Complex Graph Stream Mining

Shirui Pan

Faculty of Engineering and Information Technology

University of Technology Sydney

A thesis submitted for the degree of

Doctor of Philosophy

October 2015

CERTIFICATE OF AUTHORSHIP/ORIGINALITY

I certify that the work in this thesis has not previously been submitted for a degree nor has it been submitted as part of requirements for a degree except as fully acknowledged within the text.

I also certify that the thesis has been written by me. Any help that I have received in my research work and the preparation of the thesis itself has been acknowledged. In addition, I certify that all information sources and literature used are indicated in the thesis.

Signature of Student

_____

Acknowledgements

I would like to express my earnest thanks to my supervisors, Professor Chengqi Zhang and Professor Xingquan Zhu, who have provided tremendous support and guidance for my research in the past four years. Prof Zhang provided me an opportunity to study in the stimulating and interactive centre for Quantum Computation and Intelligent Systems (QCIS), where I met and leant a lot from many smart and sharp people. I benefit significantly from his unselfish help and invaluable suggestion on my research career. I would like to thank Prof Xingquan Zhu for his continuous guidance and supervisions during my Ph.D. study. Discussing a problem with him has been always a pleasure and eye-opening experience. He always gives me sufficient freedom and encouragement to think and explore my research interest. His vision, creativeness and enthusiasm in solving challenging problems has greatly encouraged me and inspired my works. Without his endless patience, generous support, and constant guidance, this thesis could not have been accomplished.

I would also like to thank all the people that had a positive influence on my day-to-day enjoyment of the job. My office-mates, past and present: Jia Wu, Yifan Fu, Guodong Long, Jing Jiang, Peng Zhang, Tianyi Zhou, Wei Bian, Wei Wang, Xun Wang, Ting Guo, Lianhua Chi, Meng Fang, Mingsong Mao, Zhibin Hong, Hongshu Chen, Shaoli Huang, Haishuai Wang, Mingming Gong, Sujuan Hou, Qin Zhang, Maoying Qiao, Zhiguo Long, Hua Meng, Zhe Xu, Bozhong Liu, Tongliang Liu, Junyu Xuan, and Jiang Bian. They are the ones who have given me support during both joyful and stressful times, to whom I will always be thankful.

I also wish to express my appreciation to the financial support I gained for my study. Special thanks go to China Scholarship Council (CSC), University of Technology Sydney (UTS), centre for Quantum Computation and Intelligent Systems (QCIS), ICDE student travel grant, and CIKM student travel grant.

Finally, and above all, I want to thank my family for their continuous support. I especially thank my wife, Yu Zheng, who took care of the daily life of our little baby, Yixin Pan, and myself, and shared all my pain, sorrow and joy in every moment of my research. I would like to thank my parents, brothers, and sisters for their unconditional encouragement and support, both emotionally and financially. No words could possibly express my deepest gratitude for their endless love, self-sacrifice and unwavering help. To them I dedicate this dissertation.

Abstract

Recent years have witnessed a dramatic increase of information due to the ever development of modern technologies. The large scale of information makes data analysis, particularly data mining and knowledge discovery tasks, unprecedentedly challenging. First, data is becoming more and more interconnected. In a variety of domains such as social networks, chemical compounds, and XML documents, data is no longer represented by a flat table with instance-feature format, but exhibits complex structures indicating dependency relationships. Second, data is evolving more and more dynamically. Emerging applications such as social networks continuously generate information over time. Third, the learning tasks in many real-life applications become more and more complicated in that there are various constraints on the number of labelled data, class distributions, misclassification costs, or the number of learning tasks etc.

Considering the above challenges, this research aims to investigate theoretical foundations, study new algorithm designs and system frameworks to enable the mining of complex graph streams from three aspects, including (1) Correlated Graph Stream Mining, (2) Graph Stream Classifications, and (3) Complex Task Graph Classification.

In particular, correlated graph stream mining intends to carry out structured pattern search and support the query of similar graphs from a graph stream. Due to the dynamic changing nature of the streaming data and the inherent complexity of the graph query process, treating graph streams as static datasets is computationally infeasible or ineffective. Therefore, we proposed a novel algorithm, CGStream, to identify correlated graphs from a data stream, by using a sliding window, which covers a number of consecutive batches

of stream data records. Experimental results demonstrate that the proposed algorithm is several times, or even an order of magnitude, more efficient than the straightforward algorithms.

Graph stream classification aims to build effective and efficient classification models for graph streams with continuous growing volumes and dynamic changes. We proposed two methods for complex graph stream classification. Due to the inherent complexity of graph structure, labelling graph data is very expensive. To solve this problem, we proposed a gLSU algorithm, which aims to select discriminative subgraph features with minimum redundancy by using both labelled and unlabelled graphs for graph streams. The second approach handles graph streams with imbalanced class distributions and noise. Both frameworks use an instance weighting scheme to capture the underlying concept drifts of graph streams and achieve significant performance gain on benchmark graph streams.

Complex task graph classification aims to address the graph classification problems with complex constraints. We studied two complex task graph classification problems, cost-sensitive graph classification of large-scale graphs and multi-task graph classification. As in medical diagnosis the misclassification cost/risk for different classes is inherently different and large scale graph classification is highly demanded in real-life applications, we proposed a CogBoost algorithm for cost-sensitive classification of large scale graphs. To overcome the limitation of insufficient labelled graphs for a specific learning task, we further proposed effective algorithms to leverage multiple graph learning tasks to select subgraph features and regularize multiple tasks to achieve better generalization performance for all learning tasks.

Contents

List of Figures

List of Tables

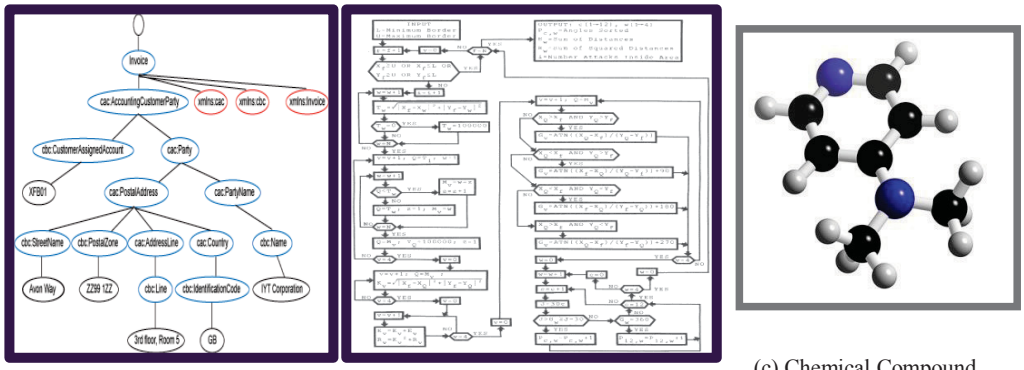
Chapter 1

Introduction

1.1 Motivations and Significances

Data mining is a fundamental task and has drawn increasing interest in the last decade, due to the ever increasing of gigantic data and the demand for discovering useful information from large scale data. Traditionally, data mining algorithm performs on data represented by a flat table with instance-feature format. However, due to the rapid development of electronic devices, networking, and social media technologies, recent years have witnessed an increasing number of applications where data are no longer represented in simple instance-feature format, but exhibit as complex network structures indicating dependency relationships. Typical examples (some are illustrated in Figure 1.1) include an XML webpage (*i.e.* an instance) which points to (or is pointed to) several other XML webpages [116], a scientific publication with a number of references [2], posts and responses generated from social networking sites (*e.g.* Tweets generated from Twitter [114]), and chemical compounds with molecules (*i.e.* nodes) linked together through some bounds (*i.e.* edges) [29, 40]. To discover knowledge from such network of information, **graphs** are natural tools to model and capture the relationships in the networks.

In real-life applications, data are becoming more and more dynamic. Instead of being a static dataset, data are generated and evolving dynamically. Such continuous flow of data is known as **data streams** [3, 5, 60]. Mining from data



(a) XML Documents (b) Program Flows (c) Chemical Compound

Figure 1.1: Examples of graphs for different applications.

streams usually has high requirements in terms of memory and time consumption. On the one hand, the system for handling data streams is never large enough to store all the streaming data for multiple scanning. On the other hand, most streaming scenarios need the results in a timely fashion. Another challenge for mining from data streams is that the underlying data distribution is gradually or rapidly changing over time, which is known as *concept drifts* [133, 141]. Concept drifts impose vast difficulty on the classification task of data streams because the decision concept will gradually or suddenly change over time, which means that a learned model of historical data will become outdated and need to be revised very soon.

The situations become more complicated when the learning tasks at hand are subject to various restrictions on the number of labeled data, distributions, misclassification cost, or number of learning tasks. Due to the inherent complexity of the graph data and the costs involved in the labeling process, collecting a large number of labeled graphs for a specific task is expensive [7]. Furthermore, the labeled graphs data in many scenarios usually have imbalanced class distributions and the misclassification costs are unequal for different classes of data.

Considering the interconnected characteristic of networks, the dynamic nature of streams, and the inherent complexity of learning tasks, this research aims to focus on **complex graph stream mining**, which essentially enriches the research of **big data** mining [148] in three perspectives (volume, velocity, and variety, that is, the"3V" properties of big data). From the view of variety, this research

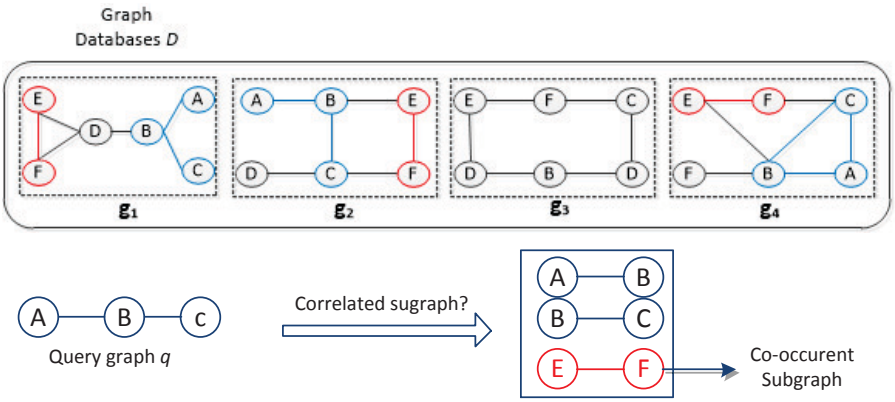


Figure 1.2: An illustrated example of correlated graph search. Given a query graph, correlated graph search can return some co-occurence subgraph patterns to the query. For instance, E-F simultaneously appears with the query graph A-B-C in the original graph database.

extends several fundamental data mining problems from traditional vector data to more complex structure data. From the perspectives of volume and velocity, this thesis proposes effective graph stream algorithms for handling dynamically increasing and changing graph data. The research explores three tasks of complex graph stream mining: (1) Graph Stream Query, (2) Graph Stream Classification, and (3) Complex Task Graph Classification. All of them are well motivated in a variety of applications.

- **Continuous Correlated Graph Query for Data Streams.** The internet has produced massive structure data representing users' browsing patterns in recent years. In this scenario, each user's browsing history in a large website can be represented as a network (graph). While user browsing patterns are dynamically flowing and evolving over time, a useful task is to discover some correlated graphs from the graph streams. The correlated graphs reveal the co-occurrent patterns shared by different users. Such group of users usually shares some common interests. Analyzing correlated graphs helps website owners understand user behaviors so that they can improve the website structures, and detect abnormal behaviors, which are very important in E-commerce [74]. The correlated graph query problem can be modeled as in Figure 1.2.

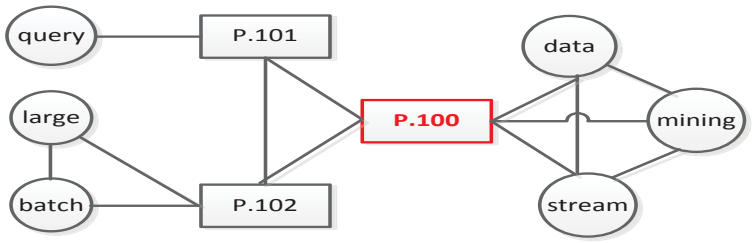


Figure 1.3: Graph representation for a scientific paper (P.100) in DBLP. The rectangles are paper ID nodes and circles are keyword nodes. Paper P.100 cites (connects) paper P.101 and P.102, and P.100 has keywords *Data*, *Stream*, and *Mining* in its title. Paper P.101 has keyword *Query* in its title, and P.102's title includes keywords *Large* and *Batch*. For each paper, the keywords in the title are linked to each other. More information is given in Section 3.3.

- **Graph Stream Classification.** Numerous applications have emerged in recent years calling for dynamic graph stream classification. For instance, in a scientific network, each publication can be represented as a network (graph), where nodes are papers or keywords and edges are citations between papers or keyword relationships. A typical publication in DBLP dataset modeled as a graph is illustrated in Figure 1.3 . The scientific publications are continuously being published, and therefore increase dynamically over time as streams. A classification task can help automatically classify a publication into several categories, such as database, data mining, and computer vision.

