definition of the frequent subgraph summarization with error tolerance below.

Problem 5.1. Frequency-Preserved Subgraph Summarization with Error Tolerance σ. Given a set of frequent subgraphs $\mathcal{F}$, and a maximum average relative error tolerance σ for the restored frequency, the problem of frequent subgraph summarization is to partition $\mathcal{F}$ into as few groups as possible, and each group $\mathcal{G}$ satisfy the following: (1) $\mathcal{G}$ can be summarized as a single regression model, and (2) $\delta_{avg}(\mathcal{G}) \leq \sigma$, where δ_{avg} is defined in Definition 5.3.

The meaning of requirement (1) is that the template subgraph of each group should be either a frequent subgraph in $\mathcal{F}$ or a connected union template subgraph, which is a union graph of maximal frequent subgraphs. Without this requirement, we can easily merge any number of template subgraphs into a large unconnected one and create a single regression model to reduce the number of groups.

5.2. Summarization Algorithms

Before explaining our framework in details, let us discuss some unique challenges in summarizing subgraphs.

5.2.1. Challenges

The problem of summarizing frequent subgraphs is an extension of concisely summarizing a large collection of frequent itemsets, which have been studied [4, 57, 56, 27]. The key criteria to evaluate the quality of a summarization lie in two aspects: coverage and frequency. First, the concise representation is capable of representing all frequent itemsets in the collection, which is usually based on the set containment relationship. Second, the frequency of any frequent itemsets can be estimated from the concise summarization accurately. The existing researches have made significant contributions for summarizing frequent itemsets.

However, it is much more challenging to summarize a set of frequent subgraphs which can meet these two key criteria. The challenge comes from two fundamental difficulties in subgraph mining: "multiple embeddings" and "topological constraint". The multiple embedding problem refers to the issue that given a template subgraph pattern g, for a targeted frequent subgraph g_i, we may find several isomorphic embeddings of g_i in g, as shown in Figure 5.2. The right subgraph has two embeddings in the left one. Thus, even though we can determine the subgraph is covered by a template subgraph, it becomes a problem if we try to apply the template subgraph to the frequency estimation (which is typically done in the frequent itemset summarization). When we consider multiple embeddings as observations, this will increase the residue in the regression model, as well as the frequency restoration error and the estimation cost of regression models.

In addition, the multiple embedding problem can further complicate the frequent subgraph partition problem. Generally, in frequent itemset mining, we can easily split the collection of patterns into two sets, one including an item and the other not. Considering we have a template itemset pattern, we can easily represent its two sub-collection of patterns. However, considering a template subgraph, we consider splitting its covered frequent subgraphs into two groups, one including an edge and the other not. Even though the similar strategy

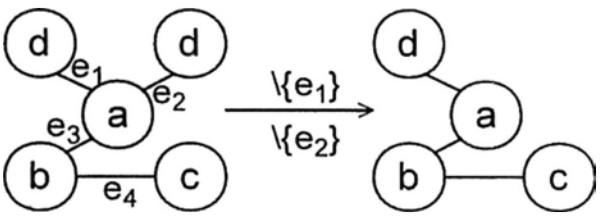


Figure 5.2: An Example of Multiple Embeddings

can be applied to template subgraphs by simply fixing one edge in the template subgraph (for one group) and dropping the edge (for the other group), the related issue is the topological constraint. We need to perform subgraph matching to determine whether one subgraph is covered or not. In the typical frequent itemset summarization, the probabilistic method is either based on item independence or conditional independence model. Can such models be applied to subgraphs? How does the topological constraint affect the models? These problems need to be addressed in frequent subgraph summarization.

To handle this issue, if a subgraph has multiple embeddings in its corresponding template, we only select one embedding in the construction of the regression model, because, as discussed, selecting all embeddings increases the residue of regression models, and results in a larger restoration error. A selection strategy is good if there is no need for extra storage cost to record selections. And apparently, we should follow the same strategy in a single regression model to further save the storage space. We set up a lexicographical order among the edge sets of frequent subgraphs in the following manner. Each edge in the template subgraph is assigned to a global id, such that all the frequent subgraphs covered by the template subgraph can be represented by a set of edge id's. By introducing a partial order between these id's, we can select the embedding whose id set is the smallest or largest according to the lexicographical order of edge id sets.

Theorem 5.1. *The average relative restoration error of a regression model by selecting the embedding with the smallest id set is the same as one of the regression model by selecting the embedding with the largest id set.*

The proof of Theorem 5.1 is obvious by reversing the partial order of edge id's and the one with the largest id set before reversing now become the smallest one after reversing. In our framework, we always select the smallest embedding of a subgraph. For example, in Figure 5.2, if $e_1 \prec e_2$, then the right graph is obtained by removing e_2 from the left one.

The second challenge comes from multiple regression models. When there is more than one template subgraphs, a frequent subgraph could belong to more than one template subgraphs, as we can see from the POG in Figure 5.1. For example, if the template subgraphs are the maximal frequent subgraphs, then the frequent subgraphs in the lower levels belong to all these template subgraphs. In this situation, a simple solution is to manipulate each template subgraph separately and consider the intersection part as observations to all template subgraphs containing them. Again, this will increase the residue in the regression, thus large restoration error, as well as overhead regression model estimation cost. One may argue that we can assign the intersection part to the template subgraph which achieves smallest total restoration error. Suppose we have k template subgraphs, this solution is not feasible, due to the maximum possible number of intersection part is 2^k or at least $k^2 - k$ if only considering intersections between two templates. What's more, even if the optimal assignment is obtained, and the total restoration error is minimized, a subgraph cannot be easily identified from generated models except we store the invert index of all the assignments, which, in the worst case, might be as large as the frequent subgraph set.

Our solution is to restrict the intersection part between template subgraphs to belong to only one template subgraph. The assignments are recorded in the POG by removing the unnecessary edges to make sure that each node (subgraph) can only be traced back to one template subgraph. As discussed, it is not feasible to compute the optimal assignment with the minimum total restoration error. We choose a good way by assigning the intersection part of two template reachable children to one that has smaller size in term of the number of edges.

Algorithm 5.1 The Summarization Framework

Input: POG $\mathcal{G}$, Error tolerance σ

Output: Template Subgraphs with Regression Models

1: $T = \{$the root of POG $\mathcal{G}\}$;
2: **while** true **do**
3: $g' = \arg\max_g \{\delta_{avg}(g) | g \in T\}$;
4: **if** $\delta_{avg}(g) > \sigma$ **then**
5: $T = T \cup$ **divide**(g');
6: **else**
7: break ;
8: **end if**
9: **end while**
10: Return T.

The approach has several advantages: (1) In general, by assigning interactions to smaller templates, we can balance the number of frequent subgraphs in each regression model. (2) The independence assumption is dangerous in practice. By assigning interaction parts to one template subgraph, we in fact split the frequent subgraph set consisting of all the reachable children of these two template subgraph into two conditional set, and transform the independence assumption into a conditional independence assumption. (3) The frequent subgraphs in intersection parts can be easily maintained for queriable summarization. Details are discussed in Section 5.3.

5.2.2. Template Subgraph Division

Algorithm 5.2 presents procedures of how to divide a template subgraph. The input template subgraph g can be either a real maximal frequent subgraph or a union of several maximal frequent subgraphs in the POG. We discuss them below.

Summarization Framework

As mentioned, we call the root graphs of regression models as template subgraphs. Our summarization framework is presented in Algorithm 5.1, which is done in a top-down fashion. The algorithm starts from a single template subgraph, the root of the POG, which is a union template subgraph for all maximal frequent subgraphs. Let g be a template subgraph, we use $\delta_{avg}(g)$ to denote the average restoration error of the regression model build on g with its reachable children. In each repeated loop from line 2 to line 9, the algorithm repeatedly divides the template subgraph whose average restoration error is larger than the one of any other template subgraph, and the threshold σ, into two template subgraphs, until all the template subgraphs have a average restoration error $\leq \sigma$.

Template subgraph is a frequent subgraph

We discuss how to handle a real frequent subgraph. The corresponding part in Algorithm 5.2 is from Line 2 to 12. When the template subgraph g to be divided is a single frequent subgraph, the potential new template subgraphs are the connected children of g. Take Figure 5.3 as an example. Assume g is the template subgraph to be divided, and c_1, c_2, and c_3 are g's connected children in the POG. Suppose c_i is selected ($1 \leq i \leq 3$). Then we have two template subgraphs: c_i and g. There exist frequent subgraphs that are the descendants of both c_i and g. We restrict them to belong to only one template subgraph, either c_i or g, in order to obtain better regression models. The rule is to let the sharing part belong to the smaller template subgraph c_i.

We discuss how to build regression models for this case. Let e be the edge that appears in g but does not appear in c_i. All the descendants of c_i in the POG do not contain e. In other words, the descendants of g are divided into two parts: frequent subgraphs containing e and frequent subgraphs not containing e. By selecting c_i, the POG rooted at g is divided into two subgraphs. One POG subgraph G_i is rooted at c_i and contains all descendants of c_i. The other POG subgraph G_g contains g and all its descendants excluding those in G_i. As

Algorithm 5.2 divide(Template g)

1: **if** g is a non-union template subgraph **then**
2: $\mathbf{C}$ = the directed children of g in POG $\mathcal{G}$;
3: **Cand** $= \emptyset$;
4: **for** Each $c_i \in \mathbf{C}$ **do**
5: let G_i be the POG subgraph of g rooted at c_i;
6: let G_g be $g \setminus G_i$ /*g corresponds to G_g*/;
7: Build regression models R_i and R_j for G_i and G_g (or equivalently c_i and g);
8: ϵ_i = total residue of R_i;
9: ϵ_g = total residue of R_g;
10: **Cand** = **Cand** $\cup \{(c_i, g, \epsilon_i + \epsilon_g)\}$;
11: **end for**
12: $(c_{min}, g_{min}) = \arg\min_{c_i}\{\epsilon \mid (c_i, g, \epsilon) \in \mathbf{Cand}\}$;
13: Update the regression models for c_i and g in the POG $\mathcal{G}$ based on c_{min} and g_{min};
14: Return $\{g_{min}, c_{min}\}$
15: **else**
16: let $\mathbf{C}$ be the children of g, $(g_1, g_2, \cdots, g_k)$, in order;
17: **Cand** $= \emptyset$;
18: **for** $i = 1$ to k **do**
19: Build a regression model R_{1i} for $(g_1, \cdots, g_i)$;
20: Build a regression model R_{ik} for $(g_{i+1}, \cdots, g_k)$;
21: ϵ_i = total residue of R_{1i};
22: ϵ_k = total residue of R_{ik};
Cand $\leftarrow$ **Cand** $\cup \{(i, \epsilon_i + \epsilon_k)\}$;
23: **end for**
24: $p = \arg min_i\{\epsilon \mid (i, \epsilon) \in \mathbf{Cand}\}$;
25: Return $\{(g_1, \cdots, g_p), (g_{p+1}, \cdots, g_k)\}$
26: **end if**

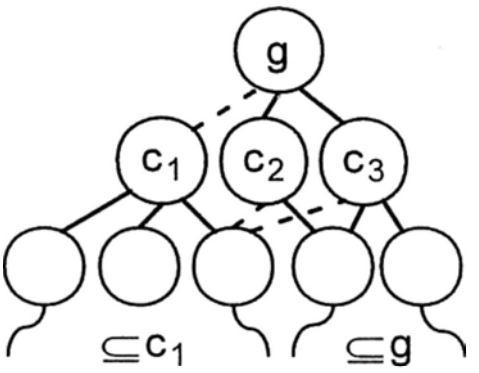


Figure 5.3: Division of Non-Union Template g

indicated in Figure 5.3, the dotted lines indicate some descendant of G_g may be a supergraph of some graph in G_i because of the existence of the edge e, and must be deleted. We build regression models for G_i and G_g, respectively. When building a regression model for a template subgraph (either G_i or G_g), we need to handle a descendant of the template subgraph that has multiple embeddings. Because multiple embeddings lead to different regression equations, we order the edge id set of the template subgraph, and take the embedding with the smallest edge id set as valid.

To compute the edge id set of a template subgraph, logically, we need to determine the edge IDs for the template subgraph, and assign edge IDs for the descendants of the template subgraph repeatedly. The process requires to compute subgraph matching and identify the multiple embeddings level by level in the corresponding POG. The edge id set computed in this way for a subgraph g' in the POG is the minimum among the embeddings of g' to all its ancestors in the POG. We cache the matching information, so the total number of subgraph matching computation is at most m, which is the number of edges of the POG. Upon the edge id set computed for a template subgraph, the regression equations can be formalized, and therefore a regression model can be built according to Eq. (5.6). Among all the possible children of g, we select the child node c_i of g in the POG which results in the minimum sum of residues of the regression models for both G_i and G_g.

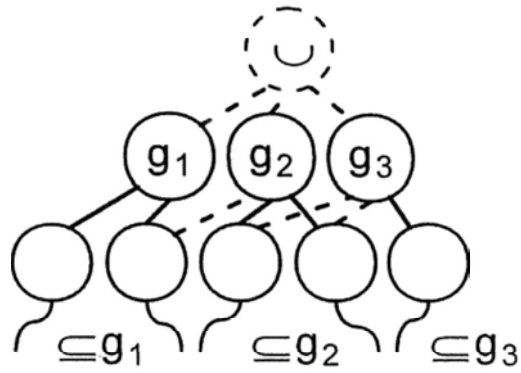


Figure 5.4: Division of Union Template g

Template subgraph is a union of maximal subgraphs

In order to reduce the number of regression models to be built, we introduce union template subgraphs. Conceptually, a union template subgraph is a supergraph containing a set of maximal frequent subgraphs that share some common edges. We use the notion of the union template subgraph to discuss our approach, but we actually do not need to compute the supergraph for a set of template subgraphs.

We first explain how to build a regression model for a union template subgraph g^u. Suppose g^u contains k maximal frequent subgraphs $\{g_1, \cdots, g_k\}$. The corresponding POG is rooted at g^u and has $\{g_1, \cdots, g_k\}$ as the children of g^u. Figure 5.4 shows an example. Suppose the dotted circle indicates a union template subgraph, and g_1, g_2, and g_3 are the maximal frequent subgraphs. Take g^u as a template supergraph, we can build a regression model using the same way as we discussed above. There are two issues. First, we do not know the frequency of the conceptual supergraph g^u whose frequency can be zero. Here, we take the frequency of the maximal frequent subgraphs from $\{g_1, \cdots, g_k\}$. For a subgraph g, suppose its embeddings in g_i is the smallest one, then the regression equation in Eq. (5.6) is

$$\log f(g_i) = \log f(g) + \sum_{j=1}^{l} \log p(e_j), \tag{5.10}$$

where $e_1, \cdots, e_j$ are the edges that appear in g_i but does not in the embedding

of g. Second, we need to order edge IDs for g^u and therefore for all $\{g_1, \cdots, g_k\}$ and their descendants, in order to reduce the number of regression equations. We order the edge IDs as follows. We arrange all $\{g_1, \cdots, g_k\}$ in a certain order, which we will discuss below. Based on the order of all maximal frequent subgraphs, we order edge IDs in a pairwise fashion, (g_i, g_{i+1}). Here, we determine the largest subgraph shared by g_i and g_{i+1} (the highest common descendant of g_i and g_{i+1} in the POG), and assign the same IDs used in g_i for the shared largest subgraph for those in g_{i+1}. We repeat this procedure one-by-one for all the maximal frequent subgraphs under g^u. This is important to note that there may exist several different largest subgraphs shared by both g_i and g_{i+1}. But, there exist new multiple embedding problems if we use all of them to order edge IDs. Consider Figure 5.5. Let g_i be the top-left graph, and g_{i+1} be the top-right graph. g_i and g_{i+1} have two shared subgraphs. Both shared subgraphs contain an edge (b, c) with the same id e_2. If we use all possible shared subgraphs to order IDs, we may end up new multiple embedding problems. When we use only one shared subgraph, since all multiple embeddings have been handled already, we do not have any new multiple embedding problems.

There are several ways to arrange all $\{g_1, g_2, \cdots, g_k\}$ under g^u in an order. In addition to a random order, we consider arranging all $\{g_1, g_2, \cdots, g_k\}$ in an ascending order based on their sizes $|g_i|$, measured by the number of edges, in the following way. First, We add the smallest subgraph to the selected set. Next, we select next subgraph as the smallest one that has repeated sub-structure with one subgraph in the selected set. The reason is that the restoration error for a small template subgraph can be small. Given two template subgraphs g_i and g_{i+1}, for $|g_i| \leq |g_{i+1}|$, we will remove some repeated sub-structure part that appear in g_i from g_{i+1}. Hence, the new g_{i+1} becomes smaller, and it may help to reduce the errors when using g_{i+1}.

As shown in our algorithms, initially, we have one union template subgraph representing all the maximal frequent subgraphs $(g_1, g_2, \cdots, g_k)$ in an

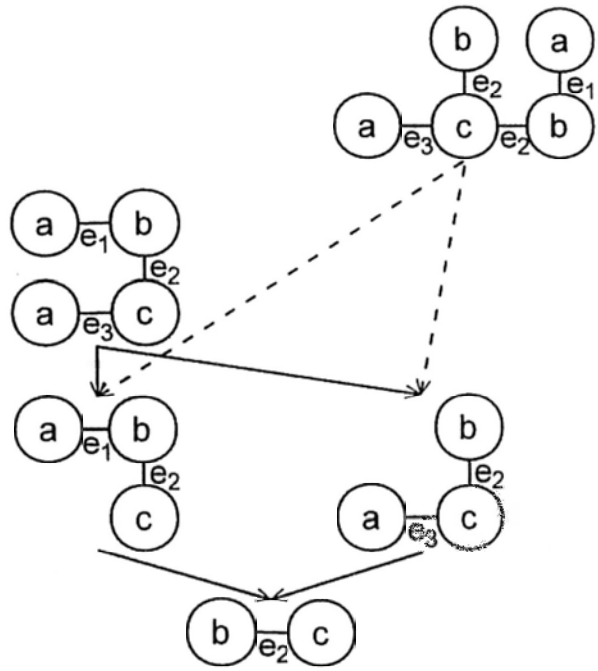


Figure 5.5: An Example of ID Assignment Conflict

order. Then, we select g_i which will minimize the sum of the residue of the left regression model (for $(g_1, \cdots, g_i)$) and one of the right regression models (for $(g_{i+1}, \cdots, g_k)$). Then, we can repeat the same procedure for both $(g_1, \cdots, g_i)$ and $(g_{i+1}, \cdots, g_k)$, respectively, until all regression models are built for all required frequent subgraphs whose restoration errors are $\leq \sigma$, as shown in Algorithm 5.1.

Algorithm Correctness

One thing worth noting is that when transforming the nonlinear regression model to a linear one in Section 5.1, the criterion of optimality is changed. The transformed linear regression model is estimated based on least square estimation, while the optimality is to minimize the average relative restoration error. The consequent question is that whether our summarization framework is correct.

Let $f(g)$ and $r(g)$ be the true and the restored frequency of subgraph g, respectively. The least square error for the linear regression model in Eq. (5.7)

is

$$\begin{aligned} E'E &= (Y - X\beta)'(Y - X\beta) \\ &= \sum_{j=1}^{n}(y_j - x_j\hat{\beta})^2 \\ &= \sum_{j=1}^{n}(\log\frac{f(g^m)}{f(g_j)} - \log\frac{f(g^m)}{r(g_j)})^2 \\ &= \sum_{j=1}^{n}\log^2(\frac{r(g_j)}{f(g_j)}). \end{aligned} \tag{5.11}$$

Following the proof in [27], we have

$$\log^2(\frac{r(g_j)}{f(g_j)}) \to 0 \Leftrightarrow \frac{r(g_j)}{f(g_j)} \to 1 \Leftrightarrow |1 - \frac{r(g_j) - f(g_j)}{f(g_j)}| \to 0. \tag{5.12}$$

As we can see from Eq. (5.12), when the sum of least square errors of a regression model approaches to 0, the relative restoration error approaches 0. We now prove that after each division, the sum of the least square errors of new regression models is less than or equal to the sum of least square errors of the regression model before division.

Let $Y = X\beta$ denote the regression model of the template graph to be divided, where $Y \in \mathbb{R}^n$, $X \in \mathbb{R}^{n\times m}$, and $\beta \in \mathbb{R}^m$ with $n \gg m$. Now we divide the frequent subgraphs, which are reachable from the template graph in the POG, into two sets with n_1 and n_2 subgraphs in each set ($n = n_1 + n_2$), then we have

$$X = \begin{pmatrix} X_1 \\ X_2 \end{pmatrix}, Y = \begin{pmatrix} Y_1 \\ Y_2 \end{pmatrix}, \tag{5.13}$$

where $X_1 \in \mathbb{R}^{n_1\times m}$, $X_2 \in \mathbb{R}^{n_2\times m}$, $Y_1 \in \mathbb{R}^{n_1}$ and $Y_2 \in \mathbb{R}^{n_2}$. Then the least square errors of these three regression models are

$$\delta = \min_{\beta}\Big\{f(\beta) \triangleq (Y - X\beta)'(Y - X\beta)\Big\}, \tag{5.14}$$

$$\delta_1 = \min_{\beta}\Big\{f_1(\beta) \triangleq (Y_1 - X_1\beta)'(Y_1 - X_1\beta)\Big\}, \tag{5.15}$$

$$\delta_2 = \min_{\beta}\Big\{f_2(\beta) \triangleq (Y_2 - X_2\beta)'(Y_2 - X_2\beta)\Big\}, \tag{5.16}$$

and the corresponding optimal solutions are

$$\hat{\beta} = \arg\min_{\beta} \Big\{ (Y - X\beta)'(Y - X\beta) \Big\}, \tag{5.17}$$

$$\hat{\beta}_1 = \arg\min_{\beta} \Big\{ (Y_1 - X_1\beta)'(Y_1 - X_1\beta) \Big\}, \tag{5.18}$$

$$\hat{\beta}_2 = \arg\min_{\beta} \Big\{ (Y_2 - X_2\beta)'(Y_2 - X_2\beta) \Big\}. \tag{5.19}$$

The following theorem is true.

Theorem 5.2. $\delta_1 + \delta_2 \leq \delta$.

Proof. Since $\hat{\beta}_1$, $\hat{\beta}_2$ are the minimizer of problem in Eq. (5.15) and (5.16), respectively, we have

$$\delta_1 = f_1(\hat{\beta}_1) \leq f_1(\hat{\beta}), \text{ and } \delta_2 = f_1(\hat{\beta}_1) \leq f_2(\hat{\beta}). \tag{5.20}$$

Then we only have to show that $\delta = f_1(\hat{\beta}) + f_2(\hat{\beta})$. Solving Eq. (5.14) yields

$$\hat{\beta} = (X'X)^{-1}XY, \text{ and}$$
$$\delta = Y'Y - Y'X(X'X)^{-1}XY.$$

Then substituting $\hat{\beta}$ in $f_1(\beta)$ and $f_2(\beta)$ gives

$$f_i(\hat{\beta}) = Y_i'Y_i + Y'X'(X'X)^{-1}(X_i'X_i)(X'X)^{-1}XY - 2Y_i'X_i(X'X)^{-1}XY, \quad i = 1, 2.$$

Note that

$$Y'Y = Y_1'Y_1 + Y_2'Y_2,$$
$$X'X = X_1'X_1 + X_2'X_2,$$
$$Y'X = Y_1'X + Y_2'X.$$

We have $f_1(\hat{\beta}) + f_2(\hat{\beta}) = \delta$, which completes the proof. □

At Line 6 in Algorithm 5.2, each time we obtain a new template subgraph, we always assign the interaction part to one template with fewer edges. Based

on Theorem 5.2, the sum of the residue of the regression models decrease after division, which means the average least square error decreases. So at least one of the average least square errors of the two regression models is less than or equal to the average error before division. Similar case happens in the division of union template subgraphs. As the division continues, the residues and the average relative restoration errors will decrease. Eventually, algorithm will stop when the average restoration error $\leq \sigma$. Our experimental results demonstrate that restoration errors are decreasing as the number of templates increases.

5.3. Queriable Summarization

Given our frequent subgraph summarization with restoration error control, we can provide an answer when a user wants to know the frequency of a frequent subgraph. Since every frequent subgraph in the POG belongs to one and only one template subgraph, in order to tell the frequency of a frequent subgraph q, we only need to know which template subgraph it belongs to and which embedding in a single template subgraph it needs to use.

In our summarization framework, there is a partial order among edges to avoid the problems caused by multiple embeddings in a template graph. Upon all the template subgraphs obtained, we can identify which template subgraph a query graph q belongs by utilizing the global edge IDs. Once the global edge IDs are all fixed, a frequent subgraph q belongs to the template subgraph in which the edge ID set of the embedding is smallest.

It is worth noting that we discuss the global edge id set when handling the union template subgraph for maximal frequent graphs. The main issue then is how to reduce the number of regression equations. Another aspect of the global edge id set is to determine which template subgraph a given query graph needs to use. Recall that we use the embedding of a subgraph with the small IDs. Based on the order of regression models computed for the template subgraphs, we need to reorder the global edge IDs to determine the right template subgraph

or the right regression model to use. In doing so, suppose that the regression models are built in the following order for $(g_1, g_2, \cdots g_k)$, we simply reorder edge IDs in a way that the edge IDs used in g_{i+1} must be greater than those edge IDs used in g_i.

5.4. Experimental Evaluation

In this section, we demonstrate the performance of our proposed summarization framework. The algorithms are implemented using matlab and C++. All the experiments were run on a server with 4 CPU and 24GB memory running GNU/Linux. One thing worthy noting is that we did not optimize our sources for multiple CPU environment, while matlab sometime utilizes more than one CPU to do matrix computation.

5.4.1. Datasets

We use three datasets: two real datasets and a synthetic one. The real datasets are the AIDS antiviral screen compound dataset[1] from Developmental Theroapeutics Program in NCI/NIH. In the current released version, there are total 43850 chemical compounds, which are classified into three categories: Confirmed Active (CA); Confirmed Moderately active (CM); and Confirmed Inactive (CI). We use CA and CM in our experiments. CA contains 463 compounds and CM contains 1093 compounds. We use the approach in [30], which is a popular frequent subgraph generator in the frequent subgraph mining area, to generate the synthetic data. There are six parameters that the generator takes as input: number of graphs (D), average size of graphs (T), number of frequent patterns as possible frequent graphs (L), average size of frequent patterns (I), distinct edge labels (E) and distinct node labels (V). We generated 5,000 graphs with average size of 20. Other parameters are as follows. The size of seed frequent

[1] `http://dtp.nci.nih.gov/docs/aids/aids_data.html`

Table 5.1: The Number of Frequent Subgraphs in Datasets

CA				
support	11%	12%	13%	14%
The number of frequent subgraphs	15231	14318	8094	7612
CM				
support	6%	7%	8%	10%
The number of frequent subgraphs	5997	4265	3415	2627
D5000T20L200I10E1V10				
minimum frequency f_{min}	270	280	290	300
The number of frequent subgraphs	12903	8093	2404	1592

subgraph is 10. There are 10 distinct node labels and only 1 edge label. Let D5000T20L200I10E1V10 denote the synthetic dataset.

We present the number of frequent subgraphs of each real and synthetic dataset in Table 5.1 for various settings which we use in the following experiments. For CA and DM, we report their number of frequent subgraphs in terms of different values of support and for D5000T20L200I10E1V10, we report the number of frequent subgraphs in terms of different values of minimum frequency. We observed the following situation during generating the synthetic data. Sometimes, only decreasing a frequency by 1, we found that the number of frequent subgraph outputted by gSpan is more than a hundred times than the original one.

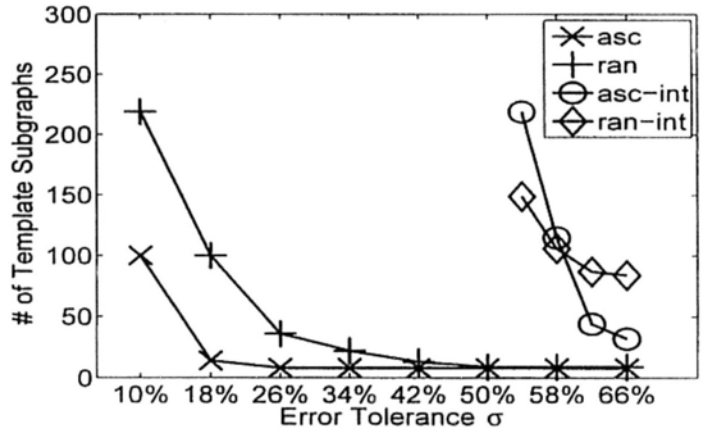
5.4.2. Experiment Setting

We adopt the gSpan [59] algorithm for mining frequent subgraphs. In our summarization framework, the input is a POG of all frequent subgraphs. This can be done by slightly modifying the gSpan algorithm, because the gSpan algorithm generates frequent subgraphs based on their minimum DFS code. Other-

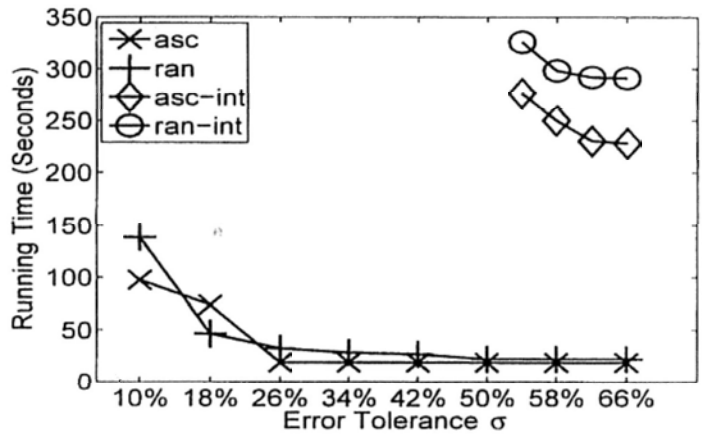
wise, if we are given only a set of frequent subgraphs, we need to call a subgraph matching algorithm to build a POG first, and then apply our approach on it. We measured the performances of four algorithms on these datasets, namely, *asc*, *ran*, *asc-int*, and *ran-int*. All of these algorithms follow the same summarization framework in Algorithm 5.1 in Section 5.2. As discussed, in a union template subgraph, we could arrange the order of maximal frequent subgraphs in different ways to avoid the issues caused by multiple embeddings and common reachable children. *ran* denotes that we arrange the order randomly in division, while *asc* denotes that maximal frequent subgraphs are always sorted in the ascending order of their sizes, measured by number of edges, during the union template creation. As discussed in Section 5.2.1, a frequent subgraph could belong to more than one template subgraph, resulting in it belonging to multiple regression models. In the algorithm *ran* and *asc*, we restrict the intersection part between template subgraphs belong to only one template subgraph. We will relax this restriction to develop two baseline algorithms, i.e., Algorithm *ran-int* and *asc-int*, which are corresponding algorithms to *ran* and *asc*, but allowing a frequent subgraph could belong to more than one template subgraph. That is, every regression model contains its intersection part with other regression models.

5.4.3. Experimental Results

Figure 5.6 and Figure 5.7 reports our results on real dataset CM. We executed our algorithms thoroughly with different combinations of the average restoration error tolerance σ and the minimum support used for mining frequent graphs. We present partial of these results here. Figure 5.6(a) and 5.6(b) show the results when the minimum support of frequent subgraphs is 7%. We tried different values of tolerance, and reported the number of template subgraphs generated. In general, the limitation of no sharing children between template graphs increases both the quality and speed. As we can see, the performances of *asc-int*



(a) Number of template subgraphs vs. error tolerance σ

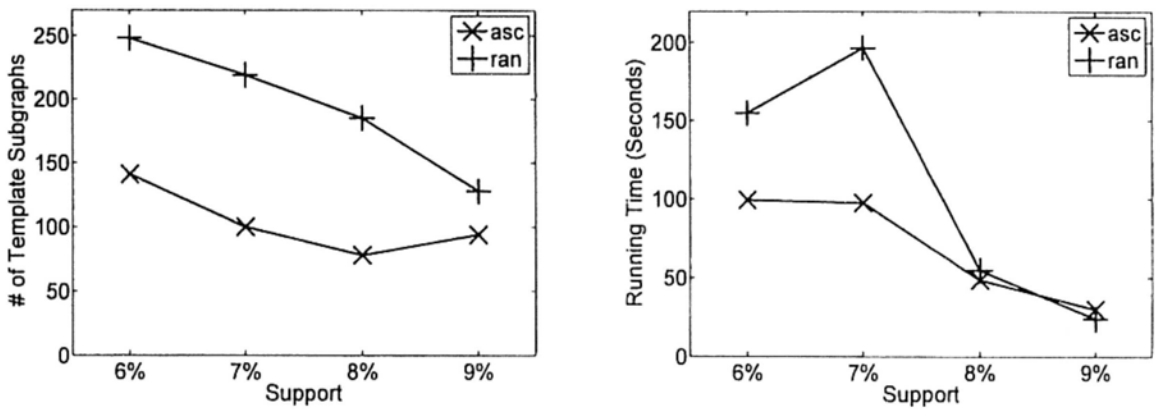


(b) Running time vs. error tolerance σ

Figure 5.6: Experimental Results on Real Dataset CM ($support = 7\%$)

and *ran-int* algorithms, which allow sharing children, are far by worse than ones of *acs* and *ran*. In the executions of our algorithms, we set a global maximum number of template subgraphs generated, that is the reason the curves of *asc-int* and *ran-int* are only shown at the right most part of in the figures. They cannot generate any useful template graph within reasonable time. The number of template graphs generated by *asc* is smaller than ones of *ran* in dataset CM for the same error tolerance. We ran *asc-int* and *ran-int* on all three datasets. Their performances on other datasets are similar to one in Figure 5.6(a) and 5.6(b), so we ignore these two in the following experimental reports.

Figure 5.7(a) and 5.7(b) report the quality and timing results with different



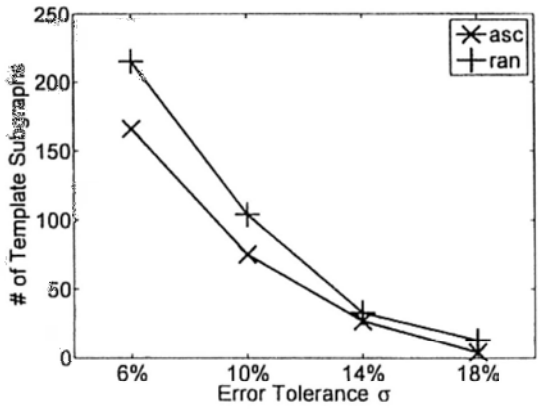
(a) Number of template subgraphs vs. support (b) Running time vs. support

Figure 5.7: Experimental Results on Real Dataset CM ($\sigma = 10\%$)

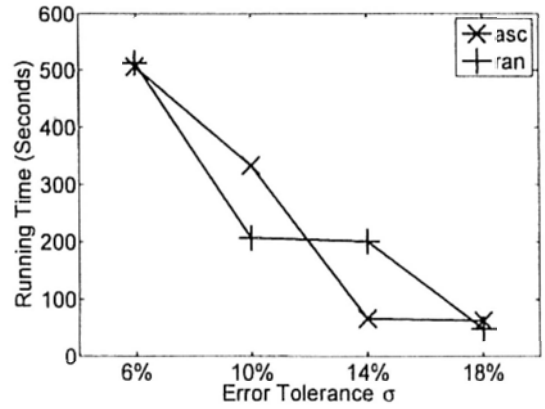
values of support when the tolerance of average restoration error is 10%. Generally, a smaller value of support means a large size of frequent subgraphs. For dataset CM, *asc* works better and when the support becomes smaller, the speed of *acs* is faster than *ran*. While as the number of frequent subgraphs decreases, both the quality and the speed of *acs* are quite close to ones of *ran*.

Figure 5.8 and Figure 5.9 reports our results on real dataset CA. We observed similar results here. Within a set of frequent subgraphs, *acs* can achieve smaller number of template subgraphs with the same bound of tolerance, as indicated in Figure 5.8(a). While for the running time of these two algorithms, *acs* does not always outperform *ran*. In Figure 5.8(b), when the tolerance is 10%, *ran* finished earlier but with a worse quality. With the tolerance set to 15%, in Figure 5.9(a) and 5.9(b), *asc* and *ran* have similar trends to ones in Figure 5.7(a) and 5.7(b). *acs* always generates less template than *ran*. Compared with the number of frequent subgraphs in CM and CA shown in Table 5.1, we can find that the generated template subgraphs are up to 100 times fewer than the frequent subgraphs in both datasets.

In the results of the synthetic dataset, as shown in Figure 5.10 and Figure 5.11, we substitute support by minimum frequency f_{min}. For frequent subgraph with minimum frequency 280, shown in Figure 5.10(a) and 5.10(b), *asc* outper-

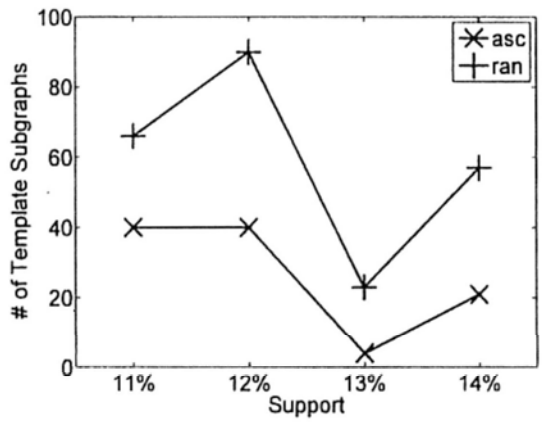


(a) Number of template subgraphs vs. error tolerance σ

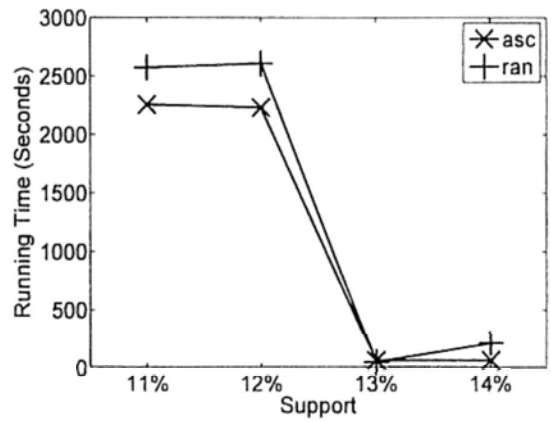


(b) Running time vs. error tolerance σ

Figure 5.8: Experimental Results on Real Dataset CA ($support = 13\%$)



(a) Number of template subgraphs vs. support

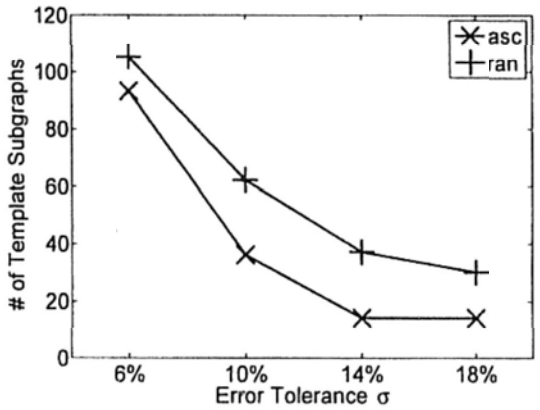


(b) Running time vs. support

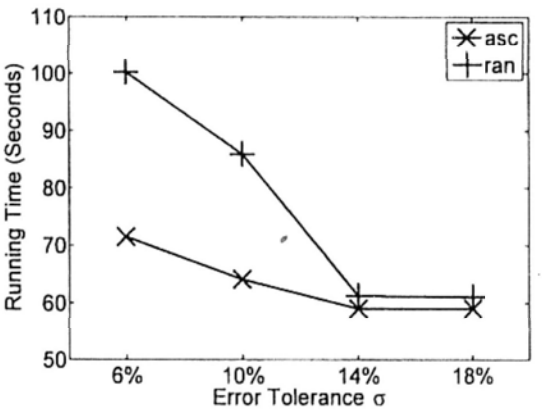
Figure 5.9: Experimental Results on Real Dataset CA ($\sigma = 15\%$)

forms *ran* in various settings of error tolerance and minimum frequency. When we set the error tolerance 10%, *asc* still outperforms *ran* in Figure 5.11(a) and 5.11(b).

In Figure 5.12, the average restoration errors of *asc-int* and *ran-int* are several times larger than ones of *asc* and *ran*. As we can see, the decreasing rate of average restoration error is quite slow and it seems there is not much difference between a small number of template graph and a large number of template graphs. Let us go back to Figure 5.6(a). When the number of template

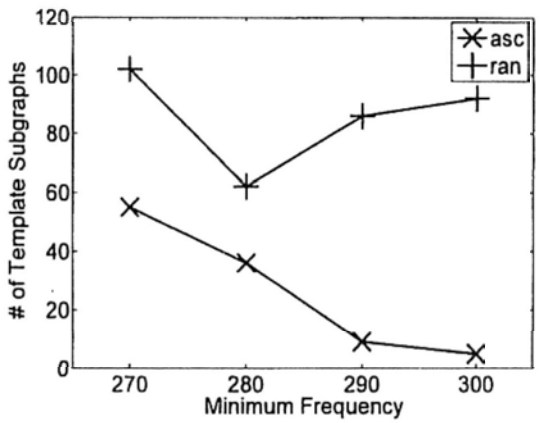


(a) Number of template subgraphs vs. error tolerance σ

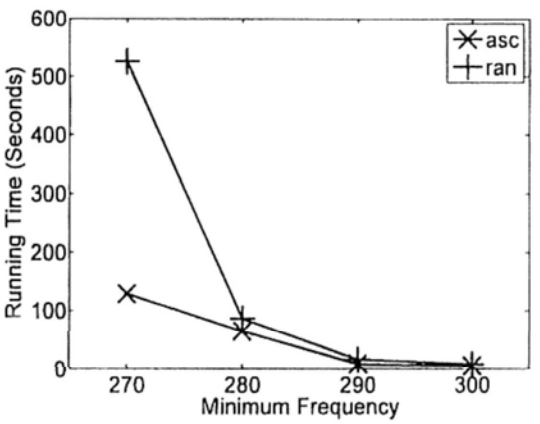


(b) Running time vs. error tolerance σ

Figure 5.10: Experimental Results on Synthetic Dataset D5000T20L200I10E1V10 ($f_{min} = 280$)



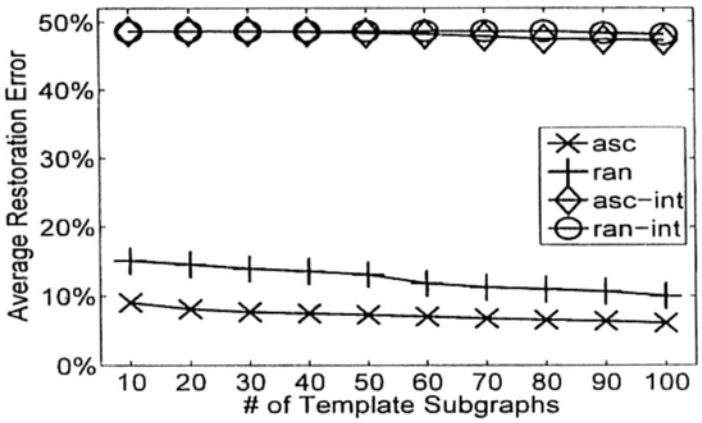
(a) Number of template subgraphs vs. minimum frequency



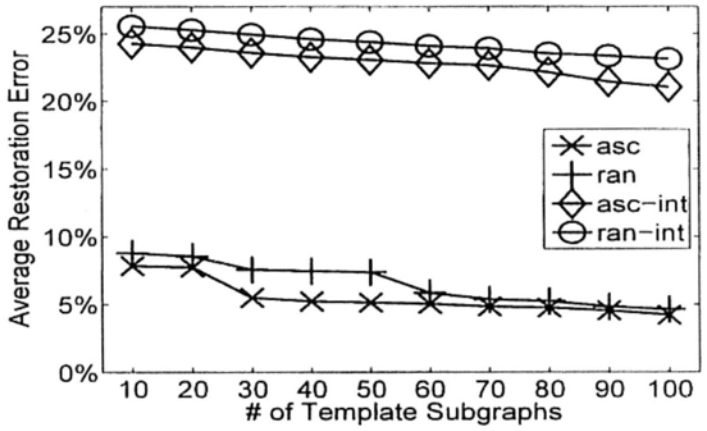
(b) Running time vs. minimum frequency

Figure 5.11: Experimental Results on Synthetic Dataset D5000T20L200I10E1V10 ($\sigma = 10\%$)

graphs equals 10 roughly, the maximum of average restoration error for template graphs is roughly 18%. While in Figure 5.12(a), the average error of all frequent subgraphs is less than 10%. Things are getting worse for *ran*. Based on this argument, we can make sure that the summarization proposed framework can achieve better quality.