- **Complex Task Graph Classification.** Real-life graph classification tasks may have other complicated restrictions. Typical applications include:

    1. *Cost-sensitive Graph Classification.* In this case, the misclassification cost for different classes are inherently different. For instance, in structure based medical diagnose [29, 131], chemical compounds active against cancer are very rare and are expected to be carefully monitored and identified. A false negative identification (that is, predicting an active compound to be inactive) has a much more severe consequence (that is, a higher cost) than a false positive identification (predicting an inactive compound to be active). Therefore, a false negative and a false positive are inherently different and a false negative prediction

may result in delay and wrong diagnosis, leading to severe complications (or extra costs) at a later stage.

2. *Multi-task Graph Classification.* In many cases, several relevant graph learning tasks may co-exist and each has a rather limited number of training graphs. An example is given as follows:

   Chemical Compound Categorization is important in biomedical research for testing whether a chemical compound is active to a specific cancer, such as melanoma. For melanoma cancer, determining activities of a molecule is expensive as it requires time, effort, and expensive resources [7] to conduct biological assay. In reality, some similar bioassay tasks [1], such as an anti-cancer test for prostate, may be available. As the graph data for different types of cancer may share common substructures, learning multiple related tasks together may potentially help improve the generalization performance of each single task.

The complex task graph classification is important but as yet has not been studied in literatures.

1.2 Research Problems

As each of the above tasks is essential in real-life applications, our research examines the unique challenge of each task carefully and exploits several research problems accordingly.

1.2.1 Graph Stream Search

For graph stream search, we are given a streaming of graphs $\mathcal{S}$, and a query graph g_q; the objective is to discover a set of subgraphs that are correlated to query graph g_q. The correlated graph query in a data stream setting is essentially a challenge in the sense that: (1) the graph stream dynamically increases and changes over time, and (2) the correlated graph search needs the results in a timely fashion.

[1] https://pubchem.ncbi.nlm.nih.gov/

1.2.2 Graph Stream Classification

Semi-supervised Graph Stream Classification: For graph stream classification, we are given a collection of graph data with class labels; its objective is to build a prediction model with maximum accuracy in classifying previously unseen graph streams. Due to the complexity of network structures, labeling graphs usually requires experts to investigate the structures carefully. In order to reduce the human resource of labeling graphs, a possible way is to combine both labeled and unlabeled graphs to construct classifier models. How to use both labeled and unlabeled graphs to perform graph classification in a streaming scenario is one of the research problems in our research.

Imbalanced Graph Stream Classification: Meanwhile, in real-life applications, especially for the graph based domain, the class distribution of data is inherent imbalanced. In the NCI chemical compound databases, there are only about 5% percent of chemical compounds which are active to the cancer bioassay test, whereas 95% of them are inactive to the cancer bioassay test. Classification for imbalanced graph streams is another research problem in our research.

1.2.3 Complex Task Graph Classification

This thesis also considers the following complex task graph classification:

Cost-sensitive Learning for Large Scale Graphs: For graph classification, all existing methods assume, explicitly or implicitly, that misclassifying a positive graph incurs an equal amount of cost/risk to the misclassification of a negative graph, *i.e.*, all misclassifications have the same cost. The induced decision rules are commonly referred to as being *cost-insensitive*. In real-life graph applications, the equal-cost assumption is mostly invalid (or at least too strong).

Another challenge of existing graph classification algorithms is that they are only designed for small size graph datasets and are inefficient to scale up to large size graph datasets.

In summary, how to conduct cost-sensitive learning for large scale graphs is an important research topic of this thesis.

Multi-task graph classification: Existing graph classification has been advanced to many complicated settings, such as multi-label [76] or multi-graph

learning [144], but all these methods are designed to handle single learning tasks. In reality, several relevant graph learning tasks may co-exist and each has a rather limited number of training graphs. How to jointly learn the multiple graph classification task is also investigated in our thesis.

1.3 Thesis Contributions

The thesis addresses a number of fundamental problems of *complex graph stream mining* from three aspects, including graph stream search, graph stream classification, and complex task graph classification. The main contributions of this study can be summarized in three parts, accordingly.

1.3.1 Graph Stream Search

Contribution:

- This thesis proposed to discover the correlated subgraph patterns from dynamic graph streams. The proposed algorithm, CGStream, is several times, or even an order of magnitude, more efficient than an exhaustive search method.

Outcome:

- The CGStream algorithm was published in CIKM-2012 [107]. We further explored its variants, top-K based correlated subgraph search for graph streams, and published a poster paper in CIKM-2012 [108] and ICPR-2012 [110].

1.3.2 Graph Stream Classification

Contributions:

- We proposed to use both labeled and unlabeled graphs for effective subgraph feature selection for graph stream classification.
- We proposed a novel graph stream classification algorithm for imbalanced and noisy graph data.

Outcome:

- Our semi-supervised feature exploration algorithms for graph stream classification was published in ICDE-2013 [101]. The techniques used in the paper [101] have inspired a number of joint works on more complicated graph classification settings, such as multi-graph classification [142, 143], multi-graph-view classification [146], and multi-view multi-instance feature selection [145].
- Our imbalanced graph classification algorithm was first published in IJCAI-2013 [109], and then the streaming version algorithm was latter published in TCYB-2015 [104].

1.3.3 Complex Task Graph Classification

Contributions:

- We proposed an effective algorithm for cost-sensitive learning for large scale graphs.
- We proposed a jointly regularized subgraph feature exploitation method for multi-task graph classifications.

Outcome:

- Our cost-sensitive learning algorithm for large scale graph data was published in TKDE-2015 [102].
- Our multi-task graph classification learning algorithm is currently under review by TKDE. The technique used for this part, has been successfully applied to single task graph classification published in Pattern Recognition [103].

1.4 Thesis Overview

The thesis is structured into three parts; these are graph stream query, graph stream classification, and complex task graph classification. Table 1.1 summarizes

the overall structure of our research with the mapping to the chapters of this thesis. The detailed roadmap of the thesis is summarized as follows:

Table 1.1: Structure of the thesis with reference to the chapters.

Part	Research Task	Chapter	Related Publication
—	Literature Review & Preliminary	Chapter 2, 3	
Part I Graph Stream Search	Correlated Graph Search	Chapter 4	[107]
Part II Graph Stream Classification	Semi-supervised Graph Stream Classification	Chapter 5	[101]
	Imbalanced & Noisy Graph Stream Classification	Chapter 6	[104, 109]
Part III Complex Graph Classification	Cost-sensitive & Large Scale Graph Classification	Chapter 7	[102]
	Multi-task Graph Classification	Chapter 8	
—-	Conclusions and Future Work	Chapter 9	

Chapter 2: Before formally studying solutions to address research problems, we review all related works from different aspects, including graph query, graph classification, imbalanced data classification, data stream and graph stream classification, cost-sensitive learning, and multi-task learning.

Chapter 3: This chapter provides preliminary and common definitions for the proposed models. It also summarizes all the datasets used in the thesis.

Part I (Chapter 4): Part I presents our algorithm, CGStream, for continuous correlated graph search over data streams. The algorithm returns a query graph's correlated graphs in a sliding window which covers a number of consecutive batches of stream records.

Part II: Chapter 5 describes the proposed gLSU algorithm which uses both labeled and unlabeled graphs for graph stream classification. The proposed algo-

rithm selects a set of discriminative subgraphs with minimum redundancy to learn a classifier from each streaming batch, and employs a dynamic instance weighting mechanism to handle the concept drift in graph streams. Chapter 6 details our algorithm for handling imbalanced and noisy graph streams. A boosting framework on each graph batch is proposed for imbalanced graph data, followed by an instance weighting scheme to capture the underlying concept drift over streams.

Part III: Chapter 7 depicts our CogBoost algorithm for handling cost-sensitive graph classification with large scale graphs. The proposed algorithm implements the Bayes optimal loss and scales to large scale graph data. Chapter 8 considers how to jointly learn multiple graph classification task to improve the generalization ability.

Note that Part III mainly focuses on complex task graph classification. However, these methods can be easily extended to streaming scenarios, following the frameworks used in Part II for semi-supervised graph stream classification (Chapter 5) or imbalanced graph stream classification (Chapter 6).

Chapter 9: We summarize the whole thesis and point out several future directions of this study in Chapter 9.

1.5 Publications

Below is a list of the papers associated with my PhD research that have been submitted, accepted, and published:

Journal Publications:

1. **Shirui Pan**, Jia Wu, Xingquan Zhu. CogBoost: A Fast Cost-sensitive Graph Boosting Algorithm. *IEEE Transactions on Knowledge and Data Engineering (**TKDE**)*, **Accepted**, 2015. (Australia ERA Ranked **A**)

2. **Shirui Pan**, Jia Wu, Xingquan Zhu, and Chengqi Zhang. Graph Ensemble Boosting for Imbalanced and Noisy Graph Stream Classification. *IEEE Transactions on Cybernetics (**TCYB**)*, vol 45, no 5, pp 940-954, 2015. (Australia ERA Ranked **A**)

3. **Shirui Pan**, Jia Wu, Guodong Long, Xingquan Zhu, Chengqi Zhang. Finding the Best Not the Most: Regularized Loss Minimization Subgraph Selection for Graph Classification. *Pattern Recognition (**PR**)*, vol 48, no 11, pp 3783-3796, 2015. (Australia ERA Ranked **A***)

4. **Shirui Pan**, Jia Wu, Xingquan Zhu, Chengqi Zhang, and Philip S. Yu. Joint Structure Feature Exploration and Regularization for Multi-Task Graph Classification. *IEEE Transactions on Knowledge and Data Engineering (**TKDE**)*. (Australia ERA Ranked **A**)

5. **Shirui Pan**, Jia Wu, Xingquan Zhu, and Chengqi Zhang. Boosting for Graph Classification with Universum Graphs. *Knowledge and Information System (**KAIS**)*, **under review**. (Australia ERA Ranked **B**)

6. Jia Wu, **Shirui Pan**, Xingquan Zhu, Zhihua Cai. Boosting for Multi-Graph Classification. *IEEE Transactions on Cybernetics (**TCYB**)*, vol 45, no 3, pp 430-443, 2015. (Australia ERA Ranked **A**).

Conference Publications:

7. **Shirui Pan**, Xingquan Zhu. Graph Classification with Imbalanced Class Distributions and Noise. *23rd International Joint Conference on Artificial Intelligence (**IJCAI**)*, 2013. (Australia ERA Ranked **A**)

8. **Shirui Pan**, Xingquan Zhu, Chengqi Zhang, and Philip S. Yu. Graph Stream Classification using Labeled and Unlabeled Graphs. *International Conference on Data Engineering (**ICDE**)*, 2013. (Australia ERA Ranked **A**)

9. **Shirui Pan** and Xingquan Zhu. CGStream: Continuous Correlated Graph Query for Data Streams. *21st ACM International Conference on Information and Knowledge Management (**CIKM**)*, 2012. (Australia ERA Ranked **A**)

10. **Shirui Pan** and Xingquan Zhu. Continuous Top-k Query for Graph Streams. *21st ACM International Conference on Information and Knowledge Management (**CIKM**)*, 2012. (Australia ERA Ranked **A**)

11. **Shirui Pan** and Xingquan Zhu. Top-k Correlated Subgraph Query for Data Streams. *21st International Conference on Pattern Recognition (**ICPR**)*, 2012. (Australia ERA Ranked **B**)

12. Jia Wu, Zhibin Hong, **Shirui Pan**, Xingquan Zhu, Chengqi Zhang. Multi-Graph-View Learning for Graph Classification. *IEEE International Conference on Data Mining (**ICDM**)*. 2014. (Australia ERA Ranked **A**)

13. Jia Wu, Zhibin Hong, **Shirui Pan**, Xingquan Zhu, Zhihua Cai, Peng Zhang, Chengqi Zhang. Exploring Features for Complicated Objects: Cross-View Feature Selection for Multi-Instance Learning. *ACM International Conference on Information and Knowledge Management (**CIKM**)*. 2014. (Australia ERA Ranked **A**)

14. Jia Wu, Zhibin Hong, **Shirui Pan**, Xingquan Zhu, Chengqi Zhang, and Zhihua Cai. Multi-Graph Learning with Positive and Unlabeled Bags. *SIAM International Conference on Data Mining (**SDM**)*, 2014. (Australia ERA Ranked **A**)

Chapter 2

Literature Review

This thesis studies several fundamental problems for complex graph stream mining. As a result, this work is closely related to a variety of works from different aspects: correlated graph search, graph classification, imbalanced data classification, data stream and graph stream classification, cost-sensitive learning, and multi-task learning. In this chapter, we review the related work accordingly.

2.1 Correlated Graph Search

Mining correlation has been widely studied in various domains in the literature. For market-basket database, extensive studies have addressed the correlation between items [150, 165]. While these methods were proposed to identify correlation defined by Pearson's correlation coefficient, some other measures such as χ^2 test [18], h-confidence [149], and m-pattern measure [89] are also investigated in the community.