(a) CM ($support = 7\%$)



(b) CA ($support = 13\%$)

Figure 5.12: Average Relative Restoration Error of All Frequent Subgraphs

We present the trend of average restoration error of all frequent subgraphs as the number of template graph increases in Figure 5.12. In frequent itemset summarization, there are research works for minimizing the total restoration error given a fixed number of patterns [57, 27]. Similarly, since our summarization framework is in a top-down fashion, we could set a maximum number k of template graphs in our summarization framework, build k regression models and use these k regression models to restore the frequencies of all frequent subgraphs. We report the average value of the relative restoration error of all frequent subgraphs, which is considered as a quality measure of our framework.

Part II

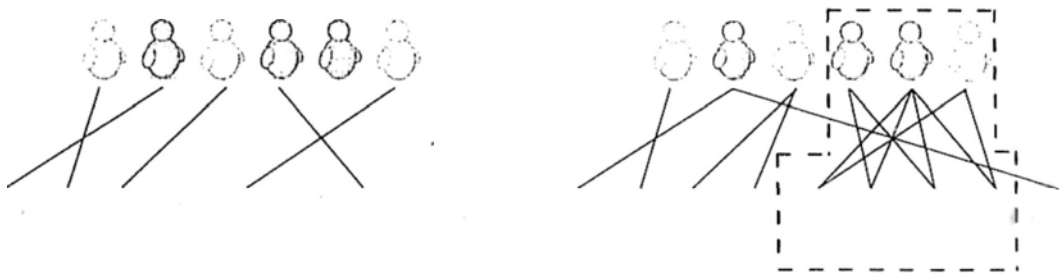
Mining Evolving Graphs

CHAPTER 6

Introduction to Graph Change Detection

Graphs are not always static, and many applications show that graphs evolve over time. In social science, large social networks evolve, which is caused by the node proximity changes [52]. In bioinformatics, finding the co-evolution relationships of structures and functions in structural genomics is an important task to understand evolution progresses [47]. In computer network, traffic jam occurring at a link may affect the traffic routing in a large range. Monitoring the dynamic topology changes and their influences provides network administrators with the insights on network configuration [9]. Also, in wireless sensor networks, query processing is done by exchanging information between sensors whose communication ranges are limited. The fact that a sensor runs out of power, has impacts on the other sensors in terms of network routing, and hence, query processing time. It is important to note, even beforehand, which subgraphs will be affected significantly when such a change occurs.

Generally speaking, graph changes when new nodes/edges join graphs, or old nodes/edges leave graphs. The meaning of changes on graphs could fall into two categories. One is based on raw node/edge evolutions, the other is based on the impact of evolutions to node relationship. Graph change detection aims to find changing areas on graphs when they evolve fast. In this part, we study the



(a) Time t (b) Time $t + \delta t$

Figure 6.1: An Evolving User-Story Graph

problems in the two categories one by one. We start with the introduction to the problems of changing area detection in this chapter, followed by an overview of related works in Chapter 7. The research work on monitoring top k burst areas is presented in Chapter 8 and the research work on spotting significant changing areas is addressed in Chapter 9.

6.1. Monitoring Top k Burst Areas

In most social networks such as Digg [2], one of the common activities among users is making comments on stories. Then a bipartite graph can be constructed by considering users and stories as nodes. There is an edge between a user and a story if the user submits a comment on the story. Let us assume Figure 6.1(a) is a user-story graph at some time t. As time goes by, users submit more comments on stories and the graph evolves. Suppose at time $t + \delta t$, the user-story graph looks like the one shown in Figure 6.1(b). Since both the involvement of users and the popularity of stories are various, the degree of change may be different at each region in the graph. For example, as shown in the dotted line area, this region is much different from the one at time t, while the remaining part looks similar, which means that the users in this region are more active and the stories in it are more attractive.

Inspired by the motivation from Figure 6.1, we study a new problem of discovering the burst areas, which exhibit dramatic changes for a limited period,

in fast graph evolutions. Intuitively, dramatic changes mean the total evolutions happened inside burst areas are much more than the ones in other areas. There are several difficulties of this problem. First, evolving graphs in social networks are huge, which contain a large amount of nodes and edges. Second, the sizes of burst areas could be various. And last, the durations of burst periods are difficult to predict, since a burst could last for minutes, hours, days or even weeks. All these difficulties make this problem challenging and interesting. A candidate solution must be efficient enough to deal with a great number of computations.

We focus on bipartite evolving graphs, since fast evolving graphs in social networks are mostly heterogeneous bipartite graphs. Similar to commenting stories, other possible activities of users could be writing blogs, tagging photos, watching videos, or playing games. Each one of these activities, plus the users and the entities, can form a fast evolving bipartite graph. In an evolving graph, there is a weight associated with each node or edge. The evolutions are in form of the changes of the weights of nodes/edges. The weight of a non-existing node/edge is zero, so it does not matter whether the coming nodes/edges are new to the graph.

The main contributions of this research are summarized below.

- We formalize the problem of discovering burst areas in rapidly evolving graphs. The burst areas are ranked by the total evolutions happened inside and the top k results are returned.

- Instead of calculating the total evolutions of every possible period, we propose to use Haar wavelet trees to maintain the upper bounds of total evolutions for burst areas. We also develop an incremental algorithm to compute the burst areas of different sizes in order to minimize the memory usage.

- We present an evaluation of our proposed approach by using large real data sets and demonstrate that our method is able to find burst areas efficiently.

6.2. Spotting Significant Changing Regions

Let $\mathcal{G}$ be an evolving graph. In this work, we take an edge-centric view regarding changes. We focus on edge changes (deletions/additions) which will cause structural changes. On the other hand, node changes also have impacts on structural changes. But adding isolated nodes before they are connected to any other nodes seems less important, while deleting nodes can be considered as removing edges connected to the deleted nodes.

Given two graphs G_i and G_{i-1} from $\mathcal{G}$ at time t_i and time t_{i-1}, there are many small subgraphs that change while the majority of the graph remains unchanged. A small changing subgraph can be a connected subgraph where every edge is changed (deleted from G_{i-1} or added into G_i), and such a small changing subgraph can be easily identified. However, the influence of a single edge change (deletion/addition) on the other parts of the large graph is more important than the physical change itself. For example, when a researcher A works with another researcher B for a new research issue, A's collaborators and B's collaborators may have new opportunities to work together. Consider the two researchers as two nodes. The newly added edge between them may change the closeness of the nodes that are directly/indirectly connected to the two nodes. Suppose that the closeness of two nodes can be measured. A changing subgraph is an induced subgraph in which the closeness between nodes changes. We focus on the problem of spotting significant changing induced subgraphs in an evolving graph.

A simple large evolving graph $\mathcal{G}$ is illustrated in Figure 6.2, in which only several edges change at a time spot. The upper left subgraph shows a connected subgraph where every edge is changed. The lower left subgraph shows a connected subgraph which involves two changing edges, (v_i, v_j) and (v_k, v_l), and other none-changing edges such as (v_j, v_k) if its two nodes are involved in some changing edges. On the right, it shows a larger connected induced subgraph, as indicated by the dotted area. It includes the changes as well as other parts

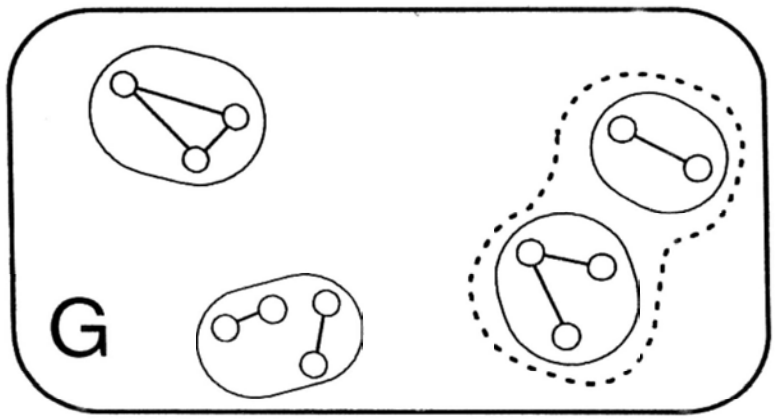


Figure 6.2: An Example of An Evolving Graph

that are significantly influenced by such changes. The issues that we concentrate on in this research include how to measure the closeness changes between two nodes that are caused by some edge changes, how to identify the boundary of the influences of a change, and how to determine a changing subgraph in which changing parts have influences on each other.

Let G_i and G_{i-1} be two graphs in an evolving graph at time t_i and t_{t-1}. In order to find changing subgraphs at time t_i, a possible solution is based on graph distance measures [9]. It is to enumerate all the possible subsets of the node sets of G_i (G_{i-1}), and compute all possible connected induced subgraphs in G_i (G_{i-1}). If the graph distance between two induced subgraphs that include the same nodes is large, then it is considered as a significant changing subgraph. However, this solution is infeasible, since the total number of subsets of the node set is up to 2^n, where n is the number of nodes in the evolving graph. In addition, it may result in many changing subgraphs that are overlapped, which may cause confusion.

The main contributions of this research are summarized below.

- We formalize the problem of spotting significant changing subgraphs in an evolving graph and propose to measure the node closeness with structure information using neighborhood random walks.

- We develop an incremental algorithm to speed up the node closeness computation, as well as a novel strategy about expanding the important node to acquire the connected induced subgraph which can reflect the closeness change between nodes.

- We present an evaluation of our proposed approach by using various large real data sets demonstrating that our method is able to find the suitable subgraph effectively and efficiently.

CHAPTER 7

Related Works

In this chapter, we present an overview the related works to graph change detection, which fall into three categories.

7.1. Community Detection in Static Graphs

The problem of identifying communities in large and sparse graphs has attracted considerable research efforts in literature. Most of the existing studies [11, 19, 17] only handle static graph data. Chakrabarti [11] proposes an approach for parameter-free graph partitioning. The key idea is to measure the encoding cost of a graph. A partition is better if it has smaller encoding cost, which is based on the minimum description length principle. Their algorithm has two steps. In the inner loop, their algorithm minimizes the total encoding cost by reassigning nodes to groups with smaller cost. In the outer loop, the algorithm selects the group which has the largest cost and splits the group into two new groups. These new groups have the minimum total encoding cost among all possibilities.

Gibson et al. [19] study the problem of finding dense subgraph in massive graphs, where graphs are too large to apply traditional clustering techniques. They propose a shingling algorithm to hash all nodes into shingles. Two nodes containing similar sets of shingles are considered to be similar. The procedure of

shingling is conducted more than once to reduce the size of data dramatically. Then, a clustering algorithm is employed to group nodes based on these shingles. Their algorithm is scalable and can handle graphs of billions of edges by using modest amounts of memory space.

Dourisboure et al. [17] also focus on the problem of community discovery in large graphs. The distinct advantage is that their approach can capture partially overlapping communities. The nodes which have small out-degrees are filtered. Then, they propose to mark nodes as centers of communities or fans to communities based on the density of their neighborhoods. At the final step, all communities found are clustering into groups and each group is represented by a set of keywords.

7.2. Community Detection in Evolving Graphs

There are only a few studies [28, 6, 48] that aim to find communities in time-evolving graphs. Such communities are usually dense areas which have many edges and last for a period of time. Kumar et al. [28] propose an approach to discover the bursty communities in a time-evolving graph, which is constructed from web blogs. Their approach consists of two steps. First, all possible communities are extracted, and then, bursty events are detected in a stream of events of each communities. The blog data is collected from seven popular blog sites. Then all people writing these blogs, called bloggers, are considered as nodes. There are links between bloggers if a blogger posts a story which has links to stories posted by the other blogger. Each edge is tagged with a time stamp which is the time when the post is submitted. During the extraction of potential communities, they prune nodes of small degrees. With the extracted communities at different time interval, they identify the relevant events which are represented by keywords and perform burst analysis on these events.

Bansal et al. [6] focus on seeking stable keyword clusters in a keyword graph, which is extracted from a large collection of blog posts and evolves with additions

of blog posts over time. Nodes in a keyword graph are keywords and there is an edge between two keywords if they frequently appear together in documents. Small clusters in the keyword graph, are extracted for each snapshot by removing nodes and edges that are not statistically significant. Clusters are sets of nodes in maximum bi-connected subgraphs. A graph of clusters is further generated by using the generated clusters as nodes. Two clusters are connected by an edges if their time stamps are adjacent. The weight of each edge is how much the keyword sets of these two clusters overlap. Finally, paths with high total weight in the cluster graph are discovered and presented as the sets of persistent keyword clusters. They propose two algorithms to search paths with high total weight in breadth first style and depth first style, respectively. They also extend their approach to a streaming environment for discovering stable keyword clusters online.

Sun et al. [48] propose GraphScope that is able to discover communities in large and dynamic graphs without user-defined parameters. Their approach, which detects communities in each time stamp, is similar to the one in [11]. By using the minimum description length principle, at each time stamp, their algorithm searches the partition of nodes with the minimum lossless encoding cost by repeatedly splitting partition of high cost and merging pair of partitions of low costs. When the encoding cost converges, all partitions at current time stamp are added to a segment. The segment is split if the encoding costs of partitions at two adjacent time stamps differ a lot.

7.3. Node Similarity Based on Random Walks

In terms of distance and similarity measures, the concept of random walk has been widely used to develop various measures that are suitable for different tasks. Jeh and Widom [25] design a measure called SimRank, which defines the similarity between two nodes in a graph based on their neighborhood structures. SimRank between two nodes u and v is essentially the expected meeting distance

of two random walks that start from u and v respectively, and randomly surfer in the graph. They propose an algorithm to compute SimRank values iteratively, until the values of all node pair converge.

Palmer and Faloutsos [44] define a similarity function, named REP, to measure the similarity between categorical attributes. They first convert a categorical dataset into an attribute-attribute graph. Then, REP between two attributes x and y is defined to be the refined escape probability that a random walk starting from x will return to x before reaching y.

There are several studies that utilize random walk with restart [53] in different applications, including automatic captioning for multimedia data [45], and center-piece subgraph discovery [51]. Pan et al. [45] propose to tag photos based on the similarity between photos and keywords. Random walk with restart is introduced for the first time in this work. They construct a hybrid graph where both photos and keywords are viewed as nodes. Two photos are connected if they share some common features, such as color and texture. A photo and a keyword are connected if the photo already has the keyword as a tag. Then those photos that are not tagged are assigned with the keywords of similar photos based on random walk with restart.

Tong et al. [53] present a fast algorithm to calculate the similarity between nodes based on random walk with restart. In order to obtain the converging values of random walk with restart, computation of large inverse matrix is necessary, which is time consuming. Their key idea is to avoid this computation, and use a low rank approximation instead. Tong et al. [51] study the problem of find a subgraph which connects a given set of query nodes. The goodness score between nodes is based on random walk with restart. Individual nodes are connected by key paths, which prefer nodes of high goodness scores.

CHAPTER 8

Discovering Burst Areas in Fast Evolving Graphs

In this chapter, we present our research work on burst area detection in fast evolving graphs. This chapter is organized as follows. Section 8.1 introduces the preliminary background knowledge and formalizes the problem of burst area discovery in evolving graphs. Section 8.2 presents our incremental computation approach, which includes how to maintain the upper bounds of the number of evolutions using Haar wavelet tree, and how to compute burst areas of different sizes incrementally. Section 8.3 reports the experimental results.

8.1. Problem Statement

An evolving graph can be represented as a sequence of graphs, $\mathcal{G} = (G_1, G_2, ...)$. Each graph G_i in the sequence is a snapshot of the evolving graph $\mathcal{G}$ at time t_i. The advantage of this way is that it is convenient for users to study the characteristics of an evolving graph at a particular time stamp, as well as the differences between graphs of adjacent time stamps. One issue of this approach is the large storage cost in proportion to both the size of the evolving graph and the time intervals between snapshots. Another approach models an evolving graph as an initial graph, which is optional, and a stream of graph evolutions. This approach is more appropriate in domains where graphs are evolving fast.

For example, the interactive activities in social networks can be considered as an evolution stream. In this chapter, we model evolving graphs using the second approach since we are more interested in the burst areas in fast graph evolutions.

An evolving graph $\mathcal{G} = (G, \Delta)$ consists of two parts, an initial graph G and a sequence of evolutions Δ. The initial graph G is a snapshot of the evolving graph at time t_0 with a set of nodes $V(G)$ and a set of edges $E(G)$. Let w_i denote the weight of node $v_i \in V$ and w_{ij} denote the weight of edge $e_{ij} = (v_i, v_j)$. Each item δ_t in the evolution stream Δ is a set of quantities indicating the weight changes of nodes or edges at the time t. There might be a number of evolutions at the same time. Let δ_t^i and δ_t^{ij} denote the weight change of node v_i and edge e_{ij} at time t, respectively. Without loss of generality, we assume the evolutions come periodically.

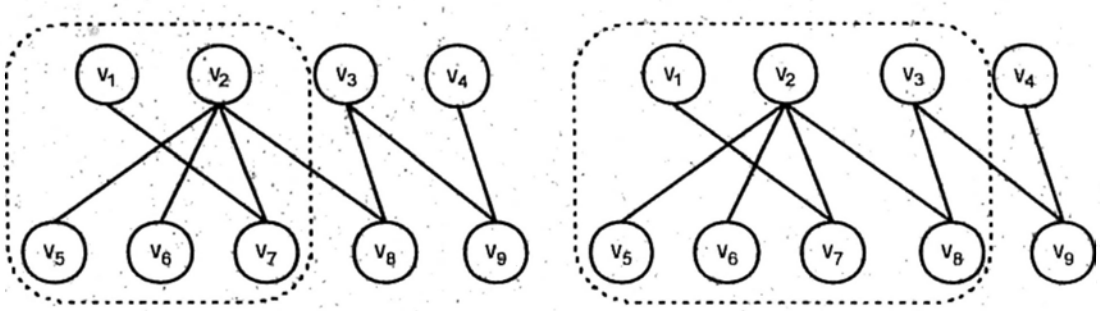
Given a large evolving graph $\mathcal{G} = (G, \Delta)$, we study the problem of finding burst areas. Since a burst area is actually a connected subgraph of the evolving graph, any connected subgraph might be a possible burst area. Apparently, it is more likely that the total evolution in a subgraph with many nodes/edges is more than the one in a subgraph with fewer nodes/edges. Thus, it is insignificant to compare total evolutions among subgraphs that have large differences in terms of node/edge quantity. Consequently, we introduce the r-radius subgraph, which is more meaningful and challenging.

For a given node v_i in a graph, the eccentricity EC of v_i is the maximum length of the shortest paths between v_i and all other nodes in the graph. Based on the definition of eccentricity, the r-radius subgraph is defined as below.

Definition 8.1. r-Radius Subgraph. A subgraph $g = (V(g), E(g))$ in a graph G is an r-radius subgraph, if

$$\min_{v_i \in V(g)} EC(v_i) = r. \tag{8.1}$$

The r-radius subgraphs in a large graph may be overlapping, which leads to redundancy. To avoid this, we introduce the concept of maximal r-radius subgraph.



(a) An r-Radius Subgraph (b) A Maximal r-Radius Subgraph

Figure 8.1: An Example of r-Radius Subgraph

Definition 8.2. Maximal r-Radius Subgraph. An r-radius subgraph g^r is a maximal r-radius subgraph if there exists no other r-radius subgraph $g'^r \subseteq G$, which contains g^r.

Figure 8.1 shows an example of r-radius subgraph and maximal r-radius subgraph. Suppose Figure 8.1(a) and Figure 8.1(b) demonstrate the same graph G. The subgraph in the dotted line area in Figure 8.1(a) is a 2-radius subgraph. It is contained by the subgraph in the dotted line area in Figure 8.1(b), which is a maximal r-radius subgraph. It is a difficult task to identify all maximal r-radius subgraphs in a large evolving graph. We observe that a maximal r-radius subgraph is in fact a maximal r-hop neighborhood subgraph, which we define below. Let $N^r_{v_i}$ denote the set of nodes (except v_i) whose shortest path distances to v_i are less than or equal to r.

Definition 8.3. r-Hop Neighborhood Subgraph. An r-hop neighborhood subgraph $g^r_{v_i}$ in a graph G is defined as the induced subgraph of G containing all the nodes in $N^r_{v_i}$. v_i is called the center of $g^r_{v_i}$.

In the following of this chapter, we use r-hop subgraph for short. We characterize the relationship between maximal r-radius subgraphs and r-hop neighborhood subgraphs in Theorem 8.1.

Theorem 8.1. *For each maximal r-radius subgraph $g^r \subseteq G$, there is a corresponding r-hop neighborhood subgraph $g^r_{v_i} \subseteq G$, and $g^r = g^r_{v_i}$.*

Proof Sketch: We will prove that (1) $\exists v_i \in g^r$, $g^r \subseteq g^r_{v_i}$, and (2) $\nexists v_j \in g^r_{v_i}$, $v_j \notin g^r$.

Let g^r be a maximal r-radius subgraph belonging to G. Recall that the eccentricity $EC(v_i)$ is the maximum length of the shortest paths between v_i and all other nodes in g^r. Then based on Definition 8.1, there exists a node $v_i \in V(g^r)$ and $EC(v_i) = r$. Let $d(v_i, v_j)$ denote the length of the shortest path between v_i and v_j. Because $EC(v_i) = r$, so $\forall v_j \in V(g^r)$, we have $d(v_i, v_j) \leq r$. Let $g^r_{v_i}$ be an r-hop neighborhood subgraph based on Definition 8.3. Since $\forall v_j \in V(g^r), d(v_i, v_j) \leq r$, so $\forall v_j \in V(g^r)$, $v_j \in N^r_{v_i}$, where $N^r_{v_i} = V(g^r_{v_i})$. Therefore, $\exists v_i \in g^r$, $g^r \subseteq g^r_{v_i}$.

Suppose $\exists v_k \in g^r_{v_i}$ and $v_k \notin g^r$, we have $d(v_i, v_k) \leq r$. Let g' denote the subgraph that $V(g') = V(g^r) \cup \{v_k\}$, then g' is also an r-radius subgraph, which is contradict to the condition that g^r is a maximal r-radius subgraph. Therefore, $\nexists v_j \in g^r_{v_i}$, $v_j \notin g^r$. □

Theorem 8.1 indicates that a maximal r-radius subgraph must be an r-hop neighborhood subgraph. This is only a necessary condition, not sufficient. It is worth noting that an r-hop neighborhood subgraph might not be an r-radius subgraph. Take node v_4 in Figure 8.1(a) as an example. Let us construct a 2-hop neighborhood subgraph $g^2_{v_4}$, which contains node v_3, v_4 and v_9. $g^2_{v_4}$ is not a maximal 2-radius subgraph, but a maximal 1-radius subgraph $g^1_{v_9}$, because $d(v_3, v_9) = d(v_4, v_9) = 1$. We consider r-hop neighborhood subgraphs as the candidates of burst areas.