In the context of static graph databases, there are several works related to correlation mining. Given a query graph, CGSearch [71] mines the correlated graphs whose correlation is above a given threshold, TopCor [74] discovers the top-k correlated graphs with the highest correlation in a graph database. These two works carry out the query based on a given query graph g_q, whereas the work in [73] does not require users to specify any query. In other words, it tries to find out all correlated graph pairs in a database. In comparison, all existing works

in this category only limit their scopes to static databases, whereas our work is designed for dynamic graph streams.

In data stream environments, recently there are some works on graph search [138] and closed frequent subgraph mining [15]. But to the best of our knowledge, no existing works/studies exist for correlated graph query for data streams.

2.2 Graph Classification

As graphs involve node-edge structures whereas most existing learning algorithms use an instance-feature representation model, the major challenge of graph classification is to transfer graphs into proper format for learning methods to train classification models. Existing methods for graph classification [24, 40, 65, 70, 75, 76, 117, 129, 152, 168] can be roughly categorized into two groups: similarity-based methods and vector representation-based methods.

2.2.1 Similarity-based methods

These approaches aim to directly learn global similarities between graphs by using graph kernels [12, 70, 98, 122] or graph embeddings [118]. Global similarities are then fed to similarity-based classifiers, such as KNN or SVM, for learning. One clear drawback of global similarity-based approaches is that the similarity is calculated based on global graph structures, such as random walks or embedding space. Therefore, it is not clear which substructures are more important for classifying graphs between different classes.

2.2.2 Vector representation-based methods

Another branch of methods transfer graphs into vector representations in structure space or in Euclidean space. In structure space [62, 63], geometrical and analytical concepts such as the angle between structures and the derivatives of functions on structures can be obtained, so that the structural pattern recognition problems can be formulated as optimization problems with certain cost functions. In Euclidean space, the goal is to transfer graphs into vector representations in Euclidean space so existing analytical techniques can be applied for data analysis.

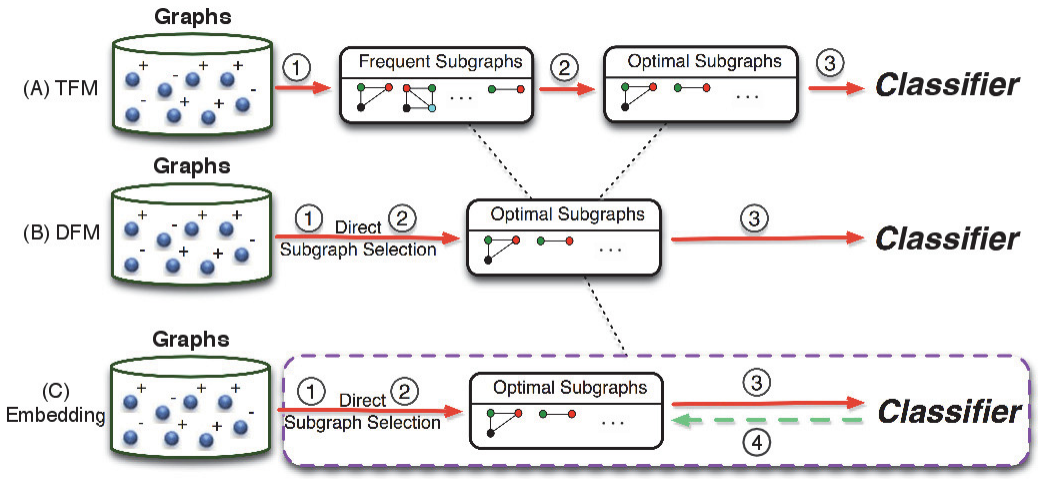


Figure 2.1: Subgraph-based methods for graph classification from the feature selection perspective. TFM methods (A) sequentially perform frequent subgraph mining ①, optimal feature selection ②, and classifier learning process ③. DFM methods (B) integrate the feature selection ② into the frequent subgraph mining ① process. Embedding methods (C) unifies all steps (①②③) into a whole framework, and iterates until convergence ④.

Methods to transfer graphs into Euclidean space, as shown in Fig. 2.1, can be grouped into three categories, including two-step filter methods (**TFMs**), direct filter methods (**DFMs**), and embedded methods (**Ems**).

Two-step filter methods (TFM): TFMs are straightforward approaches for graph classification which simply decompose frequent subgraph generation and selection as two separated steps. An early work [70] has shown that learning an SVM classifier based on the discovered frequent subgraphs can achieve reasonably good accuracy for graph classification. On the other hand, research [117, 129] also indicates that TFM methods may result in a bottleneck for the subsequent feature selection module. Specifically, the number of frequent subgraphs will grow exponentially if the minimum support threshold is low, which imposes a great challenge for the subsequent feature selection task. This challenge has motivated many direct filter methods (**DFMs**), which seek to integrate subgraph discovery and feature selection into one step.

Direct filter methods (DFMs): For DFMs (a review on this category can be found in [24]), a key issue is to define a proper measurement to assess the

utility of each subgraph. Yan *et al* [152] proposed a LEAP algorithm to exploit the correlation between structural similarity and significance similarity, so that a branch-and-bound rule can be derived to prune out unpromising searching space efficiently. Ranu and Singh [117] proposed a scalable GraphSig algorithm, which is able to mine the significant subgraph with low frequencies. Thoma *et al.* [129] propose a CORK algorithm to find subgraph features. Recently, researchers have extended the DFM method to other graph applications, and have proposed effective algorithms such as gSemi [75] for the semi-supervised setting, gCGVFL [146] for multi-view learning, gHSIC [76] for multi-label classification, and our recent multi-graph classification for classifying graph bags, each containing multiple graphs [142, 143].

Although filter methods for graph classification have been extensively studied, they all suffer from two major disadvantages: (1) the feature selection is not linked to the model learning process. As a result, the selected subgraph features may not best fit the underlying learning algorithms; and (2) the optimal number of subgraphs K for graph classification is difficult to decide and often varies from dataset to dataset, and inappropriately specified K value often results in significantly reduced classification accuracy. This is the common drawback for filter-based methods [54].

Embedded Methods (Ems): Embedded approaches for graph classification integrate the subgraph selection into the model training process. In this subcategory, Saigo *et al* [120] proposed a gBoost algorithm which formulates the graph classification as a linear program. The gBoost algorithm can be considered as a ℓ_1 regularized hinge loss classification for graph data. We propose a regularized loss minimization driven (RLMD) subgraph selection algorithm for graph classification [103], which is more general in the sense that it can adopt any differentiable loss function and use more robust regularization to produce better performance. Furthermore, because gBoost only considers applications with balanced class distributions or scenarios with equal misclassification cost/risk, we extend gBoost to handle imbalanced graph data in [109], to imbalanced graph streams in [104], and to cost-sensitive classification scenarios with large scale graph data in [102].

2.2.3 Graph-based Learning for a Single Network

It is worth noting that the graph classification considered in our thesis is different from a number of graph-based learning studies for a single network. Recent years have witnessed increasing research for graph-based learning in a single network. Instead of considering samples as *I.I.D* observations, graph-based learning takes the relationships/correlations between samples to ensure effective learning. For example, graph-based approaches have been popularly used to propagate labels in semi-supervised learning [27, 69, 90], where training samples are connected through one or multiple graphs. A recent method [84] considers preserving global and local structures inside the training data for feature selection. For large scale networks, predicting linkage relationships between nodes (that is, link prediction) can be used for friendship recommendation in social networks [82], or suggesting potential interactions between proteins in bioinformatics research. A recent work [100] proposed to use latent feature kernels to support link prediction on sparse graphs. All the above methods consider a large scale network with thousands (or millions) of integer-connected nodes in the network. In contrast, we consider the small graph classification problem, in which each graph has a label indicating the property of the graph, and the graph normally contains tens or several hundreds of nodes. The purpose is to predict the label of the graph by using node and structure information inside the graphs, for purposes such as chemical compound activity prediction [29] and gender classification using magnetic resonance connectome (that is, brain-graph) [136].

2.3 Imbalanced Data Classification

Many methods exist for imbalanced data classification, including sampling [85], ensembling [45, 80], and support vector machine adapting [6, 135]. Some recent reviews and monographs on imbalanced data classification are also available [55, 56, 125]. For all these methods, their scope is limited to data in vector format. When dealing with imbalanced graph data, a simple solution is to use under-sampling to create a relatively balanced graph set, and then apply existing graph classification methods [75, 117]. This solution has shown positive results

on general data [134], but in the graph domain, it will result in significant performance deterioration, because it not only ignores the structure information in the graph datasets, but is also subject to the risk of losing valuable information in the sampled data and causes a large angle between the ideal and learned hyperplane for margin based algorithms (*i.e.*, SVM) [6]. This problem will be further aggravated with the presence of noise (*i.e.* mislabeled samples). Because noise accounts for a small portion of the whole dataset, they are similar to instances in the minority class. As a result, any solutions trying to emphasize samples in the minority class to achieve performance gain may falsely emphasize on noise and suffer severe performance loss instead.

In our thesis (Chapter 6), we will consider the unique challenges of graph data, and propose a novel algorithm to handle imbalanced and noisy data.

2.4 Data Stream and Graph Stream Classification

The task of data stream classification [4, 5, 33, 105, 106, 124, 139, 157, 158, 159, 166] is to build predictive models for data with dynamic changing volumes. One important issue for stream classification is to handle concept drifting, and common approaches are to employ an ensemble model to accommodate changes over stream [139, 166] or to actively detect changes [10] and retrain models accordingly. Some recent works have considered both stream classification and data imbalance [23, 31, 46, 140]. In [31], the authors proposed a Learn++ framework with two variants, Learn++.CDS and Learn++.NIE, for imbalanced concept drifting data streams. To handle data imbalance, Learn++.CDS employs SMOTE [22] to synthetically generate minority class samples based on the vector data. Learn++.NIE, on the other hand, uses a weighted ensemble approach to combat concept drifting in the stream. Intuitively, one can use Learn++ to handle imbalanced graph streams by using a set of frequent subgraphs to transfer graph stream into vector format and by applying Learn++.CDS to the vector data, or integrating existing gBoost algorithm [120] as a base graph classifier into Learn++.NIE. However, all these straightforward solutions may fail to iden-

tify discriminative features for imbalanced graphs, and eventually lead to inferior accuracy.

Graph Stream Classification: Data stream classification has been recently extended to structural graph data [2, 25, 52, 53, 81, 101]. In [2], the authors proposed to hash graph edges into random numbers and used discriminative edges as patterns for classification. In [81], Li *et. al* proposed a fast subtree kernel based algorithm to enable graph stream classification. As graph stream is dynamically evolving with different subtree patterns emerging in different chunks, some works proposed to project subtree patterns of different chunks onto a set of common low-dimensional feature spaces by using hashing algorithm [25, 52]. In our thesis, we extend the graph stream classification in a semi-supervised setting (Chapter 5), with both labeled and unlabeled graphs being used to find discriminative subgraphs with minimum redundancy. We also considers both data imbalance and noise, and presents a stream based algorithm for graph classification in Chapter 6. A recent work [53] proposed to classify nodes in a large streaming network, which is essentially different from ours in that we aim to classify a collection of small graphs.

2.5 Cost-sensitive Learning

Cost-sensitive learning has been extensively studied in the last decade. Approaches for cost-sensitive learning can be mainly distinguished into the following four categories: (1) Sampling methods [156]; (2) Decision tree approaches [86, 160]; (3) Boosting algorithms [36, 87, 92]; and (4) SVM adaptation [91, 135].

Sampling approaches [156] aim to re-weight the training samples in proportion to their cost values, which can be done by over-sampling, or cost-proportionate rejection sampling. The main goal is to change the sample distributions so that any classifier can be directly used to handle cost-sensitive problems. Decision tree modelling approaches [86, 160] incorporate the costs during the tree construction, so that misclassification cost at the leaves is minimized. Boosting algorithms [36, 87, 92], such as AdaCost [36], use the misclassification costs to update the training distributions on successive boosting rounds, which has been proved to be effective in reducing the upper bound of cumulative misclassification

cost of the training set. SVM adaptation [91, 135] represents a set of approaches based on SVM adaptation for cost-sensitive learning. They either shift the decision boundaries by simply adjusting the threshold of standard SVMs [121] or introduce different penalty factors C_1 and C_{-1} for the positive and negative SVM slack variables during training [135]. Recently a CS-SVM algorithm [91] is proposed which utilizes an optimal hinge loss function. It is shown in [91] that CS-SVM outperforms previous approaches [121, 135].

In summary, the scope of all existing cost-sensitive methods is limited to data in vector format. In this thesis, the unique challenges of graph data are considered, and a novel algorithm for cost-sensitive graph classification is proposed (Chapter 7).