Recall δ^i_t and δ^{ij}_t denote the weight changes of node v_i and edge e_{ij} at time t, respectively. Obviously, if $v_i \in N^r_{v_j}$, which means v_i is in the r-hop neighborhood subgraph $g^r_{v_j}$, then δ^i_t should be counted in $g^r_{v_i}$. δ^{ij}_t belongs to an r-hop neighborhood subgraph when both v_i and v_j are in the subgraph. We define the burst score of an r-hop neighborhood subgraph as follows.

Definition 8.4. Burst Score. The node burst score of an r-hop neighborhood

subgraph $g^r_{v_i}$ at time t is the total weight of the node evolutions happened inside.

$$BurstScore^V = \sum_{v_j \in g^r_{v_i}} \delta^j_t. \tag{8.2}$$

The edge burst score of an r-hop neighborhood subgraph $g^r_{v_i}$ at time t is the total weight of the edge evolutions happened inside.

$$BurstScore^E = \sum_{v_j, v_k \in g^r_{v_i}} \delta^{jk}_t. \tag{8.3}$$

So, the burst score of an r-hop neighborhood subgraph $g^r_{v_i}$ is the sum of the node burst score and the edge burst score.

Now, we formally define the problem of discovering top k burst areas in fast evolving graphs.

Problem 8.1. Discovering Top k Burst Areas. For an evolving graph $\mathcal{G} = (G, \Delta)$, given a maximum hop size r_{max}, a burst window range (l_{min}, l_{max}), the top k burst area discovery problem is that for each burst window size between l_{min} and l_{max} and each hop size between 1 and r_{max}, finding the top k r-hop neighborhood subgraphs with the highest burst scores in $\mathcal{G}$ at each time stamp continuously.

For conciseness, in the following, we focus on edge evolutions in heterogeneous bipartite evolving graphs. Our proposed solution can deal with node evolutions as well.

8.2. Discovering Burst Areas

A direct solution to discover top k burst areas would be maintaining total $(l_{max} - l_{min} + 1) \times r$ burst scores of each window size and each hop size over sliding windows. At each time stamp t, these burst scores are updated based on the evolutions happened inside the corresponding r-hop neighborhood subgraphs. Then, for each window size and each hop size, a list of top k r-hop subgraphs

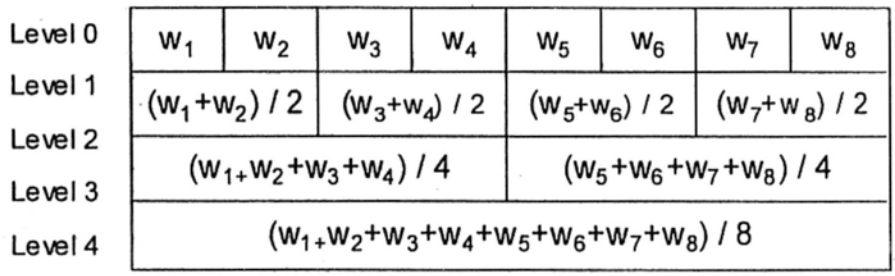


Figure 8.2: Haar Wavelet Decomposition

based on the burst scores is returned as the answer. Before we explain our proposed solution in details, which is much more efficient in both time complexity and memory consumption, we introduce some background knowledge first.

8.2.1. Haar Wavelet Decomposition

The wavelet decomposition is widely used in various domains, especially in signal processing. One of the conceptually simplest wavelet, Haar wavelet, is applied to compress time series and speed up the similarity search in time series database. The Haar wavelet decomposition of a time series is done by averaging two adjacent values on the time series repeatedly in multiple resolutions in a hierarchical structure, called Haar wavelet tree. The hierarchical structure can be constructed in $O(n)$ time. Figure 8.2 illustrates how to construct the Haar wavelet tree[1] of an eight-value time series, which is shown at Level 0. Then at Level 1, there are four average values of adjacent values in Level 0. The averaging process is performed repeatedly until there is only one average value left.

[1]The Haar wavelet decomposition consists of both averages and differences. For conciseness, we ignore the difference coefficients which are not used in our solution.

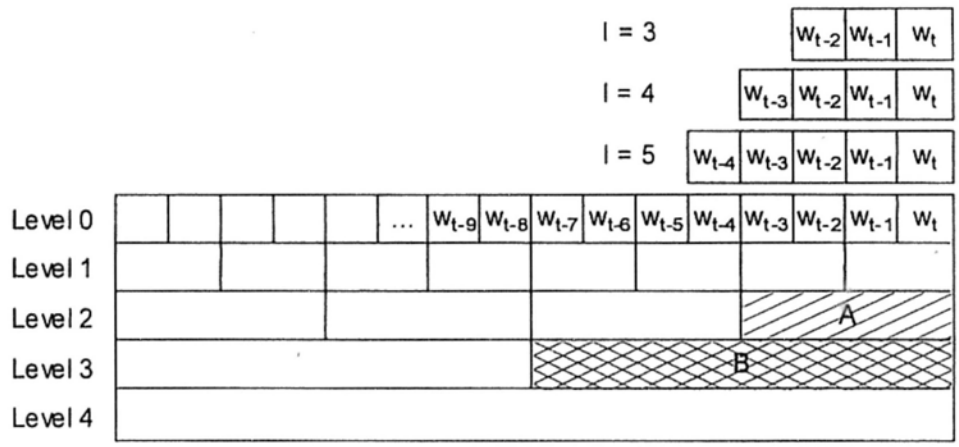


Figure 8.3: Upper Bounds of Burst Scores Using Haar Wavelet Decomposition

8.2.2. Bounding Burst Scores of r-Hop Neighborhood Subgraphs

As defined in Definition 8.4, the burst score of an r-hop neighborhood subgraph is the total weight change happened inside during a period of time. Given an evolving graph $\mathcal{G} = (G, \Delta)$ and a window size range (l_{min}, l_{max}), we introduce first how to bound the burst scores for an r-hop subgraph.

Figure 8.3 shows an example. w_t is the sum of all the weight change happened in an r-hop neighborhood subgraph at time stamp t, Based on the Haar wavelet decomposition, we can construct a Haar wavelet tree as shown at the bottom in Figure 8.3. Suppose $l_{min} = 3$ and $l_{max} = 5$, the three corresponding burst windows are shown at the top. As we can see that, the burst windows of size 3 and 4 are contained by window A at Level 2, while the burst window of size 5 is contained by windows B at Level 3. This leads to the following lemma.

Lemma 8.2. *A burst window of length l at time t is contained in the window at Level $\lceil \log_2 l \rceil$ in the hierarchical Haar wavelet tree at time t.*

Proof Sketch: Let $W = w_{t-l+1}, w_{t-l+2}, ..., w_t$ denote the burst window of length l. Based on the definition of the Haar wavelet decomposition, the length of window at time t at Level n is 2^n. Let $W' = w_t, w_{t-1}, ..., w'_{t-2^{(\lceil \log_2 l \rceil)+1}}$. Because

$2^{(\lceil \log_2 l \rceil)} \geq l$, we have $W \subseteq W'$. □

Instead of average coefficients, we maintain the sums of windows in the Haar wavelet tree. Since the weight changes are all positive, the sum of the values in a window in the Haar wavelet tree is the upper bound of the burst scores of all burst windows it contains, which leads to the following theorem.

Theorem 8.3. *The burst score of a length l burst window at time t is bounded by the sum coefficient of the window at Level $\lceil \log_2 l \rceil$ at time t in the Haar wavelet tree.*

We can use Theorem 8.3 to prune candidate burst areas. If the burst score bound of an r-hop neighborhood subgraph is larger than the minimum score in the current top k burst areas, then we perform a detailed search to check whether it is a true top k burst area. Otherwise, the r-hop subgraph is ignored. It is not necessary to build the whole Haar wavelet tree of all levels to compute burst score bounds of r-hop subgraphs. As we can see from Theorem 8.3, only the levels from $\lceil \log_2^{l_{min}} \rceil - 1$ to $\lceil \log_2^{l_{max}} \rceil$ are needed to compute the burst score bounds. Level $\lceil \log_2^{l_{min}} \rceil - 1$ is directly computed from Level 0.

Now, the problem is how to maintain the wavelet tree at each time stamp t, since graph evolutions come in as a stream. There are mainly two approaches.

1. **Continuous Updating:** The entire Haar wavelet tree is updated at each time stamp t continuously. The approach ensures there is no delay in response time to return top k answers.

2. **Lazy Updating:** Only windows at the lowest level are updated at each time stamp t. The sums maintained in the upper levels in the Haar wavelet tree are not computed until all data in the corresponding windows is available. For a burst window of size l, the response time delays at most $2^{\lceil \log_2 l \rceil}$.

In our solution, we propose to maintain the Haar wavelet trees in a dynamic manner, which can achieve both low computation cost and no delay in response

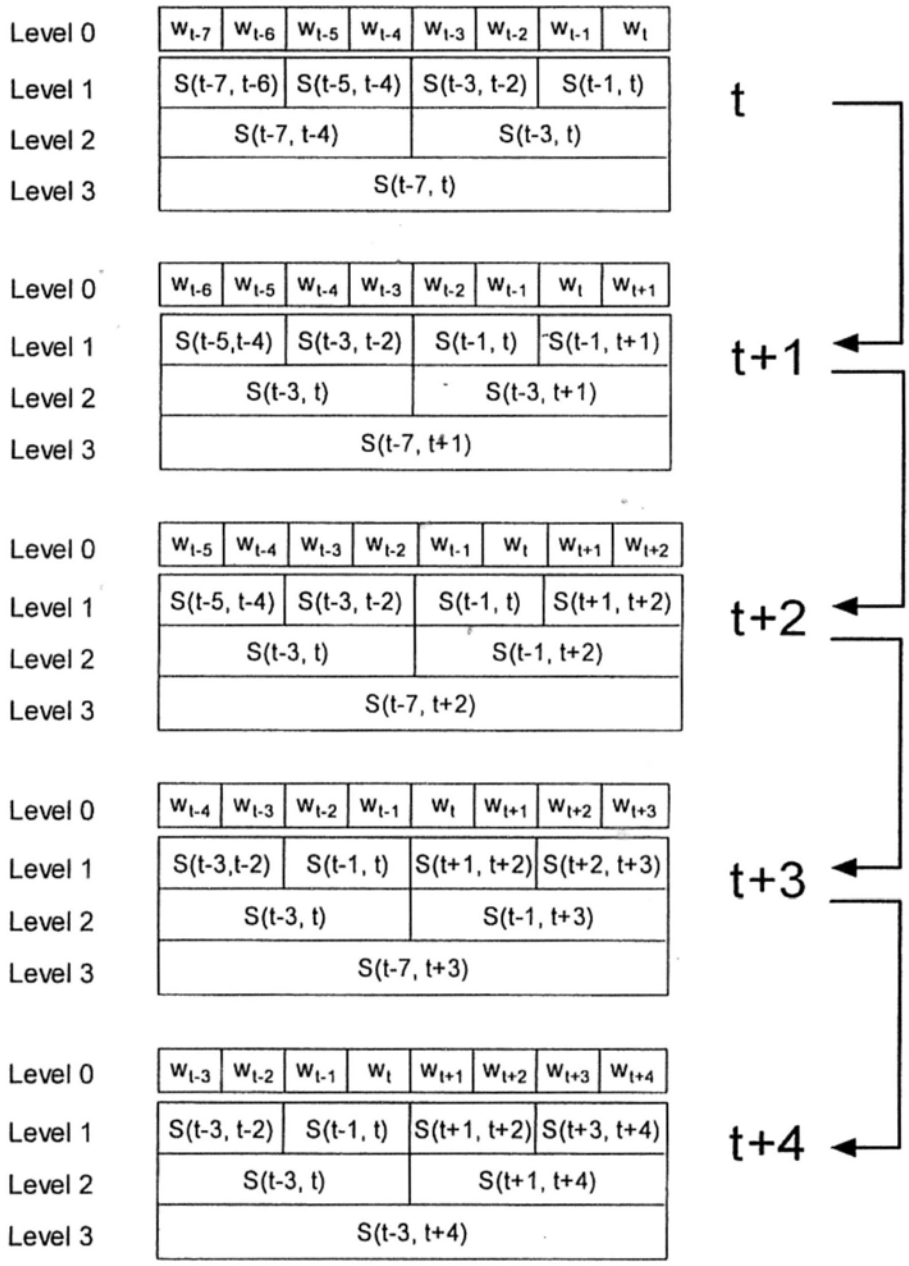
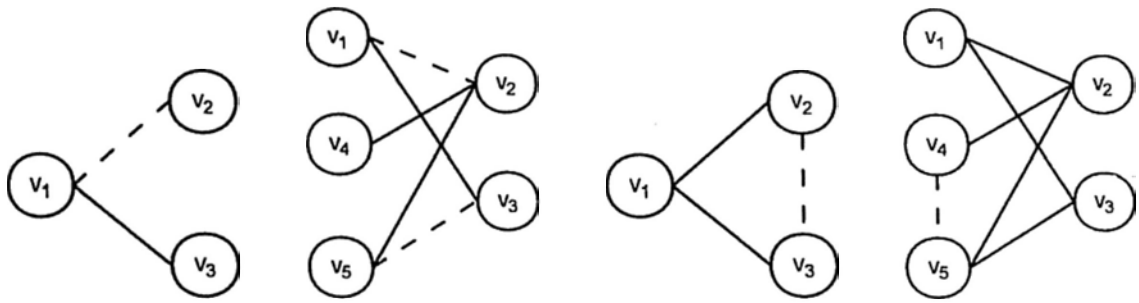


Figure 8.4: Updating Haar Wavelet Tree

time. Figure 8.4 presents a running example, which illustrates how the Haar wavelet tree changes as time goes by. Function $S(t, t')$ denotes the sum of weights in the window from time t to t'.

As shown in Figure 8.4, suppose at time t, a Haar wavelet tree is built according to the weight changes at Level 0. Then at time $t + 1$, instead of updating the entire Haar wavelet tree, we only shift each level left for one window



(a) 1-hop (b) 2-hop (c) 1-hop (d) 2-hop

Figure 8.5: Evolutions in 1-hop and 2-hop Subgraphs

and add the newly weight change w_{t+1} to the last window at each level. Since weight change are all positive, the sums of the last windows are still the upper bounds of burst scores of corresponding burst windows. At time $t+2$, since all weights at Level 0 used to compute the sum of the last window at Level 1 are available, we compute the actual sum of the last window at Level 1 based on Level 0. Then based on the weights at Level 1, the sum of the last window at Level 2 is recomputed. While the last window at Level 3 is not recomputed since the last two windows at Level 2 are overlapping. Instead, we add w_{t+1} to it. Time $t+3$ is similar to time $t+1$. At time $t+4$, the last windows at all levels are recomputed, because the last two windows of lower levels are not overlapping. In general, last window at the lowest level (Level $\lceil \log_2^{l_{min}} \rceil - 1$) is computed every $2^{\lceil \log_2^{l_{min}} \rceil - 1}$ time stamps, while last windows at upper levels are recomputed once the last two windows at lower levels are not overlapping.

8.2.3. Incremental Computation of Multiple Hop Sizes

Suppose we need to monitor r-hop neighborhood subgraphs of different hop sizes, an easy solution is to maintain a Haar wavelet tree for each hop size. The total memory usage would be $O(rN)$, where N is the total number of nodes. Obviously, it is not efficient due to the high computation cost, as well as the large memory assumption. In this section, we introduce our algorithm to maintain burst score bounds of multiple hop sizes using at most $O(N)$ memory consump-

tion. Our solution is to maintain Haar wavelet trees for 1-hop neighborhood subgraphs only. The burst score bounds of an r-hop subgraph is calculated from subgraphs of smaller hop size in an incremental manner.

We first examine how edge evolutions affect the burst scores of nearby r-hop neighborhood subgraphs using examples in Figure 8.5. Figure 8.5(a) shows an example for 1-hop neighborhood subgraph $g^1_{v_1}$. The dotted line is the edge evolution happened. It is apparent that if the edge evolution belongs to $g^1_{v_1}$, v_1 must be one of the nodes of the edge evolution. Figure 8.5(b) shows an example for 2-hop neighborhood subgraph $g^2_{v_1}$. As we can see that if the edge evolution belongs to $g^2_{v_1}$, either it is within $g^1_{v_1}$, or $N^2_{v_1} \setminus \{v_1\}$. An edge evolution belongs $N^2_{v_1} \setminus \{v_1\}$ means both nodes of the edge evolution belong to $N^2_{v_1} \setminus \{v_1\}$. We are focusing on heterogenous bipartite graph. Suppose v_1, v_4, v_5 and v_2, v_3 belong to the two sides of a bipartite graph, respectively. Then there are no such evolutions as shown by the dotted edges in Figure 8.5(c) and 8.5(d).

Now we explain how to compute the burst scores of r-hop neighborhood subgraphs incrementally from the burst scores of 1-hop subgraphs. From the above obversion, we know that the burst score of an r-hop subgraph is the sum of the two parts. One is the burst score of $(r-2)$-hop subgraph, the other is the total evolutions within $N^r_{v_i} \setminus N^{r-2}_{v_i}$. Edge evolutions in $N^r_{v_i} \setminus N^{r-2}_{v_i}$ must be connected to one of the nodes in $N^{r-1}_{v_i} \setminus N^{r-2}_{v_i}$. Let $v_j \in N^{r-1}_{v_i} \setminus N^{r-2}_{v_i}$, then we have the following lemma.

Lemma 8.4. *The total number of edge evolutions in* $N^r_{v_i} \setminus N^{r-2}_{v_i}$ *equals*

$$\sum_{v_j \in N^{r-1}_{v_i} \setminus N^{r-2}_{v_i}} BurstScore^1 v_j. \tag{8.4}$$

Based on Lemma 8.4, the burst scores of r-hop neighborhood subgraphs are calculated incrementally by using the following equation.

$$BurstScore^r_{v_i} = BurstScore^{r-2}_{v_i} + \sum_{v_j \in N^{r-1}_{v_i} \setminus N^{r-2}_{v_i}} BurstSocre^1_{v_j} \tag{8.5}$$

where $BurstScore^r_{v_i}$ denotes the burst score of r-hop neighborhood subgraph $g^r_{v_i}$, and $BurstScore^0_{v_i} = 0$. It is obvious that Eq. 8.5 is also correct if we substitute the upper bounds of the burst areas for them.

8.2.4. Top k Burst Area Discovery

The whole algorithm is presented in Algorithm 8.1. At line 3, the algorithm maintains Haar wavelet trees of all 1-hop neighborhood subgraphs at each time t. If a node is seen for the first time, a new Haar wavelet tree is constructed. Otherwise, based on Section 8.2.2, Algorithm 8.1 updates all the Haar wavelet trees which have evolutions happened inside. In each loop from line 4 to line 12, the algorithm discovers incrementally the top k results from small hop size to large hop size. At line 6, Algorithm 8.1 computes the upper bounds of burst scores according to Eq. 8.5. If the burst score bound of an r-hop neighborhood subgraph is larger than the minimum burst score $mink$ in the corresponding top k list, Algorithm 8.1 performs a detailed search at line 10 to verify whether it is a real top k result. If the true burst score is larger than $mink$, it is added to the corresponding top k list substituting the k-th item. To save memory space, instead of storing r-hop subgraphs, we only store centers of the r-hop subgraphs in the top k list.

8.3. Experimental Evaluation

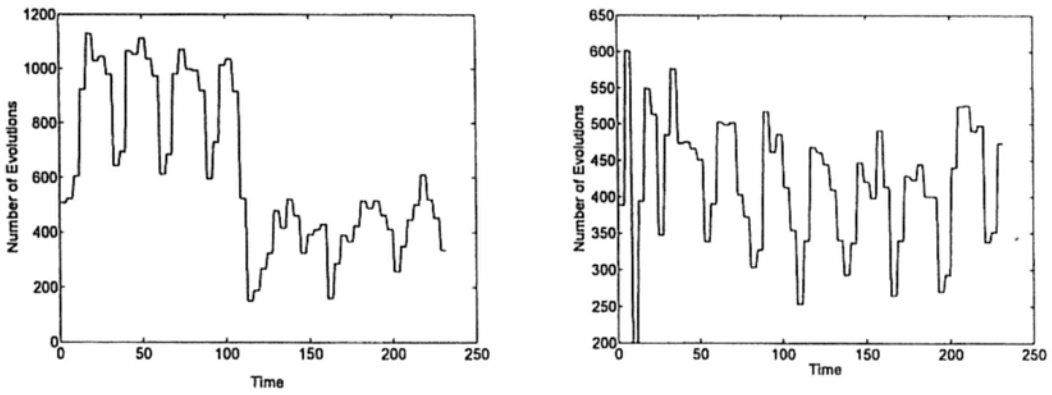
In this section, we report our experimental results on two real datasets to show both the effectiveness and the efficiency of our proposed algorithm.

Algorithm 8.1 The Top k Burst Area Discovery Algorithm

Input: An evolving graph $\mathcal{G} = (G, \Delta)$, a maximal hop size r_{max}, a window range (l_{min}, l_{max}), the value of k

Output: The top k burst areas at each time t

```
1: while time t ≤ t_max do
2:   for v_i ∈ V(G) do
3:     Update the Haar wavelet tree for 1-hop subgraph g^1_{v_i};
4:   end for
5:   for r = 1 to r_max do
6:     for v_i ∈ V(G) do
7:       Compute burst score bound B^r_{v_i} of r-hop subgraphs g^r_{v_i} using Eq. 8.5;
8:       for l = l_min to l_max do
9:         mink = the minimum burst score of the top k list of hop size r and
           window length l;
10:        if B^r_{v_i} > mink then
11:          Obtain BurstScore^r_{v_i} by detailed search;
12:          if BurstScore^r_{v_i} > mink then
13:            remove the k-th node v_j in the corresponding top k list;
14:            add v_i to the corresponding top k list.
15:          end if
16:        end if
17:      end for
18:    end for
19:  end for
20: end while
```



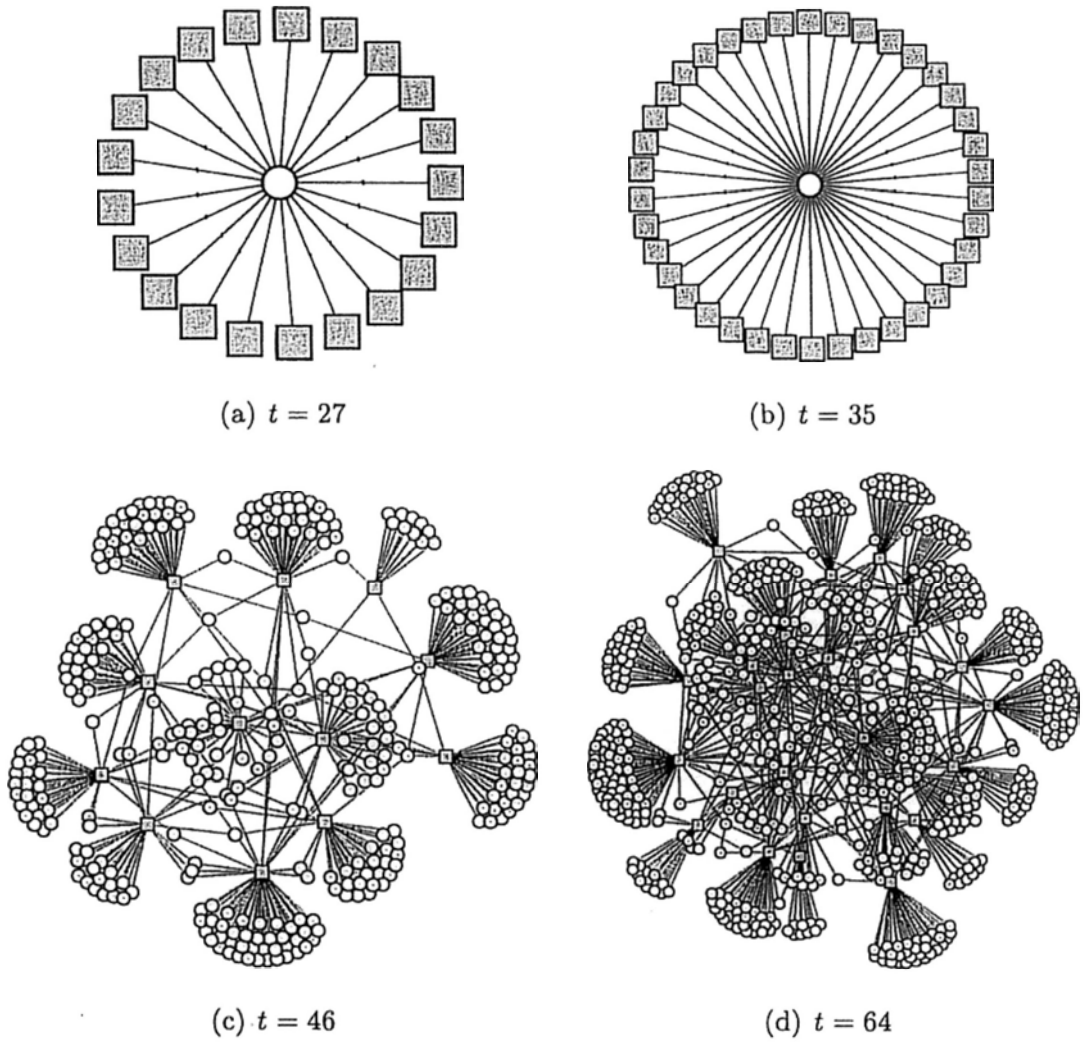
(a) Digg A (b) Digg B

Figure 8.6: Total Evolutions of The Evolving User-Story Graphs from Digg

8.3.1. Diastases

The two real diastases are extracted from Digg [2], where users can make their comments on stories. The nodes of the heterogenous bipartite evolving graph are users and stories. Graph evolutions are the comments submitted by users.

The corpus of users' comments collected contains comments for around four month [35]. For better utilization, we split it into two two-month datasets, Digg A and Digg B. Comments in the datasets are categorized day by day and there are a large number of comments in each day. So, we further divide a day into four time stamps and randomly assigned the comments in the same day into one of the four time stamps. There are total 9583 users and 44005 articles. The total time stamps of both datasets is 232. The evolution characteristics of these two datasets are summarized in Figure 8.6, which shows the total number of evolutions happened at each time stamp. There are periodic troughs in both figures, indicating that users submit fewer comments during weekend.



(a) $t = 27$ (b) $t = 35$

(c) $t = 46$ (d) $t = 64$

Figure 8.7: Top 1 Burst Areas in Digg A ($l = 8$)

8.3.2. Effectiveness

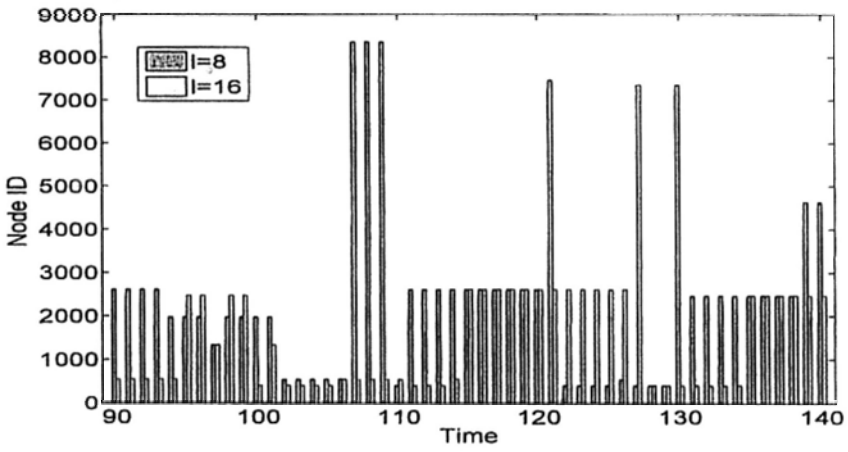
We demonstrate two examples discovered by our proposed algorithm in dataset Digg A in Figure 8.7. We monitor top 10 burst areas for each time stamp, where the length of burst window is 8. Among all these burst areas, the burst area with the highest burst score is reported in Figure 8.7. The 1-hop burst area with the highest score happened at time 35. We present the 1-hop neighborhood subgraph at time 35 in Figure 8.7(b). For reference, the corresponding 1-hop subgraph at time 27 is presented in Figure 8.7(a), which is the subgraph when all the evolutions have not happened. The round nodes and the square

nodes represent users and stories, respectively. The center round nodes in both graphs are the center of the 1-hop burst area. As we can see that the subgraph in Figure 8.7(b) has more nodes and edges than the one in Figure 8.7(a), which indicates that the center node is an active user during the burst period. Similar results could be observed in Figure 8.7(c) and Figure 8.7(d), which show the 2-hop burst area with the highest burst score among all the top 10 burst areas of each time stamp. The burst area happened at time 64 and we also present the same burst area at time 48 for reference. The centers of these two subgraphs are shown as the central white nodes in the figures. These figures show that the stories, which were commented by the user of the center node, also received many comments from other users during the burst period.

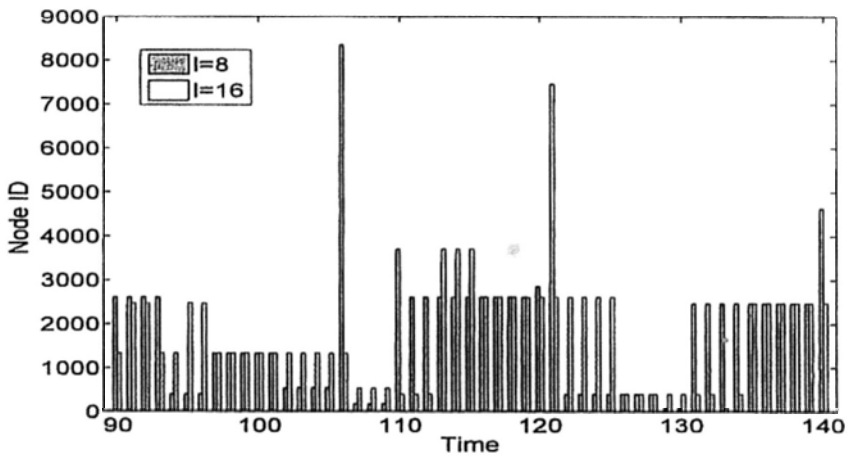
Figure 8.8(a) and Figure 8.8(b) present the center node ID of the top 1 1-hop and 2-hop burst areas from time 90 to time 140, respectively. At each time stamp, we plot the center node ID of top 1 burst area whose burst window length is 8, as well as the one of top 1 burst area whose burst window length is 16. The figures show that the top 1 burst area of large window length is not always the same as one of small window size, which explains why it is necessary to find burst areas of various burst window lengths.

8.3.3. Efficiency

We perform our efficiency experiments on dataset Digg A and Digg B. Figure 8.9(a) and Figure 8.9(b) show the overall running time of the direct algorithm, which is discussed at the beginning in Section 8.1, as well as the one of our proposed algorithm. The value of k in Figure 8.9(a) and 8.9(b) is 10 and 20, respectively. As we can see, our proposed algorithm is much faster than the direct algorithm. The lower part of each bar of the direct algorithm is the running time of updating all burst scores, and the lower part of each bar of our proposed algorithm is the running time of maintaining Haar wavelet trees. One advantage of our proposed algorithm is that the maintaining cost is less than



(a) 1-hop

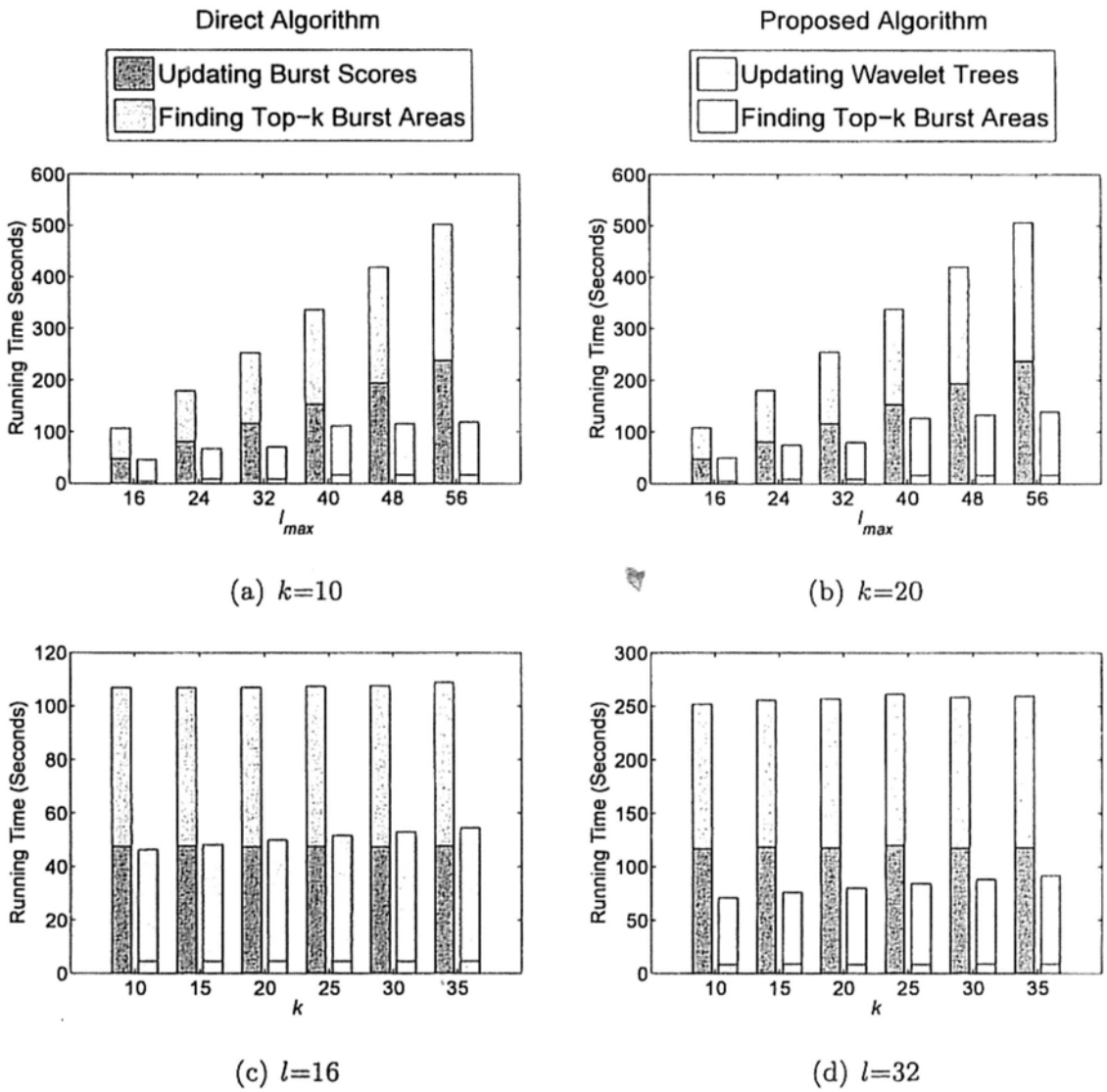


(b) 2-hop

Figure 8.8: Center Node ID of Top 1 Burst Areas

1/10 of the one of the direct algorithm. This will be useful when we do not monitor evolution streams continuously, but submit ad-hoc queries to find top k burst areas at some interesting time stamps.

Figure 8.9(c) and 8.9(d) show the overall running time for the direct algorithm and our proposed algorithm, when the value of k changes. The length l of burst window in Figure 8.9(c) and 8.9(d) is 16 and 32, respectively. We can observe similar experimental results that our proposed algorithm uses much less time. Figure 8.10 presents the corresponding results for dataset Digg B, which prove again the efficiency of our proposed algorithm.



(a) k=10

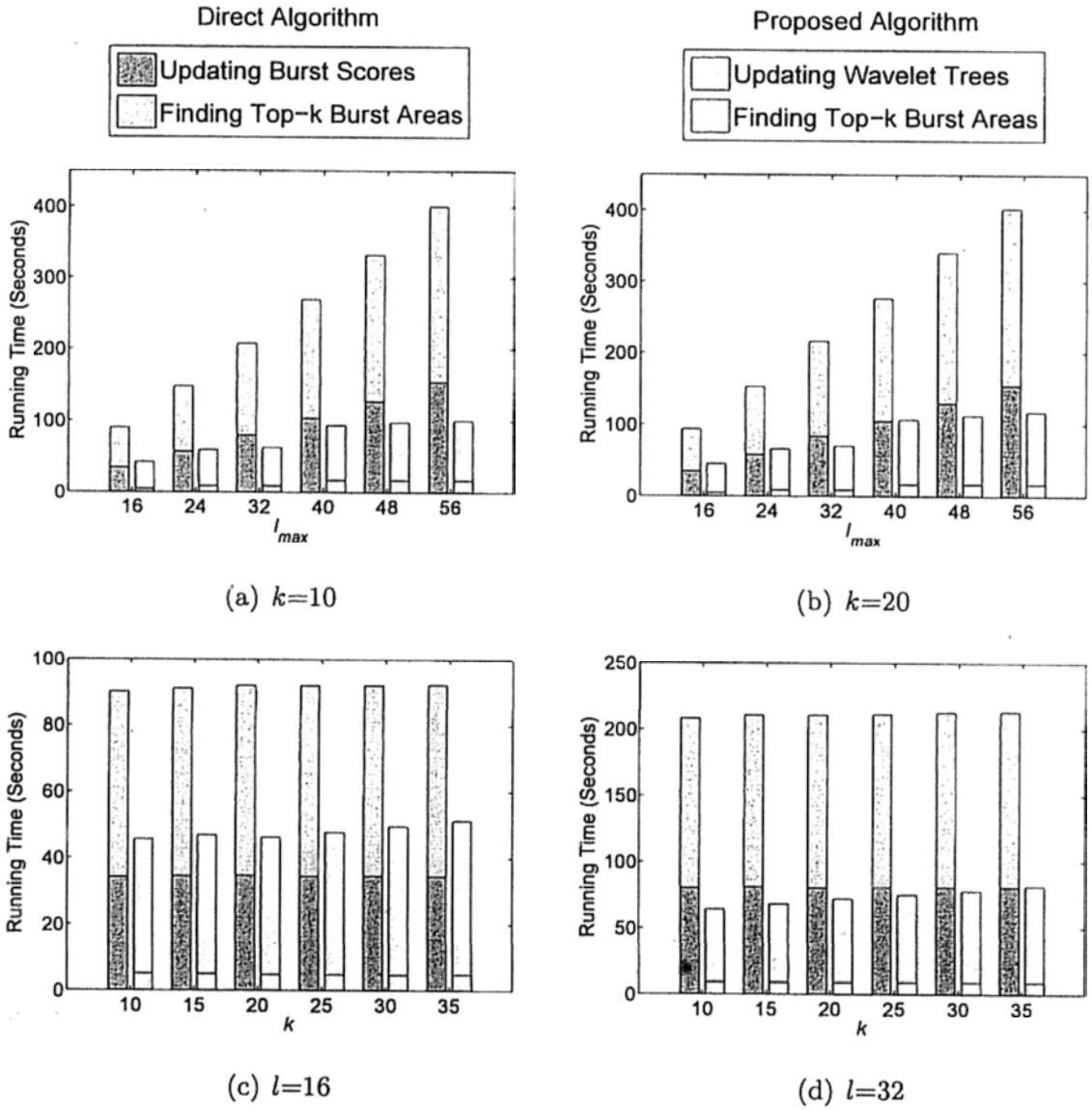
(b) k=20

(c) l=16

(d) l=32

Figure 8.9: Performance Study on Digg A

We report the pruning ability in dataset Digg A and Digg B in Figure 8.11. Figure 8.11(a) and 8.11(b) show the pruning ability of our proposed algorithm in Digg A and Digg B when the length of burst window varies. We report two sets of results where k is 10 and 20. Figure 8.11(c) and 8.11(d) show the pruning ability of our proposed algorithm in Digg A and Digg B as the value of k changes. We report two sets of results where l is 16 and 32.The results show that our proposed algorithm is able to prune most of the detailed searches. As we can see, as the increase of the value of l or k, the pruning ability decreases slightly.



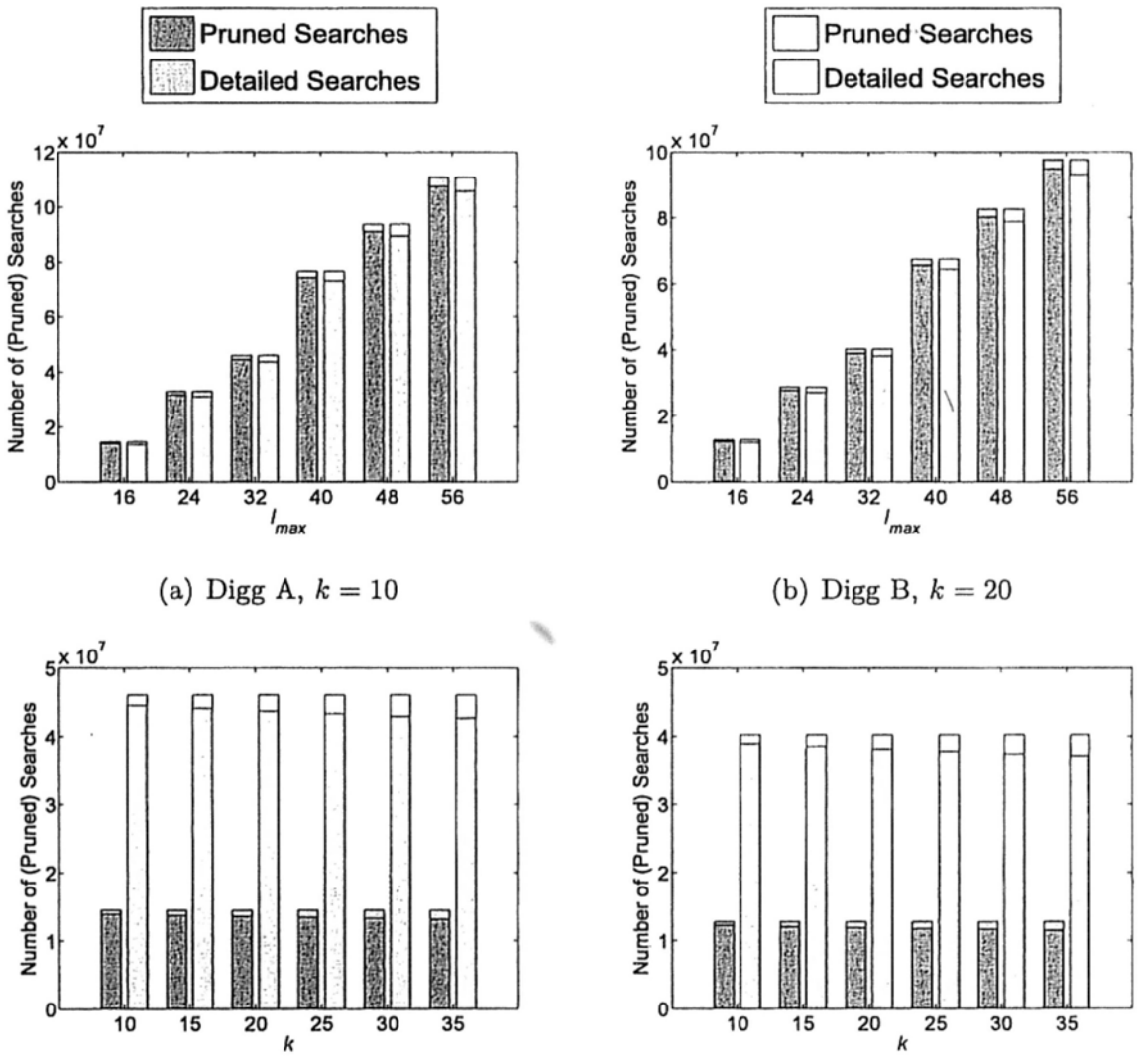
(a) k=10

(b) k=20

(c) l=16

(d) l=32

Figure 8.10: Performance Study on Digg B



(a) Digg A, $k = 10$

(b) Digg B, $k = 20$

(c) Digg A, $l = 16$

(d) Digg B, $l = 32$

Figure 8.11: Pruning Ability of The Proposed Algorithm

CHAPTER 9

Spotting Significant Changing Subgraphs in Evolving Graphs

In this chapter, we explain in detail how to spot significant changing subgraphs in evolving graphs. The chapter is organized as follows. Section 9.1 introduces the preliminary background and formalizes the problem of spotting significant subgraphs. We present our incremental computation approach and the expanding algorithm in Section 9.2 and Section 9.3, respectively. Section 9.4 reports the experimental results and Section 9.5 discusses some alternate node closeness measures.