2.6 Multi-task Learning

State-of-the-art algorithms on multi-task learning [8, 21, 35, 41, 57, 68, 95, 161, 162, 164] can be roughly divided into two categories: (1) multi-task feature learning, which explores common feature space shared by all tasks. The models, including mixed $\ell_{2,1}$ norm sparsity inducing methods [8, 83], composite regularized algorithms [48, 64], and the most recent calibration based multi-task approach [49], can be formulated as a regularized loss minimization problem aiming to explore shared feature space among tasks for learning; and (2) task relationship learning, which simultaneously exploits task relationships and parameters [161], such as task clustering [61, 78, 163] or isolating [119], so that knowledge can be shared by a group of tasks instead of all tasks.

Note that multi-task learning is closely related to transfer learning [39, 112], but the difference is fundamental. Transfer learning aims to improve the learning on a single target task by using data from other tasks as auxiliary information. For multi-task learning, all tasks are equally important and should be learned simultaneously.

2.7 Key techniques

In our thesis, we frequently use some techniques that are very useful for subgraph mining and convex optimization, including gSpan algorithm, column generation algorithm, and cutting plane algorithm.

2.7.1 gSpan Algorithm

Frequent subgraph mining is the key for many subgraph mining problem. In our thesis, we develop effective algorithms for complex graph stream mining based on a successful frequent subgraph mining algorithm gSpan [151].

Given a collection of graphs and a minimum support threshold, gSpan [151] is able to find all of the subgraphs whose frequency is above the threshold. To achieve this goal, gSpan builds a new lexicographic order among graphs, and maps each graph to a DFS code as its canonical label. Based on this lexicographic order, gSpan adopts the depth-first search strategy to mine frequent connected subgraphs efficiently.

One issue for frequent subgraph mining problem is that there are exponential subgraph patterns and many of them are duplicated. How to avoid enumerate duplicated subgraphs is a key challenge for this problem. To cope with this challenge, gSpan algorithm defines unique minimum DFS code for each subgraph pattern. Two subgraph are isomorphism if and only if they have the same DFS codes. By this way, gSpan can prune duplicated subgraph patterns effectively.

In this thesis, gSpan algorithm will be used in Chapter 4, 5,6, 7, and Chapter 8.

2.7.2 Column generation Algorithm

Column generation [97] is an efficient algorithm for solving larger linear programs.

The overarching idea is that many linear programs are too large to consider all the variables explicitly. Since most of the variables will be non-basic and assume a value of zero in the optimal solution, only a subset of variables need to be considered in theory when solving the problem. Column generation leverages this idea to generate only the variables which have the potential to improve the

objective function-that is, to find variables with negative reduced cost (assuming without loss of generality that the problem is a minimization problem).

The problem being solved is split into two problems: the master problem and the subproblem. The master problem is the original problem with only a subset of variables being considered. The subproblem is a new problem created to identify a new variable. The objective function of the subproblem is the reduced cost of the new variable with respect to the current dual variables, and the constraints require that the variable obey the naturally occurring constraints.

In this thesis, column generation will be used in Chapter 6 and Chapter 7.

2.7.3 Cutting Plane Algorithm

Cutting plane algorithms are also very popular for solving large scale machine learning problem [66]. They are popularly used for non-differentiable convex minimization, where a convex objective function and its subgradient can be evaluated efficiently but usual gradient methods for differentiable optimization can not be used. This situation is most typical for the concave maximization of Lagrangian dual functions. Another common situation is the application of the Dantzig-Wolfe decomposition to a structured optimization problem in which formulations with an exponential number of variables are obtained. Generating these variables on demand by means of *column generation* is identical to performing a cutting plane on the respective dual problem.

The cutting plane algorithm will be used in Chapter 7.

Chapter 3

Preliminary

We first give our common definitions and important notations used in the thesis. Then we summarize the graph datasets we collected and used in our experiments.

3.1 Definitions

Definition 1. ***Connected Graph:*** *A graph is denoted by* $G = (\mathcal{V}, E, L, \mathcal{A})$*, where* $V = \{v_1, \cdots, v_{n_v}\}$ *is the vertices set,* $E \subseteq \mathcal{V} \times \mathcal{V}$ *is the edge set,* $\mathcal{A}$ *is a set of labels for vertices and edges, and* $L : \mathcal{V} \to \mathcal{A}, E \to \mathcal{A}$ *is a labeling function that assigns labels to a node or an edge. A connected graph is a graph such that there is a path between any pair of vertices.*

In our thesis, we focus on connected graphs and assume that each graph G has a class label y, $y \in \mathcal{Y} = \{-1, +1\}$, which may indicate the overall property of the graph, such as the active/negative response of a chemical compound [29] or the categorization of a publication [101]. In the imbalanced classification and cost-sensitive learning settings, $y_i = +1$ denotes the minority (positive) class, and $y_i = -1$ is the majority class (negative). We only focus on binary-class classification tasks (in Parts II and Part III of the thesis), but our methods can be easily extended to multi-class tasks. When considering correlated subgraph search (in Part I of this thesis), we might simply ignore the class labels of each graph,

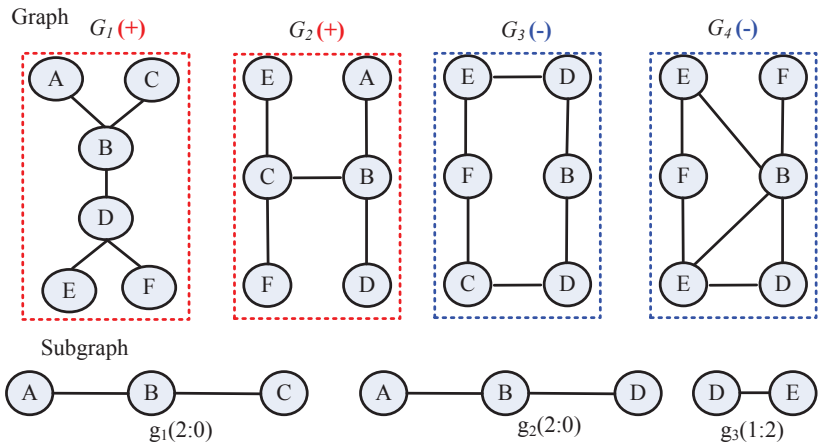


Figure 3.1: An example demonstrating subgraph representation for graphs: G_1 and G_2 are positive graphs (+), G_3 and G_4 are negative graphs (-). Each subgraph, g_1, g_2, or g_3, is marked with its frequency occurring in positive *v.s.* negative graphs.

as we focus on the substructure patterns from the graph database, regardless of the class labels.

Definition 2. ***Subgraph:*** *Given two graphs* $G = (\mathcal{V}, E, L, \mathcal{A})$ *and* $g_k = (\mathcal{V}', E', L', \mathcal{A}')$, g_k *is a subgraph of* G *(i.e.,* $g_k \subseteq G$*) if there is an injective function* $\hat{f}$*:* $\mathcal{V}' \to \mathcal{V}$*, such that* $\forall (a, b) \in E'$*, we have* $(\hat{f}(a), \hat{f}(b)) \in E$*,* $L'(a) = L(\hat{f}(a))$*,* $L'(b) = L(\hat{f}(b))$*,* $L'(a, b) = L(\hat{f}(a), \hat{f}(b))$*. If* g_k *is a subgraph of* G *(*$g_k \subseteq G$*),* G *is a supergraph of* g_k *(*$G \supseteq g_k$*).*

Definition 3. ***Subgraph Features****: Let* $\boldsymbol{g} = \{g_1, \cdots, g_m\}$ *denote a set of subgraph patterns discovered from a given graph set (In this thesis, subgraph patterns and subgraph features are equivalent terms). For each graph* G_i*, we can use a subgraph feature vector* $\boldsymbol{x}_i = [x_i^{g_1}, \cdots, x_i^{g_m}]$ *to represent* G_i *in the feature space, where* $x_i^{g_k} = 1$ *iff* g_k *is a subgraph of* G_i *(i.e.* $g_k \subseteq G_i$*) and* $x_i^{g_k} = 0$ *otherwise.*

In Fig. 3.1, three subgraph g_1, g_2, and g_3 are used to represent graph G_2 as $x_2 = [1, 1, 0]$.

Definition 4. ***Graph Stream****: A graph stream* $\mathcal{S} = \{\cdots, G_i, G_{i+1}, G_{i+2}, \cdots\}$ *contains an increasing number of graphs flowing in a streaming fashion. To process continuous stream data, we employ a "batch" concept which represents a graph chunk* $D_t = \{G_1^t, G_2^t \cdots, G_n^t\}$ *containing a number of graphs collected from a consecutive stream region. For ease of representation, we may drop* t *from each single graph* G_i^t *in graph chunk* D_t *when there is no ambiguity in the context.*

Definition 5. ***Noisy Graph****: Given a graph dataset* $\mathcal{T} = \{(G_1, y_1), \cdots, (G_n, y_n)\}$*, a noisy graph (or noise) is a graph whose label is incorrectly labeled (i.e., a positive graph is labeled as negative, or vice versa).*

Definition 6. ***Pearson's Correlation Coefficient:*** *Given two graphs* g_i *and* g_j*, their supports and joint support over a number of* N *graphs are denoted as* $supp(g_i)$*,* $supp(g_j)$*, and* $supp(g_i, g_j)$*, respectively. The Pearson's Correlation Coefficient [59] between* g_i *and* g_j*,* $\phi(g_i, g_j)$*, is defined as follows [150]:*

$$\phi(g_i, g_j) = \frac{supp(g_i, g_j) - supp(g_i)supp(g_j)}{\sqrt{supp(g_i)(1 - supp(g_i))supp(g_j)(1 - supp(g_j))}} \tag{3.1}$$

Subgraph-based Graph Classification: Given a set of labeled graphs $\mathcal{T} = \{(G_1, y_1), \cdots, (G_n, y_n)\}$, subgraph-based graph classification **aims** to select an optimal set of discriminative subgraphs from $\mathcal{T}$, and learn a classification model from the selected subgraph features to predict previously unseen test graphs with maximum accuracy.

Graph Stream Classification: Given a graph stream $\mathcal{S} = \{D_1, D_2, \cdots, D_t, \cdots\}$ collected in a number of consecutive graph chunks, the **aim** of the graph stream classification is to build a prediction model from the most recently observed k chunks $(D_{t-k+1}, \cdots, D_{t-1}, D_t)$ to predict graphs in the next chunk D_{t+1} with the best performance. In our setting, the graph data in each chunk may consist of a limited number of labeled graphs together with abundant unlabeled graphs, or

are highly imbalanced in class distributions and have noisy class labels.

3.2 Notations

The important notations used in the thesis are summarized in table 3.1. Note that in each chapter, we may also use some additional notations for each chapter.

3.3 Benchmark Graph Datasets

We have collected a set of graph datasets from various applications, including chemical compound classification, scientific publication classification, and sentiment analysis. The benchmarks we used are summarized in table 3.2, where column 2 details the tasks on which the graph collections are used in the thesis, and columns 3-6 depict the ID, the number of positive and total number of graphs in each dataset, and the dataset description in the collections.

NCI Anti-cancer activity prediction data. The NCI graph datasets are commonly used as the benchmark for graph classification. Each NCI dataset belongs to a bioassay task for anticancer activity prediction, where each chemical compound is represented as a graph, with atoms representing nodes and bonds as edges. A chemical compound is positive if it is active against the corresponding cancer, or negative otherwise. Table 3.2 summarizes the NCI graph data we download from PubChem [1]. We have removed disconnected graphs and graphs with unexpected atoms (some graphs have atoms represented as '*') in the original graphs. Columns 4-5 show the number of positive and total number of graphs in each dataset, respectively.

Full Dataset: The full datasets of NCI graphs are naturally imbalanced and ideal benchmark for streaming classification or cost-sensitive graph classification. More specifically, there are only about 5% of chemical compounds which are active to the cancer bioassay test, whereas 95% of them are inactive to the cancer

[1] http://pubchem.ncbi.nlm.nih.gov

Table 3.1: Important notations used in the chapter

Symbols	**Definition**
G, G_i	A connected graph
g, g_k	A subgraph
$\mathcal{S} = \{\cdots, G_i, G_{i+1}, \cdots\}$	A graph stream
$\mathcal{S} = \{D_1, D_2, \cdots, D_t, \cdots\}$	Chunk representation of graph stream
$D_t = \{G_1^t, G_2^t \cdots, G_n^t\}$	The t-th graph chunk, t can be dropped off
$D_t = D_t^l \bigcup D_t^u$	A graph chunk with D_t^l and D_t^u denoting labeled and unlabeled graphs
$\phi(g_i, g_j)$	Pearson's Correlation Coefficient
min_sup	Minimum support for frequent subgraph mining
$\mathbf{g} = \{g_1, \cdots, g_m\}$	A set of selected subgraphs for semi-supervised classification
$\mathcal{T} = \{G_i, y_i\}_{i=1,\cdots,n}$	A set of training graphs
l_+, l_-	Number of positive and negative graphs
$\boldsymbol{x}_i$	Vector representation of graph G_i for classification
$\hbar(G_i; g_k, \pi_k)$ or $\hbar_{g_k}(G_i; \pi_k)$ or $\hbar_{g_k}(G_i)$	A subgraph decision stump
$\mathcal{F} = \{g_1, \cdots, g_m\}$	The full set of subgraphs
S	Selected discriminative subgraphs
$\boldsymbol{w} = \{w_k\}_{k=1,\cdots,m}$	Weight vectors for all subgraphs
$f(G_i), f(\boldsymbol{x}_i)$	Classifier prediction on graph G_i
C_1, C_{-1}	Cost of positive and negative graphs, respectively
$\boldsymbol{\xi} = \{\xi_i\}_{i=1,\cdots,l}$	Vector, slack variables for objective functions
ξ	Slack variable (scalar) for cutting plane algorithm
$\boldsymbol{\mu} = \{\mu_i\}_{i=1,\cdots,l}$	Weight vectors of training graphs
T_{max}	Maximum number of iterations

Table 3.2: Description of Graph Datasets Used in the Thesis

Collections	Tasks	ID	#Pos	#Total	Description
NCI	Stream/ Cost-sensitive Graph Classification (Chapters 5, 6, 7)	1	1793	37349	Cell Lung
		33	1467	37022	Melanoma
		41	1350	25336	Prostate
		47	1735	37298	Central Nerv Sys
		81	2081	37549	Colon
		83	1959	25550	Breast
		109	1773	37518	Ovarian
		123	2715	36903	Leukemia
		145	1641	37043	Renal
NCI-*balanced*	Multi-task Graph Classification (Chapter 8)	1	1793	3586	Cell Lung
		33	1467	2934	Melanoma
		41	1350	2700	Prostate
		47	1735	3470	Central Nerv Sys
		81	2081	4162	Colon
		83	1959	3918	Breast
		109	1773	3546	Ovarian
		123	2715	5430	Leukemia
		145	1641	3282	Renal
DBLP -*balanced*	Semi-supervised Graph Stream Classification (Chapter 5)	DBLP	9530	19456	DBDM *v.s* CVPR
DBLP -*imbalanced*	Imbalanced Graph Stream Classification (Chapter 6)	DBLP	3954	24225	CV *v.s* DBDM & AIML
Twitter	Imbalanced Graph Stream/ Cost-sensitive Classification (Chapters 6, 7)	Twitter	66458	140949	Sentiment Analysis
PTC	Multi-task Classification (Chapter 8)	Sub_{MR}	32	87	Male Rat (MR)
		Sub_{FR}	35	85	Female Rat (FR)
		Sub_{MM}	29	85	Male Mouse (MM)
		Sub_{FM}	35	88	Female Mouse (FM)

bioassay test. We have considered graph stream classification in [101] (detailed in Chapter 5) and [104] (Chapter 6), and cost-sensitive graph classification in [102] (Chapter 7)

Partial Dataset (NCI-balanced): We randomly select #Pos number of negative graphs from each original graph set to create balanced graph datasets. Although each of the 9 tasks focuses on the prediction of different types of cancers, all these tasks are relevant in cancer prediction and some common discriminative substructures may exist for all types of cancers. This makes NCI an ideal benchmark for multi-task graph classification, which will be detailed in Chapter 8.

DBLP Graph Stream. The DBLP data stream [1] consists of bibliography data in computer science. Each record in DBLP is associated with a number of attributes such as abstract, authors, year, venue, title, and reference ID [127]. We have built two graph data streams, DBLP-balanced and DBLP-imbalanced, from a set of conferences. Detailed information is given as follows:

Table 3.3: DBLP-balanced used in this thesis

Categories	Descriptions	#Paper	#Graphs
DBDM	SIGMOD,VLDB,ICDE, EDBT,PODS, DASFAA,SSDBM,CIKM,DEXA KDD, ICDM, SDM, PKDD, PAKDD	20601	9530
CVPR	ICCV, CVPR, ECCV, ICPR, ICIP ACM Multimedia, ICME	18366	9926

DBLP-balanced: We select a list of conferences (as shown in Table 3.3) and use the papers published in these conferences (in chronological order) to form a balanced graph stream. The classification task is to predict whether a paper belongs to DBDM (database and data mining) or CVPR (computer vision and pattern recognition) field, by using the references and the title of each paper. Notice that DBDM and CVPR are overlapping in many aspects, such as machine learning and visual information retrieval. The shifting of the research focus makes DBLP stream an ideal test ground for concept drifting graph stream classification. For example, there are an increasing number of papers to address social

[1] http://arnetminer.org/citation

Table 3.4: DBLP-imbalanced graph stream used in experiments

Categories	Descriptions	#Paper	#Graphs
DBDM	SIGMOD, VLDB, ICDE, EDBT, PODS, ICDT, DASFAA, SSDBM, CIKM KDD, ICDM, SDM, PKDD, PAKDD	18870	10089
AIML	IJCAI, AAAI, NIPS, UAI, COLT, ACL, KR, ECAI, ICML, ECML, ACML, IJCNN	24090	10182
CV	CVPR, ICCV, ECCV, ACCV, ACM Multimedia	7032	3954

network research problems for both DBDM and CVPR fields (*i.e.*, community discovery for DBDM and social tagging in CVPR), which naturally introduces concept drifting in the stream. The DBLP-balanced graph stream is used for semi-supervised graph stream classification in Chapter 5.

DBLP-imbalanced: We also build an imbalanced graph stream from DBLP datasets. The conference list used for DBLP-imbalanced stream is given in Table 3.4). We form a minority class by using publications in computer vision (CV) as positive class (+1), and use papers in both DBDM (database and data mining) and AIML (artificial intelligence and machine learning) as negative class (-1). The graph stream is inherently imbalanced, with about 16.3% positive graphs over stream. The DBLP-imbalanced stream is used for imbalanced graph task classification in Chapter 6.

In our thesis, each paper in DBLP is represented as a graph, where each node denotes a Paper ID or a keyword and each edge denotes the citation relationship between papers or keyword relations in the title. More specifically, we denote that (1) each paper ID is a node; (2) if a paper P.A cites another paper P.B, there is an edge between P.A and P.B; (3) each keyword in the title is also a node; (4) each paper ID node is connected to the keyword nodes of the paper; and (5) for each paper, its keyword nodes are fully connected with each other. An example of DBLP graph data is shown in Fig. 1.3.

The original DBLP dataset contains a significant number of papers without references. In our experiments, we remove those papers, and choose 1000 most

frequent words appearing in the title (after removing the stop words) as keywords to construct graphs. The last column in Table 3.3 shows the number of graphs in each category in our experiments. The DBLP dataset is used in [101] (detailed in Chapter 5) and [104] (Chapter 6).

Stanford Twitter Graphs [1] are extracted from twitter sentiment classification [47]. Because of the inherently short and sparse nature, twitter sentiment analysis (*i.e.*, predicting whether a tweet reflects a positive or a negative feeling) is a difficult task. To build a graph dataset, we represent each tweet as a graph by using tweet content, with nodes in each graph denoting the terms and/or smiley symbols (*e.g*, :-D and :-P) and edges indicating the co-occurrence relationship between two words or symbols in each tweet. To ensure the quality of the graph, we only use tweets containing 20 or more words.

Twitter Stream: The twitter graphs are used for graph stream classification in [104] (detailed in Chapter 6). Specifically, we use tweets from April 6 to June 16 to generate 140,949 graphs (in a chronological order). Because tweets in the original dataset are not evenly collected over time, the number of graphs in a fixed time period varies significantly (from 100 to 10,000 per day). To reduce the difference of chunk size over stream, we divide graphs into chunks by using a fixed time period, *i.e.*, graphs are collected in 24 hours (one day) to form a graph chunk from April 6 to May 27, and collected in 8 hours to form a chunk from May 27 and latter on. To investigate algorithm performance in handling concept drifts, we synthetically control the prior distribution of positive graphs at several fixed time stamps. Specifically, 20% of positive graphs are randomly selected on Monday and Tuesday over time before June 2. By doing so, we use sudden changes of priori distributions to inject concept drifting on Monday.

Static Twitter Graph Classification: To study how our algorithms scale to large scale graph datasets, we also aggregate all twitter graphs as one dataset without considering their temporal order. This dataset is used in [102] (detailed in Chapter 7).

Predictive Toxicology Challenge Dataset (PTC). The PTC challenge in-

[1] http://jmgomezhidalgo.blogspot.com.au/2013/01/a-list-of-datasets-for-opinion-mining.html

cludes a number of carcinogenicity tasks for toxicology prediction of chemical compounds [1]. The dataset we selected contains 417 compounds with four types of test animals: MM (male mouse), FM (female mouse), MR (male rat), and FR (female rat). Each compound with one is label selected from {CE, SE, P, E, EE, IS, NE, N}, which stands for Clear Evidence of Carcinogenic Activity (CE), Some Evidence of Carcinogenic Activity (SE), Positive (P), Equivocal (E), Equivocal Evidence of Carcinogenic Activity (EE), Inadequate Study of Carcinogenic Activity (IS), No Evidence of Carcinogenic Activity (NE), and Negative (N). Similar to [77], we set {CE, SE, P} as positive labels, and {NE, N} as negative labels. In order to formulate MTG dataset, we randomly split 417 compounds into 4 equal and disjointed subsets. For each subset, we only consider one type of carcinogenicity test as its learning task. The subset information is also listed in Table 3.2. As a result the PTC collection is suitable for multi-task classification (Chapter 8).

[1] http://www.predictive-toxicology.org/ptc/

Part I

Graph Stream Query

Graph Stream Search: Overview

Correlations mining has drawn increasing interest in past years due to its great advantages in uncovering underlying dependencies between objects. For graph data, the correlation between two graphs measures their occurrence distributions, which is important for discovering the interesting patterns in the graph database.

However, existing correlation mining for graph data is only designed for static graph databases. In reality, applications usually involve data which constantly change or evolve over time, (*i.e.,* data streams). How to discover the correlated patterns (subgraphs) for the dynamic graph streams is important, yet has not been explored in existing works.

In Part I (*i.e.,* Chapter 4), we propose to query correlated graph in a data stream scenario, where given a query graph q, an algorithm is required to retrieve all the subgraphs whose Pearson's correlation coefficients with q are no less than a threshold θ over some graph data flowing in a stream fashion. Due to the dynamic changing nature of the stream data and the inherent complexity of the graph query process, treating graph streams as static datasets is computationally infeasible or ineffective. We propose a novel algorithm, CGStream, to identify correlated graphs from data stream, by using a sliding window which covers a number of consecutive batches of stream data records. Our theme is to regard stream query as the traversing along a data stream and there are some special time points called outlooks over streams. For each outlook, we derive a frequency lower bound to mine a set of frequent subgraph candidates, where the lower bound guarantees that no pattern is missing from the current outlook to the next outlook. On top of that, we derive an upper correlation bound and a heuristic rule to prune the candidates, which helps reduce the computation cost at each outlook. Experimental results demonstrate that the proposed algorithm is several times, or even up to an order of magnitude, more efficient than the straightforward algorithm. Meanwhile, our algorithm achieves good performance in terms of query precision.

Chapter 4

Continuous Correlated Graph Query for Data Streams

4.1 Introduction

Correlation mining has drawn extensive attention in the research community due to its uniqueness and advantages in uncovering underlying dependencies between objects. In recent years, there have been a considerable number of studies on correlation mining in applications including market transaction databases [150, 165], quantitative databases [72], and time series data [96], and this topic has been recently extended to graph databases, where data records or instances are linked through relationships [71, 74].

In the context of graph data, the correlation between two graphs measures their occurrence distributions. Given a graph database, correlated graph search (CGS) [71] tries to discover a set of correlated graphs whose Pearson correlation coefficients [59] with a query graph are above a given correlation threshold θ.

CGS is very useful for revealing interesting patterns in many graph representation scenarios. For instance, a user's browsing history in a web site can be represented as graphs. Correlated graphs retrieved from the user transversal graphs represent graphs with similar distributions and suggest a group of users

sharing common interests. Analyzing these correlated graphs helps web site owners understand user behaviors so that they can improve the web site structures and detect abnormal behaviors, which are very important in E-commerce [74].

Current CGS [71] is only performed in a static database. In practice, applications may involve data which constantly change or evolve over time (*e.g.*, data streams). For instance, in a communication network, the links between different nodes are changing continuously, so the network topology (which can be regarded as a graph) will dynamically change. In a chemical reaction process, the structures of chemical compounds also change from time to time, so the interactions between compounds (which can be regarded as graphs) also change dynamically. Noticing the importance of the pattern discovery from graph streams, there have been several researches on continuously querying graphs in a data stream setting [138], or mining frequent graphs in a data stream scenario [15]. However, to the best of our knowledge, there is no existing research on correlated graph pattern query in data streams.

In this chapter, we study correlated graph query for data streams, where the main challenges are as follows

- Graph correlation query involves subgraph isomorphism testing which is NP-complete. It is intractable to store and compute the frequency and correlation for every subgraph over stream.
- The correlation between graphs is constantly changing over stream and recomputing all the correlations at each single time point of data stream is computationally expensive and time-consuming.
- The streaming scenario requires the algorithm to return answers in a timely fashion.