9.1. Changing Subgraph Discovery

We model an evolving graph as a sequence of undirected graphs, denoted as $\mathcal{G} = (G_1, G_2, \cdots)$, where nodes/edges can be added and/or deleted into/from G_{i-1} which results in another large graph G_i. $G_i(V_i, E_i)$ is a snapshot of graph $\mathcal{G}$ at time t_i with a set of nodes V_i and a set of edges E_i. For simplicity, given two graphs $G_i(V_i, E_i)$ and $G_{i-1}(V_{i-1}, E_{i-1})$, the two sets of nodes, V_i and V_{i-1}, are identical, while the two sets of edges, E_i and E_{i-1}, are possibly different. The notations used in this chapter are summarized in Table 9.1.

Consider an evolving graph. An edge change may make some nodes become closer and at the same time may make some other nodes become looser. As

Table 9.1: Notations in Chapter 9

Symbol	Definition
$\mathcal{G}$	An evolving graph
G_i	The snapshot of evolving graph G at time t_i
A_i	The adjacency matrix of graph G_i
P_i	The transition matrix of graph G_i
v_j	A node on a graph
$N(v_j)$	The set of neighbors of node v_j
$d(j)$	The sum of edge weights between node v_j and $N(v_j)$
D_i	The diagonal matrix where $d_{jj} = d(v_j)$ at time t_i
Π_i	The node closeness matrix at time t_i
VI_i	The node importance score at time t_i

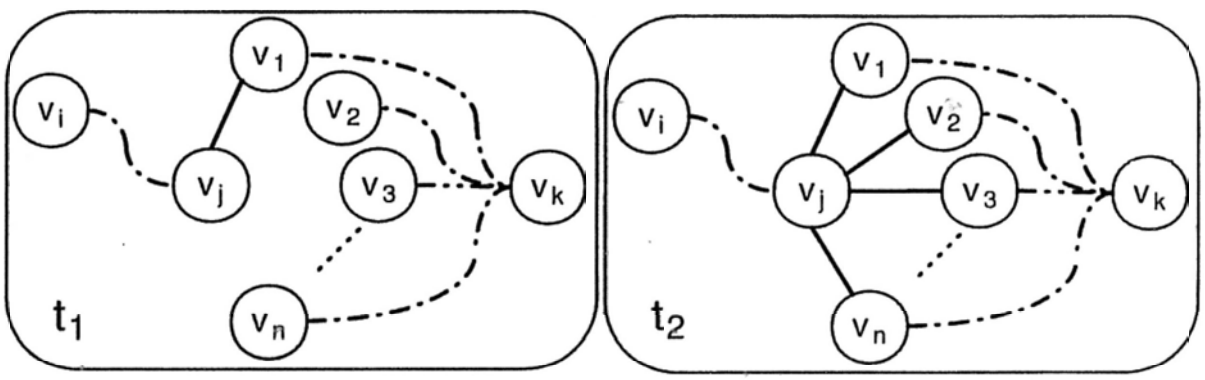


Figure 9.1: Relationship Changes as Edge Changes

shown in Figure 9.1, there are two graphs G_1 and G_2 in an evolving graph at time t_1 and t_2, respectively. At time t_1, there is an edge between v_i and v_j, and there is a path between v_j and v_k. At time t_2, there are more paths from v_j to v_k, where the edge between v_i and v_j remains unchanged. In comparison with G_1 at time t_1, the closeness between v_j and v_k becomes closer, because the newly added edges make it easier for v_j to traverse to v_k. On the other hand, the closeness between v_j and v_i becomes looser, because v_j has more opportunities to traverse to other nodes. This fact motivates us to consider an accumulative

score that measures the overall impacts of all changes on a node when a graph evolves. Reconsidering v_j in Figure 9.1, we need to consider the relationships between v_j and v_i, between v_j and v_k, and between v_j and any other node, in order to judge the change influence on v_j. Given such an accumulative score, it becomes possible to find significant changing subgraphs when a graph evolves.

We explore this issue using neighborhood random walks on graphs to help spotting significant changing subgraphs. We first review some basic concepts of random walks on graphs. Let A_i denote the adjacency matrix of a weighted graph G_i, where $A_i(j,k)$ maintains the weight for the edge (v_j, v_k). A random walk on G_i is performed in the following way. A particle starts at a certain node v_0 of the graph G_i. Suppose it walks to a node v_s in the s-th step and it is about to move to one of the neighbors of v_s, denoted as $v_t \in N(v_s)$, with probability p_{st}, where p_{st} is $A_i(s,t)/\sum_{v_k \in N(v_s)} A_i(s,k)$, and $N(v_s)$ contains all neighbors of node v_s. The node sequence of the random walk is a Markov chain. Let D_i be the diagonal matrix with the diagonal value $d(s) = \sum_{v_k \in N(v_s)} A_i(s,k)$, then the transition probability matrix P_i of the Markov chain for graph G_i is

$$P_i = D_i^{-1} A_i. \tag{9.1}$$

The probability of going from v_j to v_k through a random walk of length l can be obtained by multiplying the transition probability matrix l times and is given as $P_i^l(j,k)$.

An infinite step approach ($l \to \infty$): One possible distance measure from node v_j to node v_k is defined as the steady-state probability $P_i^l(j,k), l \to \infty$, which is the probability that the particle starting from v_j will be on node v_k after an infinite number of steps. While it might be working some field such as biology [10], it has a big drawback. When G_i is not a bipartite graph, with the memoryless property of Markov chains, the steady-state probability distribution follows the equation below [41]:

$$\forall j, \lim_{l \to +\infty} P_i^l(j,k) = \frac{d(k)}{\sum_{v_s \in V_i} d(s)}. \tag{9.2}$$

Here, V_i is the set of nodes of graph G_i. Recall that $d(s) = \sum_{v_k \in N(v_s)} A_i(s,k)$. Eq. (9.2) states that the probability of random walk to a certain node v_k from any initial node v_j shares the same limit value. In other words, random talk to a certain node is independent from the initial node. Therefore, such a measure cannot be effectively used to measure the closeness of two nodes.

An infinite step approach with restart ($l \to \infty$ and $0 < c < 1$): It is important to note that the change of closeness between two nodes should follow the principle that such a change is caused by a nearby edge change rather than an edge change that is far away. Here the issue is the locality of edge changes. In other words, the closeness measure needs to easily capture the nearby local structural information. Note that a random walk with an infinite number of steps ($l \to \infty$) can possibly visit the entire graph and might go to as far as possible from where it starts. One possible approach is to use a restart probability c ($c < 1$). Here $c(1-c)^l$ implies the probability of jumping back to the initial starting node at the l-th step. Since $c < 1$, when l is small (close to the initial starting node), the probability of jumping back is high; and when l is large (away from the initial starting node), the probability of jumping back becomes small [45]. It requires to compute the transition probability matrix P_i until it converges, which is a time consuming process. In the literature, a small $c \ll 0.5$ is usually used. However, in our problem setting, with a small $c \ll 0.5$, there are possibilities that random walks will visit nodes that are far away from the initial node, and make it uncertain how the local structural information is captured. The problem cannot be solved by simply using a large c value, because the meaning of random walk with a larger c value becomes less obvious.

A fixed step approach with restart (fixed l and $0 < c < 1$): With a fixed l, we focus on the local structural information using neighbors of a node, v_j, from which the random walk starts. The node, v_j, to start random walks is the node that is involved in an edge change. The neighbors of v_j are the nodes that v_j can reach in l steps. Random walks are only conducted in the l-step neighborhood of

the node v_j with a restart probability c. It is important to note that our algorithm is designed in a way that a user can enlarge the l value if needed at run time. We adopt the similar expected f-distance in [26, 25]. In the expected f-distance, a parameter c is used to let the expected f-distance prefer the shorter path. The proof in the Appendix shows such a parameter c is the restart probability used in [45] with minor difference which can be ignored. In short, neighborhood random walk distance, which is also called the *node closeness*, is the expected f-distance defined on random walks whose length is smaller or equal to l.

Definition 9.1. Neighborhood Random Walk Distance (Node Closeness). Let P_i be the $n \times n$ transition probability matrix of a graph G_i. Given l as the length that a random walk can go, the neighborhood random walk distance $\Pi^l(j,k)$ from v_j to v_k is defined as follows:

$$\pi(j,k) = \sum_{\tau: v_j \rightsquigarrow v_k; length(\tau) \leq l} p(\tau) c(1-c)^{length(\tau)}, \tag{9.3}$$

where $0 < c < 1$, and τ is a path from v_j to v_k whose length is $length(\tau)$ with transition probability $p(\tau)$.

The matrix form of the neighborhood random walk distance is as follows.

$$\Pi_i^l = \sum_{\gamma=1}^{l} c(1-c)^{\gamma} P_i^{\gamma}. \tag{9.4}$$

Here, P_i is the transition probability matrix for graph G_i, and Π_i is the neighborhood random walk distance matrix for graph G_i. We then define the *importance score* of a node as the accumulative change of its closeness to other nodes in Eq. (9.5).

$$VI_i(v_j) = \sum_{v_k \in V_i} |\Pi_{i-1}^l(j,k) - \Pi_i^l(j,k)|. \tag{9.5}$$

Here, $VI_i(v_j)$ is the importance score of a node v_j when a graph evolves from graph G_{i-1} to G_i.

The problem of spotting significant changing subgraphs in an evolving graph becomes a clustering problem based on the change importance score $VI_i(v_j)$

Algorithm 9.1 The Framework

Input: Two graphs G_i and G_{i-1} of an evolving graph $\mathcal{G}$ at time t_i and t_{i-1}, an integer range l, and a restart probability c

Output: The significant changing subgraphs from G_{i-1} to G_i

1: Compute the importance scores for nodes in G_i;

2: Find significant changing subgraphs from nodes with high importance scores;

using neighborhood random walk distances. We propose a two-step framework as illustrated in Algorithm 9.1. First, we compute the importance score $VI_i(v_j)$ for any node v_j in graph G_i that is involved in edge changes. Second, based on the importance scores, we find significant changing subgraphs.

9.2. Node Importance Score Computation

In this section, we discuss in detail how to calculate the difference of node closeness in two graphs G_{i-1} and G_i and the node importance scores.

9.2.1. The Straightforward Algorithm

We can develop a straightforward algorithm, which is presented in Algorithm 9.2, to compute the difference of node closeness and the node importance score based on the definitions. The straightforward algorithm iteratively calculates the respective closeness matrices Π_{i-1} and Π_i at time t_{i-1} and t_i based on Eq. (9.4) by the power method (Lines 1-12). The closeness difference matrix is simply computed as $C_i = \Pi_i - \Pi_{i-1}$, based on which the importance scores of nodes can be easily computed by Eq. (9.5). In each loop from lines 8 to 11, Algorithm 9.2 needs to multiply two matrices Q_i and P_i, which takes $O(n^3)$ time, where n is the number of nodes in the evolving graph G. Since the total number of loops is $l-1$, the time complexity of Algorithm 9.2 is $O(ln^3)$. One can use the fast sparse matrix multiplication instead of the normal matrix multiplication to improve the speed, but usually that is not enough to lower the running time especially when

Algorithm 9.2 The Straightforward Algorithm

Input: The adjacency matrices A_{i-1} and A_i for two graphs G_{i-1} and G_i at time t_{i-1} and t_i, an integer range l, and a restart probability c

Output: The closeness difference matrix C_i and the node importance vector VI_i

1: $P_i = D_i^{-1} A_i$;
2: $P_{i-1} = D_{i-1}^{-1} A_{i-1}$;
3: $Q_i = P_i$;
4: $Q_{i-1} = P_{i-1}$;
5: $\Pi_i = c(1-c)Q_i$;
6: $\Pi_{i-1} = c(1-c)Q_{i-1}$;
7: **for** $\gamma = 2$ to l **do**
8: $\quad Q_i = Q_i * P_i$;
9: $\quad Q_{i-1} = Q_{i-1} * P_{i-1}$;
10: $\quad \Pi_i = \Pi_i + c(1-c)^\gamma Q_i$;
11: $\quad \Pi_{i-1} = \Pi_{i-1} + c(1-c)^\gamma Q_{i-1}$;
12: **end for**
13: $C_i = \Pi_i - \Pi_{i-1}$;
14: **for** each v_j in V **do**
15: $\quad VI_i(j) = \sum_{v_k \in V} |C_i(j,k)|$
16: **end for**

G is large and there are a lot of edge changes.

Due to the high computational complexity of Algorithm 9.2, we seek for a more efficient way to compute the node importance scores. Since we care more for the node pair whose closeness may be different in G_{i-1} and G_i, it is not necessary to compute the whole closeness matrices Π_{i-1} and Π_i. One possible way to improve the efficiency of Algorithm 9.2 is to calculate the closeness of those node pairs that may change, rather than computing all node pairs.

Let $\Delta A = A_i - A_{i-1}$ be the difference adjacency matrix. All non-zero entries in ΔA represent the edge difference between G_i and G_{i-1}. Let $V(\Delta A)$ be a node

set, where each node $v_j \in V(\Delta A)$ has at least one non-zero entry on the j-th row of ΔA (i.e., at least one changing edge connected to v_j). Suppose that $\{v_j, v_k\}$ is a pair of nodes whose closeness $\Pi(j,k)$ needs to be computed. Then, there must exist nodes $v_m, v_n \in V(\Delta A)$, where (v_m, v_n) is a changing edge, such that the neighborhood random walks starting from v_m or v_n can reach v_j and v_k. In other words, v_m or v_n or both must lie in some tour path that connects v_j and v_k. Such node pair $\{v_j, v_k\}$ forms the *influence set* of v_m and v_n, since the closeness of $\{v_j, v_k\}$ is influenced and changed by the edge change of (v_m, v_n). Therefore, by pruning the nodes in V that are not in the influence set of any node in $V(\Delta A)$, we can improve the speed of Algorithm 9.2. The solution is to construct induced subgraphs G'_i and G'_{i-1}, which consist of only the nodes in the influence sets of $V(\Delta A)$, and use the corresponding adjacency matrix A'_i and A'_{i-1} in Algorithm 9.2. The closeness difference related to all other nodes are all zero. The time complexity of this approach is $O(l(\alpha n)^3)$, where α is the fraction of nodes that are in influence sets.

Unfortunately, the above approach does not always work very well. For example, in social networks, the average graph diameter is usually small due to the small world property [16]. As a result, the size of the induced subgraphs G'_i and G'_{i-1} for such networks is close to that of the original graphs G_i and G_{i-1}. In other words, the corresponding α is close to 1 and the complexity of this approach is almost the same as that of Algorithm 9.2.

9.2.2. A Novel Incremental Algorithm

As mentioned above, in order to compute the node importance score, we only need to consider the node pairs whose closenesses change and it is not necessary to calculate the closeness for all node pairs. In this section, we introduce a novel incremental algorithm that computes the closeness difference directly for those node pairs with changing closeness.

Let us start from a simple case. Suppose that there is only one edge e

that is different between two graphs G_i and G_{i-1}. It can be either the addition of e to G_i or the deletion of e from G_{i-1}. The question is to identify those node pairs whose closenesses change due to the difference of e, as well as the quantities changed. Recall our closeness measure in Eq. 9.3. The answer to the above question is that if a node pair has at least one tour path passing through the edge e or one of the two nodes incident to edge e, the closeness of the node pair changes. By identifying those paths, we can find the node pairs with changing closenesses. Furthermore, the summation of the differences of these path probabilities is exactly the quantity changed in the closeness of each node pair.

By Eq. (9.4), the iterative form of the node closeness is

$$\begin{aligned}\Pi_i^l &= \sum_{\gamma=1}^{l} c(1-c)^\gamma P_i^\gamma \\ &= c(1-c)^l P_i^l + \sum_{\gamma=1}^{l-1} c(1-c)^\gamma P_i^\gamma \\ &= c(1-c)^l P_i^l + \Pi_i^{l-1}.\end{aligned} \tag{9.6}$$

Therefore, the closeness difference matrix is

$$\begin{aligned}\Delta\Pi_i^l &= \Pi_i^l - \Pi_{i-1}^l \\ &= (c(1-c)^l P_i^l + \Pi_i^{l-1}) - (c(1-c)^l P_{i-1}^l + \Pi_{i-1}^{l-1}) \\ &= c(1-c)^l (P_i^l - P_{i-1}^l) + (\Pi_i^{l-1} - \Pi_{i-1}^{l-1}) \\ &= c(1-c)^l (P_i^l - P_{i-1}^l) + \Delta\Pi_i^{l-1}.\end{aligned} \tag{9.7}$$

By Eq. (9.7), we can see that the key step in computing $\Delta\Pi_i^l$ is to compute $(P_i^l - P_{i-1}^l)$, which is easy when $l = 1$. When $l > 1$, obviously we cannot compute it in a naive way by the power method since it is computational expensive. Recall that $P_i^l(j,k)$ is the probability of going from v_j to v_k through random walks of length l on graph G_i. We now show how to calculate $\Delta P_i^l = P_i^l - P_{i-1}^l$ in an efficient way. Apparently,

$$P_i^l(j,k) = \sum_{\tau: v_j \rightsquigarrow v_k} p_i(\tau), \tag{9.8}$$

where τ is a tour path from node v_j to v_k, and $p_i(\tau)$ is the probability of path τ in G_i. Suppose $\tau = \langle v_1, v_2, ..., v_l \rangle$, where $v_1 = v_j$ and $v_l = v_k$, then $p_i(\tau) = \prod_{m=1}^{l-1} A_i(v_m, v_{m+1})/d(v_m)$. The sum of the probability of all these distinct tour paths is $P_i^l(j,k)$.

In order to compute ΔP_i^l, we only need to consider the different paths on G_i and G_{i-1}, as well as the difference in the probability of the same paths. For simplicity, we only discuss the case when there are only additions of edges or increase of edge weights. We will show later that our algorithm can handle deletions of edges and decrease of edge weights as well.

Let (v_m, v_n) be one of the added edges or one of the edges whose weights increase. For any node pair $\{v_j, v_k\}$, if there is a tour path τ of the maximum length l starting from v_j, passing through the edge (v_m, v_n) and ending at v_k, then the node closeness $\Pi_i(j,k)$ will increase by $p_i(\tau)$, since this path does not exist in G_{i-1}. On the other hand, if there is a tour path τ of the maximum length l starting from v_j, passing v_m or v_n or both, and ending at v_n, but without passing through (v_m, v_n), then the node closeness $\Pi_i(j,k)$ will decrease by $(p_{i-1}(\tau) - p_i(\tau))$, since the path τ exists in both G_i and G_{i-1}, and with the increase of $d(m)$ and $d(n)$ in G_i, the probability of the path τ decreases. We formalize the above analysis in Theorem 9.1.

Theorem 9.1. *Given two graphs G_i and G_{i-1} of an evolving graph G, let (v_m, v_n) denote the changing edge, then $\Delta P_i^l(j,k)$ can be computed as follows:*

$$\Delta P_i^l(j,k) = \sum_{\tau: v_j \rightsquigarrow v_k; (v_m,v_n) \in \tau} p(\tau) + \sum_{\tau: v_j \rightsquigarrow v_k; (v_m,v_n) \notin \tau; v_m \text{ or } v_n \in \tau} (p_i(\tau) - p_{i-1}(\tau)). \tag{9.9}$$

Theorem 9.1 suggests an effective way to calculate the change quantity of the closeness between node pairs. The key is to find all the related paths distinctly and completely so that the change quantity is computed correctly. To enumerate

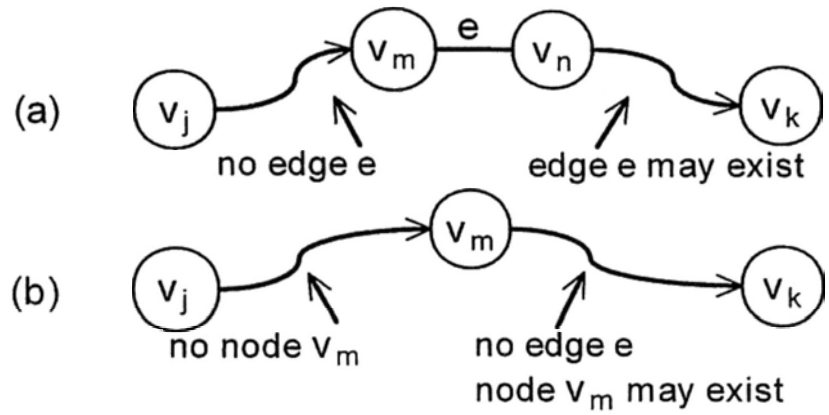


Figure 9.2: Path Enumeration for Correct Closeness Difference Computation

all the possible positions of the edge (v_m, v_n) in a path τ is obviously not a good solution due to the exponential number of combinations with respect to the number of changing edges and the range l.

We first discuss the case of the path $\tau : v_j \rightsquigarrow v_k$ when $(v_m, v_n) \in \tau$. We can calculate the closeness difference in the following way. For the changing edge $e = (v_m, v_n)$, we first calculate the probability of a path τ_1 from v_j to v_m with length l_1, where $l_1 \leq l-1$. We then calculate the probability of a path τ_2 from v_n to v_k with length $l_2 = l - l_1 - 1$. In this way, we ensure that the computed paths from v_j to v_k passing the edge (v_m, v_n) is of length l. The closeness difference that is accounted for such paths can be computed as

$$\pi_1(j,k) = \sum_{\tau_1 : v_j \rightsquigarrow v_m ; \tau_2 : v_n \rightsquigarrow v_k} p(\tau_1) P_i(m,n) p(\tau_2), \tag{9.10}$$

where $\pi_1(j,k)$ denotes the first term of $\Delta P_i^l(j,k)$ in Eq. (9.9).

In order to compute $p(\tau_1 : v_j \rightsquigarrow v_m)$ and $p(\tau_2 : v_n \rightsquigarrow v_k)$ correctly without missing and double-computing any path, we perform the random walks as shown in Figure 9.2(a). We do not allow a path τ_1 from v_j to v_m to pass e , while we do not have this restriction on path τ_2.

As for the other case of the path $\tau : v_j \rightsquigarrow v_k$ when v_m or $v_n \in \tau$ but

$(v_m, v_n) \notin \tau$, the closeness difference can be computed in a similar way:

$$\pi_2(j,k) = \sum_{\substack{\tau_3: v_j \rightsquigarrow v_m; \tau_4: v_m \rightsquigarrow v_k \\ or\ \tau_3: v_j \rightsquigarrow v_n; \tau_4: v_n \rightsquigarrow v_k}} p(\tau_3)p(\tau_4), \tag{9.11}$$

where $\pi_2(j,k)$ denotes the second term of $\Delta P_i^l(j,k)$ in Eq. (9.9). For the correctness of computation, we do not allow τ_3 to contain the node v_m (or v_n when τ_3 is from v_j to v_n) and we do not allow τ_4 to contain the edge e, as shown in Figure 9.2(b).

The whole incremental algorithm to compute the closeness difference matrix is presented in Algorithm 9.3. $\mathcal{I}$ and $\mathcal{J}$ at line 5 and 6 denote the index of all changed nodes and all unchanged nodes, respectively. A changed node is the node that at least one added edge connects to it. Let P denote a matrix. $P[:, \mathcal{I}]$ denotes the submatrix of all rows and all i-th columns, where $i \in \mathcal{I}$. $P[\mathcal{I}, :]$ denotes the submatrix of all i-th rows and all columns. $P[\mathcal{J}, \mathcal{I}]$ denotes the submatrix of all j-th rows and all i-th columns, where $j \in \mathcal{J}$ and $i \in \mathcal{I}$.

When we explain the strategy of our incremental algorithm, for simplicity, we assume only one edge is added to the graph during the evolving, but in reality, there may be many edge changes at one time. Apparently, it is not a good way to handle them one by one. In Algorithm 9.3, We handle all the changed edges together in matrix form instead of one by one. The key idea is similar, we use two arrays of $P_i^{\tau_1}$ and $P_i^{\tau_2}$ to store the corresponding probabilities of the τ_1 and τ_2 types of paths. Four other arrays of $P_{i-1}^{\tau_3}$, $P_{i-1}^{\tau_4}$, $P_i^{\tau_3}$ and $P_i^{\tau_4}$ are used to store the probabilities of the τ_3 and τ_4 types of paths on graph G_{i-1} and G_i, respectively.

The algorithm computer the difference distances matrix C_i incrementally In each loop starting at line 14, the algorithm first computer the probabilities of four types paths. For a certain length $k \leq l$, there are k possible positions for the changed edge (v_m, v_n) or node v_m or v_n on the tour path. During the loop from line 22 to line 33, the algorithm enumerate all the k possibilities and sum up all the paths to obtain the matrix R, which equals to the item $(P_i^l - P_{i-1}^l)$ in Eq. (9.7). One may argue that in order to store the six arrays of matrices, a

Algorithm 9.3 The Incremental Algorithm

Input: The adjacency matrices A_{i-1} and A_i for two graphs G_{i-1} and G_i at time t_{i-1} and t_i, respectively a integer of range l

Output: The difference distances matrix C_i and the node change importance vector VI.

$P_{i-1} = D_{i-1}^{-1} A_{i-1}$;
$P_i = D_i^{-1} A_i$;
$P_{Ui} = D_i^{-1} A_{i-1}$;
$P_{Ci} = P_i - P_{Ui}$;
$\mathcal{I}$ = Index of all changed nodes;
$\mathcal{J}$ = Index of all unchanged nodes;
$P_i^{\mathcal{T}1}[1] = P_{Ui}[:, \mathcal{I}]$;
$P_i^{\mathcal{T}2}[1] = P_i[\mathcal{I}, :]$;
$P_{i-1}^{\mathcal{T}3}[1] = P_{i-1}[\mathcal{J}, \mathcal{I}]$;
$P_{i-1}^{\mathcal{T}4}[1] = P_{i-1}[\mathcal{I}, :]$;
$P_i^{\mathcal{T}3}[1] = P_{Ui}[\mathcal{J}, \mathcal{I}]$;
$P_i^{\mathcal{T}4}[1] = P_{Ui}[\mathcal{I}, :]$;
$C_i = c(1-c)(P_i - P_{i-1})$;
for $k = 2 : l$ **do**
    $P_i^{\mathcal{T}1}[k] = P_{Ui} P_i^{\mathcal{T}1}[k-1]$;
    $P_i^{\mathcal{T}2}[k] = P_i^{\mathcal{T}2}[k-1] P_i$;
    $P_{i-1}^{\mathcal{T}3}[k] = P_{i-1}[\mathcal{J}, \mathcal{J}] P_{i-1}^{\mathcal{T}3}[k-1]$;
    $P_{i-1}^{\mathcal{T}4}[k] = P_{i-1}^{\mathcal{T}4}[k-1] P_{i-1}$;
    $P_i^{\mathcal{T}3}[k] = P_{Ui}[\mathcal{J}, \mathcal{J}] P_i^{\mathcal{T}3}[k-1]$;
    $P_i^{\mathcal{T}4}[k] = P_i^{\mathcal{T}4}[k-1] P_{Ui}$;
    $R = n \times n$ zero matrix;
    for $m = 1 : k$ **do**
        if $m == 1$ **then**
            $R[\mathcal{I}, :] = R[\mathcal{I}, :] + P_{Ci}[\mathcal{I}, \mathcal{I}] P_i^{\mathcal{T}2}[k-m]$;
            $R[\mathcal{I}, :] = R[\mathcal{I}, :] + P_i^{\mathcal{T}4}[k-m] P_{Ui} - P_{i-1}^{\mathcal{T}4}[k-m] P_{i-1}$;
        else if $m == k$ **then**
            $R[:, \mathcal{I}] = R[:, \mathcal{I}] + P_i^{\mathcal{T}1}[m-1] P_{Ci}[\mathcal{I}, \mathcal{I}]$;
            $R = R + P_i^{\mathcal{T}3}[m-1] P_{Ui}[\mathcal{I}, :] - P_{i-1}^{\mathcal{T}3}[m-1] P_{i-1}[\mathcal{I}, :]$;
        else
            $R = R + P_i^{\mathcal{T}1}[m-1] P_{Ci}[\mathcal{I}, \mathcal{I}] P_i^{\mathcal{T}2}[k-m]$;
            $R[\mathcal{J}, :] = R[\mathcal{J}, :] + P_i^{\mathcal{T}3}[m-1] P_i^{\mathcal{T}4}[k-m] P_{Ui} - P_{i-1}^{\mathcal{T}3}[m-1] P_{i-1}^{\mathcal{T}4}[k-m] P_{i-1}$;
        end if
    end for
    $C_i = c(1-c)^k R + C_i$;
end for
for each v_j **in** V **do**
    $VI(j) = \sum_{v_k \in V} |C_i(j,k)|$;
end for

large amount of memory is needed. But the fact is that in worst case, each sparse matrix contains $n|\mathcal{I}|$ entries. So the total number of entries are $6nl|\mathcal{I}|$. Since l is a small number and $|\mathcal{I}|$ is usually much smaller than n, the total number of entries is then smaller than n^2.

We have discussed how to handle the additions of edges and the increase of edge weights. In fact, our algorithm can also handle the situation when there are deletions of edges and decrease of edge weights. Let us first suppose that there are only deletions of edges and decrease of edge weights from G_{i-1} to G_i. It is easy to see that this is exactly the same as the evolvement from G_i to G_{i-1}, where only additions of edges and increase of edge weights happen. The only difference is that the closeness difference matrix should be multiplied by -1. In general, we can first handle all the additions of edges, together with the increase of edge weights, and then handle the deletions of edges and decrease of edge weights. In order to do this, we can add a ghost graph, G'_i, such that $(G'_i - G_{i-1})$ contains all the edges added or with increased weights and $(G_i - G'_i)$ contains all the edges deleted or with decreased weights. The sum of these two closeness difference matrices gives the same closeness difference matrix from G_{i-1} to G_i.

9.3. Spotting Significant Subgraphs

With the closeness difference matrix Π_i^l at time t_i and the node importance score vector VI, we now explain how to expand those nodes of high importance scores to obtain significant changing subgraphs. As mentioned, a changing subgraph is significant if the node closeness in the subgraph changes a lot. In our experiments, we find that the node importance scores follow the power law distribution. Therefore, instead of defining an absolute threshold for the score, we use the value ξ as the threshold such that more than 85% of the scores are smaller than it. Apparently, significant changing subgraphs should contain all the important nodes (i.e., those with high importance scores beyond the threshold ξ) and most of the nodes whose closenesses with the important nodes change a lot.

We develop an expanding strategy which is similar to the density clustering. The basic idea is to include the nodes whose closeness differences with the important nodes are high.

We present the expanding algorithm in Algorithm 9.4. A max heap is used to store all the neighbors of current subgraph g. At line 2, the union graph of G_{i-1} and G_i only keeps the information of connectivity. The algorithm starts from an important node v_j with the maximum importance score in each loop at line 4 to generate a significant changing subgraph. At line 9, the algorithm fetches the first node v_k in heap H. $\mathcal{I}(V(g))$ in line 11 is the index of all the nodes in subgraph g. From line 11 to 15, the algorithm checks whether the max node closeness difference from the important nodes in the current significant subgraph g to v_k is smaller than the threshold ϵ. If so, the algorithm clears the heap H and outputs the significant changing subgraph g. At line 17, threshold ϵ is set to be 1/5 of the maximum transition probability of the important node lastly included into the subgraph. Next, all the unvisited neighbors of the node lastly included into the subgraph are inserted into the heap H at line 20 to 25. This procedure is repeated until all the important nodes are visited. In the final result set of the significant changing subgraphs, two subgraphs are merged if they are directly connected.

9.4. Experimental Evaluation

In this section, we present the experimental results on four real datasets to show both the effectiveness and the efficiency of our proposed algorithms.

Algorithm 9.4 The Expanding Algorithm

Input: The adjacency matrices A_{i-1} and A_i for two graphs G_{i-1} and G_i at time t_{i-1} and t_i; the closeness difference matrix Π_i at time t_i

Output: The significant changing subgraphs

Let H be a max heap;
Let G' be the union graph of G_{i-1} and G_i;
while not all the important nodes visited **do**
  v_j = the unvisited important node with the highest importance score;
  Let v_m be the node that $\Pi_i(j, m)$ is maximum among all unvisited nodes;
  Insert $\langle v_j, \Pi_i(j, m)\rangle$ to the max heap H;
  $\epsilon = 0$;
  while H is not empty **do**
    v_k = the first node in H;
    if $|V(g)| > 0$ **then**
      if $\max(\Pi_i(\mathcal{I}(V(g)), k)) < \epsilon$ **then**
        Empty H;
        Output the current subgraph g;
      end if
    end if
    if v_k is an importance node **then**
      $\epsilon = \max(\Pi_i(k, :))/5$;
    end if
    Mark v_k as visited and add v_k to the current subgraph g;
    for each neighbor v_n of v_k on the union graph G' **do**
      if v_n is unvisited and not in H **then**
        Let v_m be the node that $\Pi_i(n, m)$ is maximum among all unvisited nodes;
        Insert $\langle v_n, \Pi_i(n, m)\rangle$ into the heap H;
      end if
    end for
  end while
end while

Table 9.2: Dataset Characteristics

Datasets	Nodes	Average Edges Added	Time Steps
DB	5492	1734	10
DM	5574	1079	10
Enron2001	16639	320	184
Enron2002	16639	203	164

9.4.1. Datasets

The four real datasets are extracted from the DBLP [1] co-authorship dataset and the Enron email dataset [3]. In the DBLP co-authorship dataset, each author is represented by a node and there is an edge between two authors if they co-authored some paper. In the Enron email dataset, each email sender or receiver is considered as a node and there are edges between senders and receivers. The first two datasets DB and DM are from the DBLP co-authorship dataset. DB contains the co-authorship information of six major database conferences from 1998 to 2007, including SIGMOD, PODS, VLDB, ICDE, EDBT and ICDT. DM contains the co-authorship information of five major data mining conferences from 1998 to 2007, including KDD, ICDM, PKDD, SDM and PAKDD. DB has 5492 nodes and DM has 5574 nodes. The other two datasets Enron2001 and Enron2002 are extracted from the Enron email dataset. Enron2001 contains the email communication information of each day from 2001-07-01 to 2001-12-31, while Enron2002 contains the email communication information of each day from 2002-01-01 to 2002-7-31. The number of nodes of both Enron2001 and Enron2002 is 16639. The characteristics of these datasets are summarized in Table 9.2.

The authors in [54] introduced three aggregation methods: global aggregation, exponential aggregation and sliding window. It is worth noting that our proposed approaches can cooperate with all these three aggregation methods. In

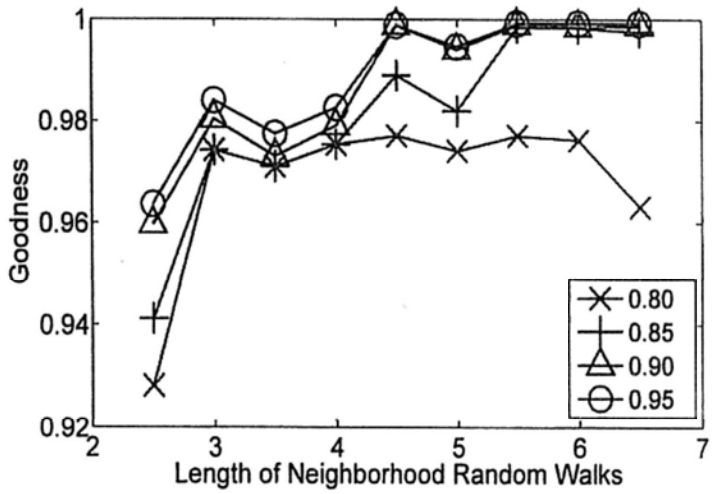


Figure 9.3: The Goodness of Significant Subgraphs in Dataset DB

our experiments, we choose global aggregation to perform our experiments, which aggregates the new edges or edge weights to the adjacency matrix of previous time. Let ΔA_i be the adjacency matrix of the graph at time step t_i, then

$$A_i = \sum_{t=1}^{i} \Delta A_t \tag{9.12}$$

The average number of added edges per time step is presented in Table 9.2.

9.4.2. Effectiveness

Let us first introduce our criterion of the significant subgraphs. Let g_i denote a significant subgraph found at time t_i. We evaluate the goodness of significant subgraphs as

$$Goodness = \frac{\sum_{v_j, v_k \in g_i} \Delta\Pi_i(j,k)}{\sum_{v_j \in g_i, v_k \in G_i} \Delta\Pi_i(j,k)}, \tag{9.13}$$

where $\Delta\Pi_i(j,k) = |\Pi_i(j,k) - \Pi_{i-1}(j,k)|$, is the closeness difference for v_j and v_k between G_{i-1} and G_i. The goodness is essentially the fraction of the closeness differences between G_{i-1} and G_i that are captured by significant subgraphs.

We use $c = 0.15$ in all experiments. Figures 9.3 and 9.4 present the average goodness for different values of ξ, which is a parameter in Algorithm 9.4, when varying the length of neighborhood random walks l from 2 to 10. For dataset

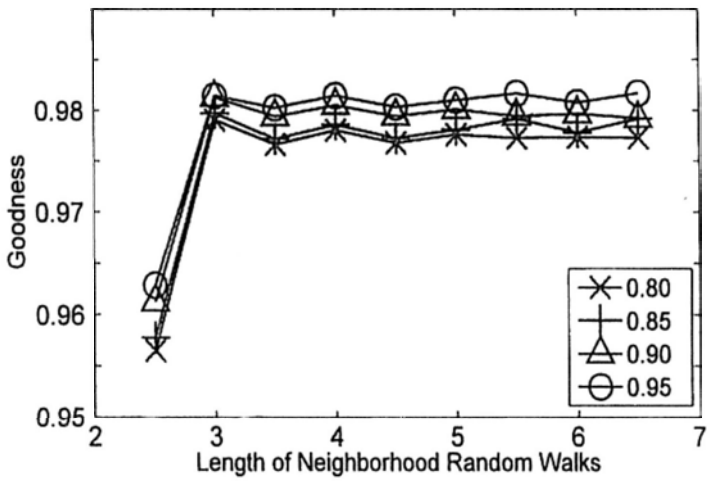
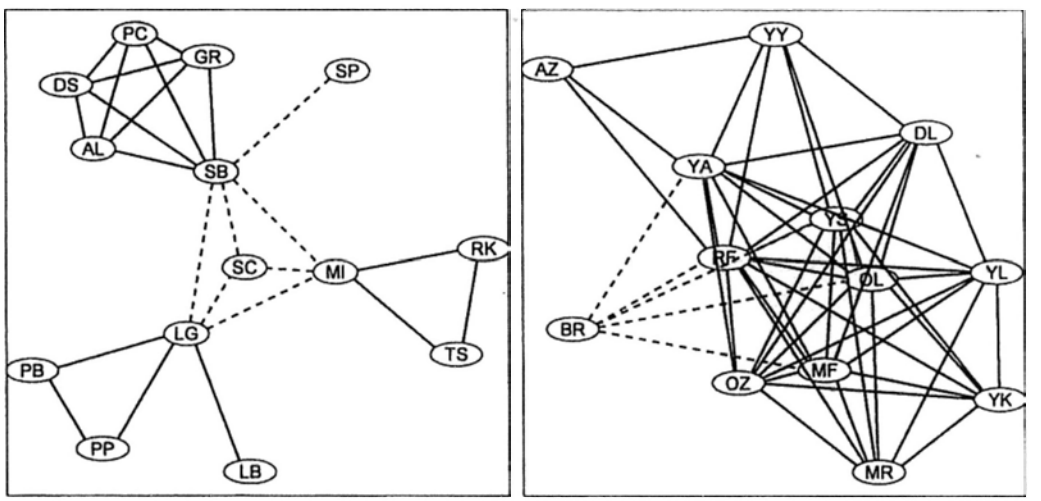


Figure 9.4: The Goodness of Significant Subgraphs in Dataset DM

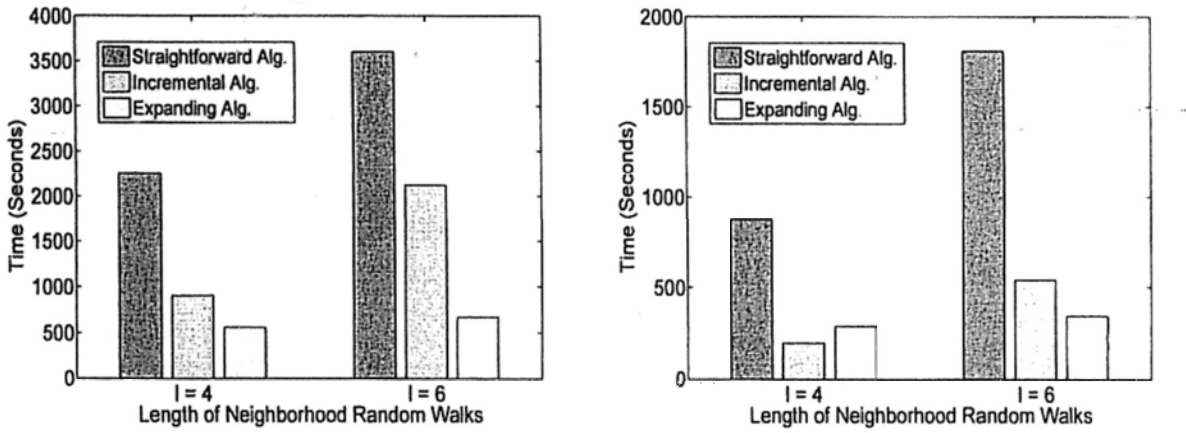


(a) Dataset DB (b) Dataset DM

Figure 9.5: Two Significant Subgraphs

DB, our algorithm captures 92% changes in node closeness, while for dataset DM, our algorithm captures more than 96%. For a higher value of ξ and longer length of l, the goodness scores increase.

Two significant subgraphs found are presented as examples in Figures 9.5(a) and 9.5(b), which is from the experiments with $l = 4$ and $\xi = 0.8$. For privacy, we replace author names by abbreviations. The newly added edges are dotted in both subgraphs. Figure 9.5(a) shows the subgraph from dataset DB. There



(a) Dataset Enron2001

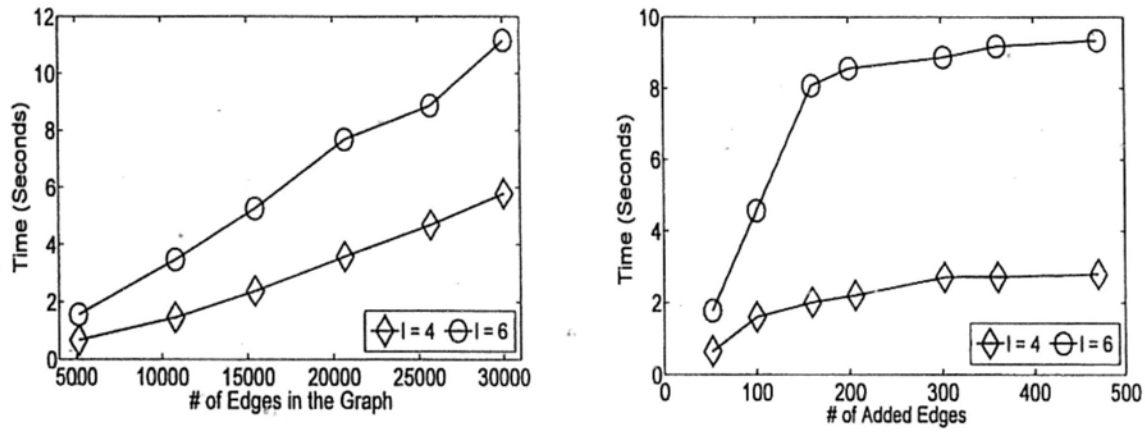
(b) Dataset Enron2002

Figure 9.6: Overall Running Time on Dataset Enron2001 and Enron2002

are originally three communities (dense areas) and the newly added edges make three communities connected, which usually indicates that there is a joint research work involving multiple research groups. Apparently, only the subgraph consisting of the added edges cannot provide this information. There are other nodes connecting to some of the nodes in three communities, and these nodes are not included in the significant subgraph because the difference of the node closeness between them and the node importance scores are small. In Figure 9.5(b), the researcher BR co-authored papers with researchers in a very dense community. Researchers in the same research group tend to co-author a lot and form a very dense community. Therefore, it is obvious that BR should be a new member to some research group.

9.4.3. Efficiency

We perform our efficiency testing on datasets Enron2001 and Enron2002. Figures 9.6(a) and 9.6(b) show the overall running time for the three algorithms: the straightforward algorithm in Algorithm 9.2, the incremental algorithm in Algorithm 9.3 to compute the node importance scores, as well as the expanding algorithm in Algorithm 9.4 to generate the significant subgraphs. Each figure



(a) The Straightforward Algorithm

(b) The Incremental Algorithm

Figure 9.7: Average Running Time on Dataset Enron2001 and Enron2002

shows two groups of running time for $l = 4$ and $l = 6$. On the Enron2001 Dataset, the incremental algorithm is almost twice faster than the straightforward algorithm, while on the Enron2002 Dataset, the incremental algorithm is about four times faster. The running time of the expanding algorithm increases slightly when l becomes larger.