Intuitively, a straightforward approach to solve our problem is exhaustive search, which uses a sliding window to scan the stream and computes the correlation by employing a static graph database based algorithm, such as CGSearch [71] [1], to query the correlated graphs in each window. While this exhaustive approach can ensure the results being complete and correct, it is computationally

[1]The algorithm for CGS problem is named CGSearch in [71].

ineffective because the reoccurring query process in each window involves frequent subgraph mining procedure and subgraph frequency counting procedure, both of which are expensive operations, especially when the window size is considerable large. An incremental method for stream-based correlated graph query is highly demanded.

In this chapter, we propose a solution to incrementally retrieve correlated graphs over stream. Our theme is to regard each query g_q as an operator, which constantly queries subgraph patterns correlated to itself while traversing along the graph stream. To answer the query in an efficient way, we propose to maintain a number of outlooks ($\mathcal{O}$) over the data stream. For each query g_q, each outlook ($\mathcal{O}_i$) carefully maintains a candidate list with two properties: (1) all subgraph patterns correlated to the query g_q, with respect to the current sliding window, are included in this list; and (2) before reaching the next outlook ($\mathcal{O}_{i+1}$), no subgraph pattern correlated to the query g_q is missing. Accordingly, the correlated graph query can be achieved by querying the lists maintained at each outlook. Because the mining procedure is only triggered at each outlook, we can significantly reduce the computational cost by using effective methods to build and maintain the candidate list.

The candidate list at each outlook is vitally important to determine the system efficiency. If the list is infinitely long and includes all possible subgraphs, a query will not miss any patterns but scanning such an infinite list will be inefficient. Meanwhile, because each outlook requires a significant amount of computational cost to build and maintain its candidate list, a query system is going to be very inefficient if there are a large number of outlooks in the stream. To ensure the system runtime performance, we need to maintain as few stream outlooks as possible, yet the candidate list in each outlook should also be sufficient to ensure the query quality. In the chapter, we derive a set of theoretical bounds to maintain the length of the candidate list, and also propose some forecasting procedures to minimize the number of outlooks through which we can accelerate the system runtime performance.

The main contributions of this chapter can be summarized as follows:

- We propose a CGStream algorithm for correlated graph query for streams, where the algorithm will return a query graph's correlated graphs in a

sliding window which covers a number of consecutive batches of stream records. To the best of our knowledge, this is the first endeavor to mine threshold based correlated graphs in a data stream scenario.

- We derive a lower frequency bound to mine and maintain a candidate list for each stream outlook ($\mathcal{O}_i$). Our solution guarantees that any subgraph patterns not included in the candidate list are not correlated to the query g_q before reaching the next outlook ($\mathcal{O}_{i+1}$) (so we can safely reduce these patterns from the list).
- We derive a loose upper correlation bound to prune the candidate list for each outlook ($\mathcal{O}_i$), and use a heuristic rule to speed up the mining process, which helps reduce the computational cost.
- Experiments confirm that our algorithm is several times, or even an order of magnitude, more efficient than an exhaustive search method.

The rest of this chapter is structured as follows: The preliminaries and formulation of our research problem are given in Section 4.2. We derive a new frequency(support) lower bound for candidate generation in Section 4.3. In Section 4.4, we further derive a correlation upper bound and a heuristic rule for candidate pruning. The algorithm is presented in Section 4.5, and experimental results are presented in Section 4.6. We conclude the chapter in Section 4.7.

4.2 Preliminaries and Problem Definition

4.2.1 Preliminaries

Given a graph database $\mathcal{T}$, the projected dataset with respect to a graph g_i is a subset of $\mathcal{T}$ which contains g_i, denoted as $\mathcal{T}_{g_i} = \bigcup\{G|g_i \subseteq G, G \in \mathcal{T}\}$, whose frequency $N_{g_i} = |\mathcal{T}_g|$ and support $supp(g_i) = |\mathcal{T}_{g_i}|/|\mathcal{T}|$, where $|\bullet|$ denotes the cardinality of the set $\bullet$.

For two graphs g_i and g_q in $\mathcal{T}$, their joint frequencies are the number of graphs in $\mathcal{T}$ which contains both graph g_i and g_q, denoted as $N_{g_ig_q} = |\mathcal{T}_{g_q} \bigcap \mathcal{T}_{g_i}|$, and their joint support is $supp(g_i, q_q) = |\mathcal{T}_{g_q} \bigcap \mathcal{T}_{g_i}|/|\mathcal{T}|$.

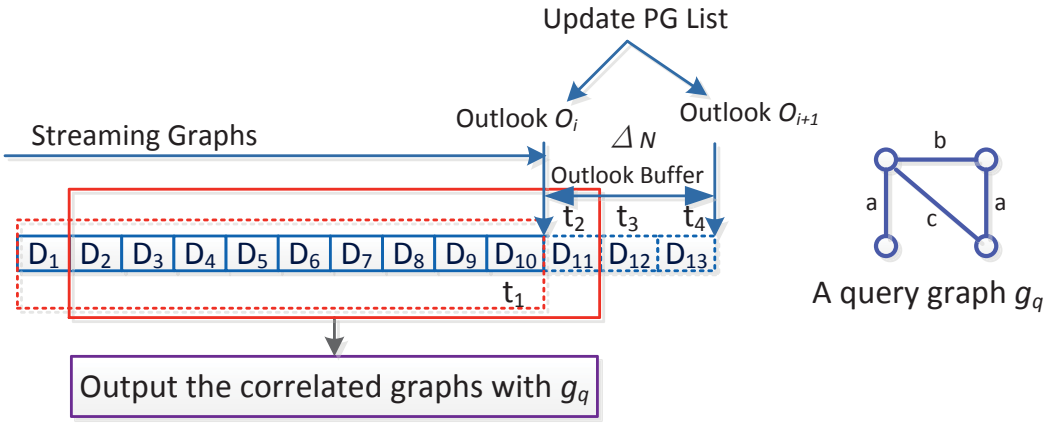


Figure 4.1: A framework of sliding window based correlated graph query for data streams. At time point t_1, the sliding window (dashed red rectangle) covers batches $D_1, D_2, \ldots, D_{10}$. A new batch D_{11} arrives at time point t_2 (D_{11} becomes the most recent batch), the sliding window updates to cover $D_2, D_3, \ldots, D_{11}$ (solid red rectangle). Continuous correlated graph query intends to discover the correlated graphs in every sliding window. Outlooks are specific time points where we build and update the potential candidate list PG (such as $\mathcal{O}_i$ and $\mathcal{O}_{i+1}$). At any other time points between two outlooks (like time points t_2 and t_3), we only update the frequency of candidates and output the correlated graphs, without carrying out any pattern mining process.

4.2.2 Problem definition

Given a query graph g_q, a threshold θ, and a graph stream $\mathcal{S}$, we emphasize on discover correlated graphs whose correlations with g_q are above θ from $\mathcal{S}$. Because $\mathcal{S}$ represents a dynamic changing graph stream, we assume that graph data arrives batch by batch, and use a sliding window $D_{win} = \{D_{j-w+1}, D_{j-w+2}, \cdots, D_j\}$ to denote a consecutive region of the graph stream, where $D_\iota, j - w + 1 \leq \iota \leq j$ represents a batch of graphs and D_j is the most recent batch. Then our problem is to monitor and report the correlated graphs whose Pearson's correlations [59] with g_q in a sliding window are greater than θ (in the most recent w batches). A typical correlated graph query in a data stream with window size w=10 is shown in Figure 4.1.

Pearson's Correlation Coefficient (Definition 6): Given two graphs g_i and g_q, their supports and joint support over a number of N graphs are denoted as $supp(g_i)$, $supp(g_q)$, and $supp(g_i, g_q)$, respectively. The Pearson's Correlation

Coefficient [59] between g_i and g_q, $\phi(g_i, g_q)$, is defined as follows [150]:

$$\phi(g_i, g_q) = \frac{supp(g_i, g_q) - supp(g_i)supp(g_q)}{\sqrt{supp(g_i)(1 - supp(g_i))supp(g_q)(1 - supp(g_q))}} \tag{4.1}$$

When $supp(g_i)$ or $supp(g_q)$ is equal to 0 or 1, $\phi(g_i, g_q))$ is defined to be 0. The range of $\phi(g_i, g_q)$ falls into $[0, 1]$, as we only consider the positive correlated graphs in this thesis.

The Pearson's correlation coefficient over a set containing a number of N graphs can be rearranged into another form in terms of frequency [165]:

$$\phi(g_i, g_q) = \frac{NN_{g_i g_q} - N_{g_i}N_{g_q}}{\sqrt{N_{g_i}(N - N_{g_i})N_{g_q}(N - N_{g_q})}} \tag{4.2}$$

N_{g_i}, N_{g_q}, and $N_{g_i g_q}$ denote the number of graphs containing g_i, g_q, and g_i and g_q for the N graphs, respectively.

4.2.3 Challenges and Solutions

A main challenge of correlated graph search (CGS) for data stream is that the correlation is constantly changing over time and recomputing the correlation for each candidate is time-consuming. This is because computing $\phi(g_i, g_q)$ involves graph isomorphism testing when counting the frequency of g_i (*i.e.* N_{g_i}), which is NP-Complete. In addition, because a graph consists of an exponential number of subgraphs and each of which is a potential correlated graph candidate to the query g_q, the search space of CGS, for data stream, is extremely large.

In this chapter, we propose a CGStream algorithm to address the problem and ensure that each correlated graph query can be answered in an efficient and accurate way. Our solution is inspired by a *checkpoint* idea in [165], and our theme is to create a number of outlooks ($\mathcal{O}$) over stream. The framework of CGStream is illustrated in Fig. 4.1.

In summary, an outlook ($\mathcal{O}_i$) is a specific time point which can help to derive some theoretical correlation bounds, so that a CGS algorithm only needs

to maintain a small set of potential candidates without referring to the original graph stream data to answer the query. The rationale is as follows: given a graph stream and a sliding window which covers a number of graphs, assume at a outlook $\mathcal{O}_i$ the sliding window covers N graphs in the stream. Suppose that ΔN new graphs ($\Delta N \ll N$) will arrive at next outlook $\mathcal{O}_{i+1}$, and ΔN graphs are removed from the monitor window D_{win}, we can build and maintain a candidate list PG at time point $\mathcal{O}_i$ such that any patterns not belonging to the PG are not going to satisfy the query before reaching time point $\mathcal{O}_{i+1}$. As a result, only the candidates remaining in the PG list are promising for future investigation. In this case, the increment ΔN between two outlooks can be regarded as a computation buffer (outlook buffer). For graph stream between two adjacent outlooks (e.g., t_2 and t_3 in Fig. 4.1), we just need to check the PG list to output the correlated graphs with query graph g_q, without reoccurring the query process in the whole window of data.

While maintaining the PG list, two issues should be considered:

1. To achieve high retrieval recall, the PG list for each outlook should be as complete as possible. In an ideal case, a PG list should include all possible candidates, so the CGS search for data stream can precisely return all graphs correlated to the query g_q.

2. To meet the requirement of high speed of data stream, query to PG list should be as efficient as possible. In an ideal case, a PG list should only contain graph patterns correlated to the query g_q, so the query can be answered with minimum cost.

While the above two issues are contradictory to each other, we address each of them by employing the stream outlooks as follows.

On the one hand, for each outlook ($\mathcal{O}_i$), we derive a lower frequency bound $lower(N_{g_i g_q})$, and transfer the CGS problem to a frequent subgraph mining problem, which can be addressed by some existing graph mining algorithms such as gSpan algorithm [153]. More specifically, instead of mining from the original window of graph data, we mine a set of correlated graph as potential graphs (PG) from the projected database of a query graph g_q. The PG list has taken the increment (ΔN) between two adjacent outlooks ($\mathcal{O}_i$ and $\mathcal{O}_{i+1}$) into consideration

and guarantees that all potential candidates are stored as long as the number of increment graphs (between $\mathcal{O}_i$ and $\mathcal{O}_{i+1}$) containing the query graph g_q, *i.e.* ΔN_{g_q}, is within a certain range $\alpha \leq \Delta N_{g_q} \leq \beta$. By doing so, we can ensure that the PG list maintained at each outlook $\mathcal{O}_i$ is as complete as possible.

On the other hand, when the lower bound $lower(N_{g_i g_q})$ is relatively small, the size of the PG list may be too large, which will significantly slow down the query process. In the chapter, we further derive an upper correlation bound to reduce the size of the PG list. On top of that, some heuristic rules are also applied to the candidate checking procedure to speed up the mining process. As a result, we can guarantees the efficiency of our algorithm.

Our algorithm is based on the assumption that we can estimate the mean and variance of N_{g_q} in each batch from the data stream. Based on this assumption, we can derive the frequency lower bound to generate the PG list (Sect. 4.3), and derive the correlation upper bound to further reduce the PG list (Sect. 4.4).

4.3 Frequency lower bound for candidate generation

Motivated by the existing CGSearch algorithm [71], our CGStream algorithm transfers the CGS for data stream problem to a frequent subgraph mining problem. In other words, instead of mining from the original window of graph data, we mine a set of frequent graphs as potential graphs (PG) from the projected database $\mathcal{T}_q$ of the query graph g_q. As $\mathcal{T}_q$ is a much smaller subset of $\mathcal{T}$, the search space and time consumption can be greatly reduced. Accordingly, the first technical challenge of our algorithm is:

How to derive a lower bound for frequent subgraph mining from $\mathcal{T}_q$, so that those frequent subgraphs from $\mathcal{T}_q$ are not only potential correlated graphs in the current outlook ($\mathcal{O}_i$ in Fig. 4.1), but are also candidates before reaching the next outlook ($\mathcal{O}_{i+1}$ in Fig. 4.1)?

If we can derive such a lower bound, the re-computation process can be greatly reduced. For instance, at time stamps t_2 and t_3 in Fig. 4.1, we only need to update

the frequency of the candidates and then output the results quickly without querying from the entire sliding window.

4.3.1 Frequency lower bound

Lemma 1. *Given a query graph g_q, the maximum correlation of graph g_i with g_q is achieved when $supp(g_i) = supp(g_i, g_q)$, or $N_{g_i} = N_{g_i g_q}$*

$$\begin{aligned}\phi_{max}(g, q) &= \sqrt{\frac{(1 - supp(g_q))supp(g_i, g_q)}{supp(g_q)(1 - supp(g_i, g_q))}} \\ &= \sqrt{\frac{(N - N_{g_q})N_{g_i g_q}}{N_{g_q}(N - N_{g_i g_q})}}\end{aligned} \tag{4.3}$$

Proof. From Eq. (4.1), we know that when we fix $supp(g_i, g_q)$ and $supp(g_q)$, $\phi(g_i, g_q)$ monotonically decreases with $supp(g_i)$. As $supp(g_i, g_q) \leq supp(g_i) \leq 1$, $\phi(g_i, g_q)$ will achieve its maximum value when $supp(g_i) = supp(g_i, g_q)$. Similarly, we can prove that when $N_{g_i} = N_{g_i g_q}$, $\phi(g_i, g_q)$ achieves its maximum value according to Eq. (4.2). □

In Eq. (4.3), N is a constant, and N_{g_q} can be easily computed after obtaining the projected database $\mathcal{T}_q$. Then for each subgraph g_i, we need to count $N_{g_i g_q}$ to check if it is a potential candidate (g_i is a potential candidate if $\phi_{max}(g, q) \geq \theta$). Here we derive a new lower bound of $N_{g_i g_q}$, by using outlooks as a number of stream processing nodes.

Suppose with the arriving of data stream, there are ΔN new graphs (outlook buffer) flowing into the sliding window. Meanwhile, ΔN graphs will become outdated and be removed from the sliding window (We consider the case when the sliding window is full, so the total number of graphs in the sliding window remain the same). Similarly, the number of N_{g_q} will increase with $\overline{\Delta N_{g_q}}$, and $\underline{\Delta N_{g_q}}$ will be removed from the sliding window.

Let $\gamma = \Delta N_{g_q} = \overline{\Delta N_{g_q}} - \underline{\Delta N_{g_q}}$ be the number of increment of N_{g_q} between

two outlooks. γ can be either ≥ 0 (if $\overline{\Delta N_{g_q}} \geq \underline{\Delta N_{g_q}}$) or ≤ 0. Then Eq. (4.3) can be rearranged as follows:

$$\phi'_{max}(g,q) = \sqrt{\frac{(N - N_{g_q} - \gamma)(N_{g_i g_q} + \Delta N_{g_i g_q})}{(N_{g_q} + \gamma)(N - N_{g_i g_q} - \Delta N_{g_i g_q})}} \tag{4.4}$$

We know that $\phi'_{max}(g,q)$ monotonically increases with $N_{g_i g_q} + \Delta N_{g_i g_q}$ according to Eq. (4.4), and $\Delta N_{g_i g_q} \leq |\gamma|$, the maximum correlation will be

$$\phi'_{max}(g,q) = \begin{cases} \sqrt{\frac{(N-N_{g_q}-\gamma)(N_{g_i g_q}+\gamma)}{(N_{g_q}+\gamma)(N-N_{g_i g_q}-\gamma)}} & : \quad \gamma \geq 0 \\ \sqrt{\frac{(N-N_{g_q}-\gamma)N_{g_i g_q}}{(N_{g_q}+\gamma)(N-N_{g_i g_q})}} & : \quad \gamma \leq 0 \end{cases} \tag{4.5}$$

Because we can estimate the mean and variance of N_{g_q} in each batch from the data stream, γ can be estimated from the historical stream data. In Sec. 4.3.2, we will propose techniques to estimate the range of γ with $\gamma \in [\alpha, \beta]$, $\alpha \leq 0, \beta \geq 0$.

Assume that we need to find out the correlated graph with correlation above θ (with γ increment of N_{g_q}) within the outlook buffer, *i.e.*, we have a requirement

$$\phi'_{max}(g,q) \geq \theta \tag{4.6}$$

From Eq. (4.5) and Eq. (4.6), we know when $\gamma \leq 0$, we have

$$\sqrt{\frac{(N - N_{g_q} - \gamma)N_{g_i g_q}}{(N_{g_q} + \gamma)(N - N_{g_i g_q})}} \geq \theta$$

Then we have minimum $N_{g_i g_q}$ for $\gamma \leq 0$

$$N_{g_i g_q} \geq \frac{\theta^2(N_{g_q} + \gamma)(N - \gamma)}{\theta^2(N_{g_q} + \gamma) + (N - N_{g_q} - \gamma)} = f_1(\gamma) \tag{4.7}$$

Taking the partial derivative of γ, we have

$$f_1'(\gamma) = \frac{\theta^2(N - N_{g_q})}{(N_{g_q} + \gamma)\theta^2 + (N - N_{g_q} - \gamma)} - \frac{\theta^2(\theta^2 - 1)(N_{g_q} + \gamma)(N - \gamma)}{(N - N_{g_q} - \gamma + (N_{g_q} + \gamma)^2)^2}$$

As $\gamma \leq N_{g_q} \leq N$ and $\theta \leq 1$, the first term of $f_1'(\gamma) \geq 0$, and the second term ≤ 0. As a result, $f_1'(\gamma) \geq 0$. $f_1(\gamma)$ is a monotonic increasing function, so the minimum of $f_1(\gamma)$ is $f_1(\alpha)$ $(\alpha \leq \gamma \leq 0)$.

Similarly, we can compute the minimum $N_{g_i g_q}$ when $\gamma \geq 0$

$$N_{g_i g_q} \geq \frac{\theta^2(N_{g_q} + \gamma)(N - \gamma) - (N - N_{g_q} - \gamma)\gamma}{\theta^2(N_{g_q} + \gamma) + (N - N_{g_q} - \gamma)} = f_2(\gamma) \tag{4.8}$$

It can be proved that $f_2(\gamma)$ is a monotonic decreasing function with γ, $0 \leq \gamma \leq \beta$, so the minimum of $f_2(\gamma)$ is $f_2(\beta)$;

Theorem 1. *Given N, N_{g_q}, γ, $\alpha \leq \gamma \leq \beta$, if a graph is a potential candidate before reaching the next outlook, its minimum frequency in the projected database $\mathcal{T}_q$ at current outlook is at least*

$$lower(N_{g_i g_q}) = \min(f_1(\alpha), f_2(\beta)) \tag{4.9}$$

where $f_1(\gamma)$ and $f_2(\gamma)$ are defined in Eq. (4.7) and Eq.(4.8).

We can use $lower(N_{g_i g_q})$ as a frequency bound to mine a set of frequent subgraphs from the projected database, these frequent subgraphs are potential candidates after taking next outlook into consideration. In other words, when ΔN graphs (outlook buffer) flow into the sliding window, as long as the increment of N_{g_q}, say $\alpha \leq \gamma \leq \beta$, all the true correlated graphs will be kept as candidates.

4.3.2 Estimation the increment of γ

From Eq. (4.9) it is clear that to get $lower(N_{g_i g_q})$, we must know α and β, *i.e.* the range of $\gamma = \Delta N_{g_q} \in [\alpha, \beta]$. In fact, because N_{g_q} is always larger than 0, it is easy to find that γ will fall into the range $[-\Delta N_{g_q}, \Delta N]$. However, a loose and large β ($\beta = \Delta N$ in this case) will result in a relatively small $lower(N_{g_i g_q})$ value, which in turn increases the system runtime for frequent subgraph mining. So a tight range of γ (small α or β) is preferred. Because we can easily collect N_{g_q} in each batch over stream, we maintain a list of frequency record of N_{g_q}, and use Poisson distribution/Skellam distribution [123] to estimate the range of γ. Here Poisson distribution is used, as it is best to express the probability of a given number of events occurring in a fixed interval of time.

Assume that the frequency of occurrence for the query graph g_q in a fixed time period (in a batch) follows a Poisson distribution. The incoming graphs containing g_q follows $\overline{\Delta N_{g_q}} = P(x = k; \lambda_1) = \frac{\lambda_1^k e^{-\lambda_1}}{k!}$ distribution, and the outdated graphs (which will be discarded as we only focus on the most w batches of data) containing g_q follows $\underline{\Delta N_{g_q}} = P(x = k; \lambda_2) = \frac{\lambda_2^k e^{-\lambda_2}}{k!}$ distribution. Then the difference between these two Poisson distribution follows a Skellam distribution [123].

$$f(k; \lambda_1, \lambda_2) = e^{-(\lambda_1+\lambda_2)}(\frac{\lambda_1}{\lambda_2})^{k/2} I_{|k|}(2\sqrt{\lambda_1+\lambda_2}) \tag{4.10}$$

where $I_k(z)$ is the modified Bessel function of the first kind [20].

In our setting, we set the average of N_{g_q} in the most recent $w/2$ batches (half size of the sliding window) as λ_1, and the average of N_{g_q} in the oldest $w/2$ batches (half window) as λ_2.

The Skellam distribution $f(k; \lambda_1, \lambda_2)$ has mean $\mu = \lambda_1 - \lambda_2$, variance $\sigma^2 = \lambda_1 + \lambda_2$. Then we set $\alpha = \mu - 3\sqrt{\sigma}$, and $\beta = \mu + 3\sqrt{\sigma}$. As a result, it is reasonable to set α and β in the range within the three-standard-deviation range of the mean, as the probability that a point falls into this range is over 0.99 for a Skellam distribution.

4.4 Correlation upper bound and Heuristic rules for candidate pruning

After obtaining the estimated value of γ, we can mine the frequent subgraphs from the projected data using $lower(N_{g_i g_q})$ as a lower bound of threshold (Eq. (4.9)), and then add these frequent graphs into the candidate set. In the incoming batches before reaching next outlook, we can quickly output the correlated graphs without involving the candidate building procedure from the scratch, which will greatly reduce the system computation cost.

In reality, when $lower(N_{g_i g_q})$ value is relatively small, it may end up with a large candidate set, which requires a significant amount of time to query for each candidate graph g_i. In this subsection, we further reduce the candidate set by deriving a new correlation bound. The idea is as follows: For any outlook $\mathcal{O}_i$, based on current statistics of N_{g_i}, N_{g_q}, $N_{g_i g_q}$, we can compute and predict the upper bound correlation value between each graph g_i and the query g_q, $\phi^{\diamond}_{max}(g_i, g_q)$, with respect to the ΔN increment graphs. If $\phi^{\diamond}_{max}(g_i, g_q)$ is lower than the given correlation threshold, we can remove it safely.

Suppose when ΔN new graphs arrive, there are also ΔN graphs are removed. The increments for $N_{g_i g_q}$, N_{g_i}, N_{g_q} are $\Delta N_{g_i g_q}$, ΔN_{g_i}, and $\Delta N_{g_q} = \gamma$, respectively. Let

$$\begin{aligned}
\tau &= N(N_{g_i g_q} + \Delta N_{g_i g_q}) - (N_{g_i} + \Delta N_{g_i})(N_{g_q} + \gamma) \\
\omega &= (N - N_{g_q} - \gamma)(N_{g_q} + \gamma) \\
\upsilon &= (N_{g_i} + \Delta N_{g_i})(N - N_{g_i} - \Delta N_{g_i})
\end{aligned}$$

Then the new correlation at the next outlook ($\mathcal{O}_{i+1}$) for graph g_i is

$$\phi^{\diamond}(g, q) = \frac{\tau}{\sqrt{\upsilon\omega}} \tag{4.11}$$

Here, we are trying to calculate the maximum value of $\phi^{\diamond}(g, q)$ for graph g_i, denoted by $\phi^{\diamond}_{max}(g, q)$. If $\phi^{\diamond}_{max}(g, q) < \theta$ within the next outlook buffer, we can

safely prune g_i.

$\phi^{\diamond}_{max}(g,q)$ can be achieved by maximizing the numerator (τ) and minimizing denominator (υ and ω) simultaneously.