Figure 9.7(a) presents the average running time of the straightforward algorithm versus the average number of edges in the graphs at each time spot for both dataset Enron2001 and Enron2002. As we can see that the running time of the straightforward algorithm is proportional to the total number of edges in the graph at current time spot. The average running time of the incremental algorithm versus the average number of the newly added edges in the graphs for both datasets is shown in Figure 9.7(b). The running time of the incremental algorithm is proportional to the total number of edges added. This explains why the incremental algorithm is faster. The running time of the incremental algorithm is more related to the number of changing edges while the running time of the straightforward one is more related to the total number of edges in the current graph. When the number of changing edges is much smaller than the total number of edges in the current graph, which is true in most evolving

graphs, the incremental algorithm is much faster than the straightforward one.

9.5. Discussion of Alternate Node Closeness Measures

9.5.1. Relationship between Expected f-Distance and Random Walk with Restart

The node closeness matrix using the expected f-distance is not much different from one using random walk with restart. The proof is presented below. Based on the iterative form of the definition of random walk with restart, the node closeness matrix $\Pi_i'^l$ of graph G_i can be expressed as following.

$$\Pi_i'^l = (1-c)\Pi_i'^{l-1}P_i + cI, \tag{9.14}$$

where c is the restart probability, P_i is the transition matrix of G_i and I is identity matrix. Then we have

$$\begin{aligned}
\Pi_i'^l &= (1-c)\Pi_i'^{l-1}P_i + cI \\
&= (1-c)((1-c)\Pi_i'^{l-2}P_i + cI)P_i + cI \\
&= (1-c)((1-c)((1-c)\Pi_i'^{l-3}P_i + cI)P_i + cI)P_i + cI \\
&= \ldots \\
&= (1-c)^l P_i^l + c\sum_{\gamma=1}^{l-1}(1-c)^\gamma P_i^\gamma + cI \\
&= c(1-c)^l P_i^l + c\sum_{\gamma=1}^{l-1}(1-c)^\gamma P_i^\gamma + (1-c)^{l+1}P_i^l + cI \\
&= \sum_{\gamma=1}^{l} c(1-c)^\gamma P_i^\gamma + (1-c)^{l+1}P_i^l + cI \\
&= \Pi_i^l + (1-c)^{l+1}P_i^l + cI.
\end{aligned} \tag{9.15}$$

The last line of Eq. (9.15) contains three items. The first item is the node closeness matrix Π_i^l using the expected f-distance. The third item cI affects

only the diagonal entries of the node closeness matrix, which is ignored since we do not consider the node self-closeness. Then, the difference using random walk with restart and the expected f-distance lies in the second item $(1-c)^{l+1}P_i^l$. When l goes to infinity, the node closeness matrices using expected f-Distance and random walk with restart are the same expect the diagonal entries. Even when l is small, the corresponding entries of two matrices do not differ so much since $(1-c)^{l+1}P_i^l$ is very small comparing with $\Pi_i^l = \sum_{\gamma=1}^{l} c(1-c)^\gamma P_i^\gamma$.

9.5.2. Using Random Walk with Restart

It is worth noting that our framework can be adapted to cooperate with the definition of random walk with restart. The node closeness matrix using random walk with restart is presented in Eq. (9.14). And based on the proof in Eq. 9.15, the closeness difference matrix is

$$\Delta\Pi_i'^l = (1-c)^l(P_i^l - P_{i-1}^l) + c\sum_{k=1}^{l-1}(1-c)^k(P_i^k - P_{i-1}^k). \tag{9.16}$$

Then we can incrementally calculate the $\Delta\Pi_i'^l$ in the following way.

$$\begin{aligned}
\Delta\Pi_i'^1 &= (1-c)(P_i - P_{i-1}),\\
\Phi_1 &= \Delta\Pi_i'^1;\\
\Delta\Pi_i'^2 &= (1-c)^2(P_i^2 - P_{i-1}^2) + c(1-c)(P_i - P_{i-1}),\\
&= (1-c)^2(P_i^2 - P_{i-1}^2) + c\Phi_1;\\
\Phi_2 &= (1-c)(\Delta\Pi_i'^2 + (1-c)\Phi_1),
\end{aligned}$$

$$\Delta\Pi'^{3}_{i} = (1-c)^3(P^3_i - P^3_{i-1}) + c\sum_{\gamma=1}^{2}(1-c)^{\gamma}(P^{\gamma}_i - P^{\gamma}_{i-1})$$

$$= (1-c)^3(P^3_i - P^3_{i-1}) + c\Phi_2;$$

$$\Phi_3 = (1-c)(\Delta\Pi'^{3}_{i} + (1-c)\Phi_2);$$

$$\ldots;$$

$$\Delta\Pi'^{l}_{i} = (1-c)^l(P^l_i - P^l_{i-1}) + c\sum_{\gamma=1}^{l-1}(1-c)^{\gamma}(P^{\gamma}_i - P^{\gamma}_{i-1})$$

$$= (1-c)^l(P^l_i - P^l_{i-1}) + c\Phi_{l-1}.$$

As we can see from the last line, the key task is still to calculate $(P^l_i - P^l_{i-1})$, which is the same as using the expected f-distance. The only difference is that the algorithm needs to store an intermediate matrix Φ_k at each step.

CHAPTER 10

CONCLUSIONS

We focus on two challenging topics in this thesis under the context of managing and mining graph data, namely, graph summarization and graph change detection. In graph summarization, we study the following challenging problems.

- **Approximate Homogeneous Graph Summarization.** We studied the problem of graph summarization using a new information-preserving approach based on information theory. A graph is summarized by partitioning node set into subsets and constructing a super-graph based on the partition. We analyzed the exact and approximate homogeneous partition criteria and proposed a unified entropy framework to relax all three criteria for homogeneity. Our proposed summarization framework can obtain the graph summary of small size and high quality, whose quality is measured by the total weighted entropy of each node subset in the partition. We proposed a lazy exact partition algorithm, as well as two other approximate partition algorithms to compute the exact homogeneous partition and the approximate homogeneous partition, respectively.

- **Frequent Subgraph Summarization with Error Control.** We proposed a frequent subgraph summarization framework with an independence probabilistic model. We formally defined the problem and applied a regression approach to estimate the parameters in the summarization model.

Our summarization framework takes a top-down approach to recursively partition a summarization graph template, until the user-specified error tolerance is met. Our summarization model can effectively control the frequency restoration error within 10% with a compact size.

In graph change detection, we study two different meanings of graph change, and focus on finding meaningful changing areas, which are summarized as follows.

- **Discovering Burst Areas in Fast Evolving Graphs.** We studied the problem of finding top-k burst areas under the context of fast graph evolutions. We proposed to update the Haar wavelet tree in a dynamic manner to avoid high computation complexity while keeping its high pruning ability. The top-k burst areas are computed incrementally from small hop size to large hop size in order to minimize memory consumption.

- **Spotting Significant Changing Subgraphs in Evolving Graphs.** We studied the challenging problem of spotting significant changing subgraphs in evolving graphs. We proposed to use the neighborhood random walk to measure the node closeness, as well as a novel incremental algorithm for fast computation. The significant subgraphs are generated based on the node importance scores.

In the near future, we are planning to extend our recent work of frequent subgraph summarization in three directions. The first direction is to integrate our summarization algorithm into the pattern mining process to avoid the computation cost of finding all frequent subgraphs. Frequent subgraph mining is a time-consuming task even for graph collections of moderate sizes. An integrated framework could provide users the estimated structures and frequencies for better understanding before running the mining algorithm. The second direction is summarizing frequent subgraphs mined from a single large graph where the anti-monotonicity property does not always hold due to the different definitions of

frequency. Therefore, new challenge of avoiding false-positive frequent subgraphs arises. The last direction is to construct a summary which supports subgraph query. Query optimization depends on accurate estimation of the number of query results. Subgraph query estimator is difficult due to the complex structures of graphs, which could have numerous correlations between edges. Moreover, such subgraph query estimator can answer aggregate queries on graphs approximately.

In the middle future, we will focus on OLAP of huge graph collections. OLAP on traditional traction data works well and serves as an important role in business intelligence, while OLAP on graph data is difficult for the following issues. First, the multidimensional model for OLAP on graph data is not clear. Second, answering aggregation queries of graphs is slow, which makes it impossible to handle large graph collections. Third, whether the concept of cube in traditional OLAP can be applied on graph data to utilize material view for saving computation cost is not clear yet. We would like to develop a hybrid multidimensional model consists of both explicit and implicit dimensions for attribute graphs. The explicit dimensions are the dimensions from traditional OLAP for attribute information on graphs. The implicit dimensions are for structure information. Based on this combined approach, we will study on how to answer aggregation queries on graph data efficiently.

BIBLIOGRAPHY

[1] DBLP bibliography. http://www.informatik.uni-trier.de/~ley/db/index.html.

[2] Digg. http://www.cs.cmu.edu/enron/.

[3] Enron email datasets. http://www.cs.cmu.edu/enron/.

[4] Foto Afrati, Aristides Gionis, and Heikki Mannila. Approximating a collection of frequent sets. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '04, pages 12–19, New York, NY, USA, 2004. ACM.

[5] Mohammad Al Hasan and Mohammed J. Zaki. Output space sampling for graph patterns. *Proc. VLDB Endow.*, 2:730–741, August 2009.

[6] Nilesh Bansal, Fei Chiang, Nick Koudas, and Frank Wm. Tompa. Seeking stable clusters in the blogosphere. In *Proceedings of the 33rd international conference on Very large data bases*, VLDB '07, pages 806–817. VLDB Endowment, 2007.

[7] Christian Borgelt and Michael R. Berthold. Mining molecular fragments: Finding relevant substructures of molecules. In *Proceedings of the 2002 IEEE International Conference on Data Mining*, ICDM '02, pages 51–, Washington, DC, USA, 2002. IEEE Computer Society.

[8] Björn Bringmann and Siegfried Nijssen. What is frequent in a single graph? In *Proceedings of the 12th Pacific-Asia conference on Advances in knowledge discovery and data mining*, PAKDD'08, pages 858–863, Berlin, Heidelberg, 2008. Springer-Verlag.

[9] Horst Bunke, Peter J. Dickinson, Miro Kraetzl, and Walter D. Wallis. *A Graph-Theoretic Approach to Enterprise Network Dynamics (Progress in Computer Science and Applied Logic (PCS))*. Birkhauser, 2006.

[10] Orhan Çamoğlu, Tolga Can, and Ambuj K. Singh. Integrating multi-attribute similarity networks for robust representation of the protein space. *Bioinformatics*, 22:1585–1592, July 2006.

[11] Deepayan Chakrabarti. Autopart: parameter-free graph partitioning and outlier detection. In *Proceedings of the 8th European Conference on Principles and Practice of Knowledge Discovery in Databases*, PKDD '04, pages 112–124, New York, NY, USA, 2004. Springer-Verlag New York, Inc.

[12] Deepayan Chakrabarti and Christos Faloutsos. Graph mining: Laws, generators, and algorithms. *ACM Comput. Surv.*, 38(1):2, 2006.

[13] Chen Chen, Cindy Xide Lin, Xifeng Yan, and Jiawei Han. On effective presentation of graph patterns: a structural representative approach. In *Proceeding of the 17th ACM conference on Information and knowledge management*, CIKM '08, pages 299–308, New York, NY, USA, 2008. ACM.

[14] Thomas M. Cover and Joy A. Thomas. *Elements of information theory*. Wiley-Interscience, New York, NY, USA, 1991.

[15] Mukund Deshpande, Michihiro Kuramochi, Nikil Wale, and George Karypis. Frequent substructure-based approaches for classifying chemical compounds. *IEEE Trans. on Knowl. and Data Eng.*, 17:1036–1050, August 2005.

[16] S. N. Dorogovtsev and J. F. F. Mendes. *Evolution of Networks: From Biological Nets to the Internet and WWW (Physics)*. Oxford University Press, USA, March 2003.

[17] Yon Dourisboure, Filippo Geraci, and Marco Pellegrini. Extraction and classification of dense communities in the web. In *Proceedings of the 16th international conference on World Wide Web*, WWW '07, pages 461–470, New York, NY, USA, 2007. ACM.

[18] Mathias Fiedler and Christian Borgelt. Subgraph support in a single large graph. In *Proceedings of the Seventh IEEE International Conference on Data Mining Workshops*, ICDMW '07, pages 399–404, Washington, DC, USA, 2007. IEEE Computer Society.

[19] David Gibson, Ravi Kumar, and Andrew Tomkins. Discovering large dense subgraphs in massive graphs. In *Proceedings of the 31st international conference on Very large data bases*, VLDB '05, pages 721–732. VLDB Endowment, 2005.

[20] Mohammad Al Hasan, Vineet Chaoji, Saeed Salem, Jeremy Besson, and Mohammed J. Zaki. Origami: Mining representative orthogonal graph patterns. In *Proceedings of the 2007 Seventh IEEE International Conference on Data Mining*, pages 153–162, Washington, DC, USA, 2007. IEEE Computer Society.

[21] Thomas Hofmann. Probabilistic latent semantic indexing. In *SIGIR '99: Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 50–57, New York, NY, USA, 1999. ACM.

[22] Jun Huan, Wei Wang, and Jan Prins. Efficient mining of frequent subgraphs in the presence of isomorphism. In *Proceedings of the Third IEEE Inter-*

national Conference on Data Mining, ICDM '03, pages 549–, Washington, DC, USA, 2003. IEEE Computer Society.

[23] Jun Huan, Wei Wang, Jan Prins, and Jiong Yang. Spin: mining maximal frequent subgraphs from graph databases. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '04, pages 581–586, New York, NY, USA, 2004. ACM.

[24] Akihiro Inokuchi, Takashi Washio, and Hiroshi Motoda. An apriori-based algorithm for mining frequent substructures from graph data. In *Proceedings of the 4th European Conference on Principles of Data Mining and Knowledge Discovery*, PKDD '00, pages 13–23, London, UK, 2000. Springer-Verlag.

[25] Glen Jeh and Jennifer Widom. Simrank: a measure of structural-context similarity. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '02, pages 538–543, New York, NY, USA, 2002. ACM.

[26] Glen Jeh and Jennifer Widom. Scaling personalized web search. In *Proceedings of the 12th international conference on World Wide Web*, WWW '03, pages 271–279, New York, NY, USA, 2003. ACM.

[27] Ruoming Jin, Muad Abu-Ata, Yang Xiang, and Ning Ruan. Effective and efficient itemset pattern summarization: regression-based approaches. In *Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '08, pages 399–407, New York, NY, USA, 2008. ACM.

[28] Ravi Kumar, Jasmine Novak, Prabhakar Raghavan, and Andrew Tomkins. On the bursty evolution of blogspace. *World Wide Web*, 8:159–178, June 2005.

[29] Michihiro Kuramochi and George Karypis. Frequent subgraph discovery. In *Proceedings of the 2001 IEEE International Conference on Data Mining*,

ICDM '01, pages 313–320, Washington, DC, USA, 2001. IEEE Computer Society.

[30] Michihiro Kuramochi and George Karypis. Grew-a scalable frequent subgraph discovery algorithm. In *Proceedings of the Fourth IEEE International Conference on Data Mining*, ICDM '04, pages 439–442, Washington, DC, USA, 2004. IEEE Computer Society.

[31] Michihiro Kuramochi and George Karypis. Finding frequent patterns in a large sparse graph. *Data Min. Knowl. Discov.*, 11:243–271, November 2005.

[32] Jure Leskovec and Christos Faloutsos. Scalable modeling of real graphs using kronecker multiplication. In *ICML '07: Proceedings of the 24th international conference on Machine learning*, pages 497–504, New York, NY, USA, 2007. ACM.

[33] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. Graphs over time: densification laws, shrinking diameters and possible explanations. In *KDD '05: Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 177–187, New York, NY, USA, 2005. ACM.

[34] Shirong Li, Shijie Zhang, and Jiong Yang. Dessin: mining dense subgraph patterns in a single graph. In *Proceedings of the 22nd international conference on Scientific and statistical database management*, SSDBM'10, pages 178–195, Berlin, Heidelberg, 2010. Springer-Verlag.

[35] Yu-Ru Lin, Jimeng Sun, Paul Castro, Ravi Konuru, Hari Sundaram, and Aisling Kelliher. Metafac: community discovery via relational hypergraph factorization. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '09, pages 527–536, New York, NY, USA, 2009. ACM.

[36] Yong Liu, Jianzhong Li, and Hong Gao. Summarizing graph patterns. In *Proceedings of the 2008 IEEE 24th International Conference on Data Engineering*, pages 903–912, Washington, DC, USA, 2008. IEEE Computer Society.

[37] Zheng Liu and Jeffrey Xu Yu. Discovering burst areas in fast evolving graphs. In *Proceedings of the 15th international conference on Database systems for advanced applications*, DASFAA'10, pages 171–185, Berlin, Heidelberg, 2010. Springer-Verlag.

[38] Zheng Liu and Jeffrey Xu Yu. Approximate homogeneous graph summarzation. *The Journal of the Association for Information Systems (Invited for Publication)*, 2011.

[39] Zheng Liu and Jeffrey Xu Yu. On summarizing graph homogeneously. In *Proceedings of the 16th international conference on Database systems for advanced applications*, DASFAA'11, pages 299–310, Berlin, Heidelberg, 2011. Springer-Verlag.

[40] Zheng Liu, J.X. Yu, Yiping Ke, Xuemin Lin, and Lei Chen. Spotting significant changing subgraphs in evolving graphs. In *Data Mining, 2008. ICDM '08. Eighth IEEE International Conference on*, pages 917 –922, dec. 2008.

[41] László Lovász. Random walks on graphs: A survey.

[42] Saket Navlakha, Rajeev Rastogi, and Nisheeth Shrivastava. Graph summarization with bounded error. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, SIGMOD '08, pages 419–432, New York, NY, USA, 2008. ACM.

[43] Siegfried Nijssen and Joost N. Kok. A quickstart in frequent structure mining can make a difference. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '04, pages 647–652, New York, NY, USA, 2004. ACM.

[44] Christopher R. Palmer and Christos Faloutsos. Electricity based external similarity of categorical attributes. In *Proceedings of the 7th Pacific-Asia conference on Advances in knowledge discovery and data mining*, PAKDD'03, pages 486–500, Berlin, Heidelberg, 2003. Springer-Verlag.

[45] Jia-Yu Pan, Hyung-Jeong Yang, Christos Faloutsos, and Pinar Duygulu. Automatic multimedia cross-modal correlation discovery. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '04, pages 653–658, New York, NY, USA, 2004. ACM.

[46] Ping Sa Rudolf J. Freund, William J. Wilson. *Regression Analysis: Statistical Modeling of a Response Variable.* Academic Press; 2 edition, 2006.

[47] Boris E. Shakhnovich and J. Max Harvey. Quantifying structure-function uncertainty: A graph theoretical exploration into the origins and limitations of protein annotation. *Journal of Molecular Biology*, 337(4):933 – 949, 2004.

[48] Jimeng Sun, Christos Faloutsos, Spiros Papadimitriou, and Philip S. Yu. Graphscope: parameter-free mining of large time-evolving graphs. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '07, pages 687–696, New York, NY, USA, 2007. ACM.

[49] Lini T. Thomas, Satyanarayana R. Valluri, and Kamalakar Karlapalem. Margin: Maximal frequent subgraph mining. *ACM Trans. Knowl. Discov. Data*, 4:10:1–10:42, October 2010.

[50] Yuanyuan Tian, Richard A. Hankins, and Jignesh M. Patel. Efficient aggregation for graph summarization. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, SIGMOD '08, pages 567–580, New York, NY, USA, 2008. ACM.

[51] Hanghang Tong and Christos Faloutsos. Center-piece subgraphs: problem definition and fast solutions. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '06, pages 404–413, New York, NY, USA, 2006. ACM.

[52] Hanghang Tong, Christos Faloutsos, and Yehuda Koren. Fast direction-aware proximity for graph mining. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '07, pages 747–756, New York, NY, USA, 2007. ACM.

[53] Hanghang Tong, Christos Faloutsos, and Jia-Yu Pan. Fast random walk with restart and its applications. In *Proceedings of the Sixth International Conference on Data Mining*, ICDM '06, pages 613–622, Washington, DC, USA, 2006. IEEE Computer Society.

[54] Hanghang Tong, Spiros Papadimitriou, Philip S. Yu, and Christos Faloutsos. Proximity tracking on time-evolving bipartite graphs. In *Proceeding of the 2008 SIAM conference on Data Mining*, SDM '08, pages 704–715, 2008.

[55] N. Vanetik, E. Gudes, and S. E. Shimony. Computing frequent graph patterns from semistructured data. In *Proceedings of the 2002 IEEE International Conference on Data Mining*, ICDM '02, pages 458–, Washington, DC, USA, 2002. IEEE Computer Society.

[56] Chao Wang and Srinivasan Parthasarathy. Summarizing itemset patterns using probabilistic models. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '06, pages 730–735, New York, NY, USA, 2006. ACM.

[57] Xifeng Yan, Hong Cheng, Jiawei Han, and Dong Xin. Summarizing itemset patterns: a profile-based approach. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, KDD '05, pages 314–323, New York, NY, USA, 2005. ACM.

[58] Xifeng Yan, Hong Cheng, Jiawei Han, and Dong Xin. Summarizing itemset patterns: a profile-based approach. In *KDD '05: Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 314–323, New York, NY, USA, 2005. ACM.

[59] Xifeng Yan and Jiawei Han. gspan: Graph-based substructure pattern mining. In *Proceedings of the 2002 IEEE International Conference on Data Mining*, ICDM '02, pages 721–, Washington, DC, USA, 2002. IEEE Computer Society.

[60] Xifeng Yan and Jiawei Han. Closegraph: mining closed frequent graph patterns. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '03, pages 286–295, New York, NY, USA, 2003. ACM.

[61] Xifeng Yan, Philip S. Yu, and Jiawei Han. Graph indexing based on discriminative frequent structure analysis. *ACM Trans. Database Syst.*, 30:960–993, December 2005.

[62] ChengXiang Zhai, Atulya Velivelli, and Bei Yu. A cross-collection mixture model for comparative text mining. In *KDD '04: Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 743–748, New York, NY, USA, 2004. ACM.

[63] Ning Zhang, Yuanyuan Tian, and Jignesh M. Patel Patel. Discovery-driven graph summarization. In *Proceedings of the 36th international conference on Data Engineering*, ICDE '10, pages 880–891, Long Beach, CA, USA, 2010. IEEE.

[64] Shijie Zhang, Jiong Yang, and Shirong Li. Ring: An integrated method for frequent representative subgraph mining. In *Proceedings of the 2009 Ninth IEEE International Conference on Data Mining*, ICDM '09, pages 1082–1087, Washington, DC, USA, 2009. IEEE Computer Society.