4.4.1 Maximum Value of the Numerator

In order to maximize the numerator τ, let $\Delta N_{g_i g_q} = y$, $\gamma = y + c_1$, $\Delta N_{g_i} = y + c_2$, then

$$\tau = N(N_{g_i g_q} + y) - (N_{g_q} + y + c_1)(N_{g_i} + y + c_2)$$

Similar to the estimation process for γ which concludes that $\alpha \leq \gamma \leq \beta$, the increment ΔN_{g_i} can be estimated in the same way as we estimate ΔN_{g_q}, which results in $\zeta_{g_i} \leq \Delta N_{g_i} \leq \eta_{g_i}$. As a result, we have the following inequalities:

$$\begin{aligned} \alpha \leq y \leq \beta; \qquad & y + N_{g_i g_q} \geq 0; \\ \alpha \leq c_1 \leq \beta; \qquad & y + c_1 + N_{g_q} \geq 0; \\ \zeta_{g_i} \leq c_2 \leq \eta_{g_i}; \qquad & y + c_2 + N_{g_i} \geq 0; \end{aligned}$$

Lemma 2. *For a stream outlook $\mathcal{O}_i$, given N, N_{g_i}, $N_{g_i g_q}$, N_{g_q}, and ΔN, $\alpha \leq \gamma \leq \beta$, $\zeta_{g_i} \leq \Delta N_{g_i} \leq \eta_{g_i}$, the maximum possible value for τ, the numerator of the $\phi^{\diamond}(g,q)$ at the next outlook $\mathcal{O}_{i+1}$ is:*

$$\tau^{\diamond} = \begin{cases} h_\tau(\alpha) & : \quad t < 3\alpha \\ h_\tau(\hat{y}) & : \quad 3\alpha \leq t \leq 2\beta + \alpha \\ h_\tau(\beta) & : \quad t > 2\beta + \alpha \end{cases} \tag{4.12}$$

Here we have $h_\tau(y) = N(N_{g_i g_q} + y) - (N_{g_i} + y + \zeta_{g_i})(N_{g_q} + y + \alpha)$, $t = N - N_{g_i} - N_{g_q} - \zeta_{g_i}$, and $\hat{y} = (N - N_{g_i} - N_{g_q} - \zeta_{g_i} - \alpha)/2$.

Proof. The maximum value of τ can be derived by taking first and second partial

derivatives. $\frac{\partial \tau}{\partial c_1} = -(N_{g_i} + y + c_2) \leq 0$. τ increases monotonically as c_1 decreases. So τ achieves maximum value in the minimum value of c_1, i.e., $c_1 = \alpha$. Similarly, $\frac{\partial \tau}{\partial c_2} = -(N_{g_q} + y + c_1) \leq 0$, so τ is maximized when $c_2 = \zeta_{g_i}$;

$\frac{\partial \tau}{\partial y} = N - N_{g_i} - N_{g_q} - \alpha - \zeta_{g_i} - 2y$. And $\frac{\partial^2 \tau}{\partial y^2} = -2 < 0$. So τ has the maximum value at $\frac{\partial \tau}{\partial y} = 0$. The solution for this equation is $\hat{y} = (N - N_{g_i} - N_{g_q} - \zeta_{g_i} - \alpha)/2$. However, $\alpha \leq y \leq \beta$, the above value can be reached only if $3\alpha \leq N - N_{g_q} - N_{g_i} - \zeta_{g_i} \leq 2\beta + \alpha$. If $N - N_{g_q} - N_{g_i} - \zeta_{g_i} > 2\beta + \alpha$, $\frac{\partial \tau}{\partial y} = N - N_{g_i} - N_{g_q} - \alpha - \zeta_{g_i} - 2y > 2\beta + \alpha - \alpha - 2y > 0$. τ will achieve maximum value at $y = \beta$; Similarly, if $N - N_{g_q} - N_{g_i} - \zeta_{g_i} < 3\alpha$, $\frac{\partial \tau}{\partial y} < 2\alpha - 2y < 0$. τ will reach maximum value at $y = \alpha$. Now the Lemma 2 is proven. □

4.4.2 Minimum Value of the Denominator

In this subsection, we derive the minimum value of the denominator $\sqrt{\upsilon\omega}$.

Lemma 3. *Given N, N_{g_q}, and ΔN, $\alpha \leq \gamma \leq \beta$, the minimum possible value for ω is:*

$$\omega^{\diamond} = \min\{h_{\omega}(N_{g_q} + \alpha), h_{\omega}(N_{g_q} + \beta)\} \tag{4.13}$$

where $h_{\omega}(x) = x(N - x)$;

Proof. Since $h_{\omega}(x)$ is a quadratic function of x, it is concave and symmetric about $x = N/2$. Its minimum value will be located at either $N_{g_q} + \alpha$ or $N_{g_q} + \beta$. □

Lemma 4. *Given N, N_{g_q}, and ΔN, $\zeta_{g_i} \leq \Delta N_{g_i} \leq \eta_{g_i}$, the minimum possible value for υ is:*

$$\upsilon^{\diamond} = \min\{h_{\upsilon}(N_{g_i} + \zeta_{g_i}), h_{\upsilon}(N_{g_i} + \eta_{g_i})\} \tag{4.14}$$

where $h_{\upsilon}(x) = x(N - x)$;

Similar to Lemma 3, we can easily obtain Lemma 4.

4.4.3 Loose correlation upper bound

Theorem 2. *For any candidate graph g_i in the PG list of the current outlook $\mathcal{O}_i$, its correlation upper bound to the query graph g_q before reaching the next outlook $\mathcal{O}_{i+1}$ is:*

$$upper(\phi^{\diamond}(g_i, g_q)) = \phi^{\diamond}_{max}(g_i, g_q) = \frac{\tau^{\diamond}}{\sqrt{\omega^{\diamond}\upsilon^{\diamond}}} \tag{4.15}$$

where $\tau^{\diamond}$ follows Lemma 2, $\omega^{\diamond}$ follows Lemma 3, and $\upsilon^{\diamond}$ follows Lemma 4.

Proof. $\phi^{\diamond}_{max}(g_i, g_q)$ is achieved by maximizing the numerator (τ) and minimizing denominator (υ and ω) simultaneously according to Eq. (4.11). □

4.4.4 Heuristic Rule

To speed up the candidate pruning procedure, we derive a heuristic rule as follows.

Lemma 5. *Let $f^{\diamond}(a, b) = (\theta\sqrt{a(N-a)b(N-b)} + ab)/N$. Given current outlook $\mathcal{O}_i$, two graphs g_1 and g_2, $g_2 \subseteq g_1$, if $N_{g_2q} + \beta < f^{\diamond}(N_{g_1} + \zeta_{g_1}, N_{g_q} + \alpha)$, $N_{g_q} + \beta \leq N/2$, and $N_{g_1} + \zeta_{g_1} \leq N/2$, the correlation between g_2 and g_q at the next outlook $\mathcal{O}_{i+1}$*

$$\phi^{\diamond}(g_2, g_q) < \theta$$

Proof. Since $g_2 \subseteq g_1$, we have $N_{g_1} < N_{g_2}$, $N_{g_1} + \Delta N_{g_1} < N_{g_2} + \Delta N_{g_2}$. Taking the partial derivative of $f^{\diamond}$ with respect to a, we can easily know that, with $a \leq N/2$, $f^{\diamond}$ monotonically increases with a. Similarly, with $b \leq N/2$, $f^{\diamond}$ monotonically

increases with b. So we have

$$N_{g_2g_q} + \beta < f^{\diamond}(N_{g_1} + \zeta_{g_1}, N_{g_q} + \alpha) < f^{\diamond}(N_{g_2} + \Delta N_{g_2}, N_{g_q} + \Delta N_{g_q})$$

Here, $N_{g_2g_q} + \beta$ is the maximum possible value of $N_{g_2g_q}$ at outlook $\mathcal{O}_{i+1}$. Let $x' = N_{g_2} + \Delta N_{g_2}$, $y' = N_{g_q} + \Delta N_{g_q}$, $z' = N_{g_2g_q} + \Delta N_{g_2g_q}$; Then

$$\phi^{\diamond}(g_2, g_q) = \frac{Nz' - x'y'}{\sqrt{x'(N-x')y'(N-y')}} \tag{4.16}$$

As $z' \leq N_{g_2q} + \beta < f^{\diamond}(x', y')$, replacing z' with $f^{\diamond}(x', y')$ in the Eq. (4.16), we have

$$\begin{aligned} \phi^{\diamond}(g_2, g_q) &< \frac{\theta\sqrt{x'(N-x')y'(N-y')} + x'y' - x'y'}{\sqrt{x'(N-x')y'(N-y')}} \\ &= \theta \end{aligned} \tag{4.17}$$

Note that if $N_{g_2g_q} + \beta < f^{\diamond}(N_{g_1} + \zeta_{g_1}, N_{g_q} + \alpha)$, so is $N_{g_1g_q}$. This is because $N_{g_1} < N_{g_2}$, $N_{g_1} + \beta < N_{g_2} + \beta$. It means that $\phi^{\diamond}(g_1, g_q) < \theta$. Now we have proven the lemma 5. □

Applying the heuristic rule: The heuristic rule is integrated with the candidate checking process. Specifically, if we find that a graph g_1 is not a potential candidate, we check all its subgraphs in the candidate set. For each $g_2 \subseteq g_1$, if $\phi^{\diamond}(g_2, g_q) < \theta$, we can prune it according to Lemma 5.

4.5 Algorithm

Algorithm 1 lists the detailed procedures of the proposed algorithm for correlated graph query in data streams. Our algorithm handles the data stream in a batch

Algorithm 1 Correlated subgraph search for data stream

Require:

θ : query threshold value;
D_j : a new batch of graphs arriving at time point t_j;
PG: Potential correlated subgraphs of the current outlook;
w : number of batches in a sliding window;
ΔN: Number of graphs between two outlooks ($\Delta N = m \times |D_j|, m < w$);
$D_{win} = \bigcup\{D_{j-\iota} | \iota = 1, \cdots, w\}$; //*current sliding window*;
t_{cp}: The batch ID of the previous outlook;

Ensure:

A_g: Answer set of correlated subgraphs w.r.t. g_q;

1: $A_g = \varnothing$;
2: **while** A new graph batch D_j arrives **do**
3: $D_{win} \leftarrow D_{win}/D_{j-w}$; $D_{win} \leftarrow D_{win} \bigcup D_j$;
4: **if** t_j is an outlook (*i.e.*, $j - t_{cp} == m$) **then**
5: $PG \leftarrow$ Rebuild the candidate set by Algorithm 2;
6: $t_{cp} \leftarrow j$;
7: **else**
8: $\mathcal{T}_q^j \leftarrow$ Projected database of the query graph g_q *w.r.t.* the current batch D_j;
9: **for** each $g_i \in PG$ **do**
10: Increasing the frequency of $N_{g_i g_q}$ from D_q^j, N_{g_i} from current batch D_j;
11: Decrease the frequency of $N_{g_i g_q}$, N_{g_i} in batch D_{j-w};
12: **for** each $g_i \in PG$ **do**
13: **if** $\phi(g_i, g_q) \geq \theta$ **then**
14: $A_g \leftarrow A_g \bigcup g_i$;
15: Output A_g when necessary;

Algorithm 2 Building the candidate list

Require:

θ : query threshold value;

w : number of batch in the sliding window;

$D_{win} = \bigcup\{G_{j-\iota} | \iota = 0, \cdots, w-1\}$; //*current window*

Ensure:

PG: The potential correlated subgraphs;

1: $PG = \varnothing$;

2: Estimate the statistics of γ, $\alpha \leq \gamma \leq \beta$;

3: $\mathcal{T}_q \leftarrow$ The projected database of query graph g_q *w.r.t.* to the current sliding window D_{win};

4: Using gSpan algorithm [151] to mine the frequent subgraphs $\mathcal{C}$ from $\mathcal{T}_q$ with the lower bound $lower(N_{g_i g_q})$ determined by Theorem 1;

5: Sorting $\mathcal{C}$ in a descendant order according to their sizes of edges;

6: **for** each $g_i \in \mathcal{C}$ **do**

7: Compute $upper(\phi^{\diamond}(g_i, g_q))$ according to Theorem 2;

8: **if** $upper(\phi^{\diamond}(g_i, g_q)) \geq \theta$ **then**

9: $PG \leftarrow PG \bigcup g_i$;

10: **else**

11: $H \leftarrow \bigcup\{g' | g' \subseteq g_i, g' \in \mathcal{C}, N_{g' g_q} + \beta < f(N_{g_i} + \zeta_{g_i}, N_{g_q} + \alpha), N_g + \eta_g < N/2, N_{g_q} + \beta < N/2\}$;

12: $\mathcal{C} \leftarrow \mathcal{C} - H$;

13: **return** PG;

by batch manner. As soon as a new data batch D_j arrives at time point t_j, the sliding window will discard the most outdated data batch to ensure that the window covers w data batches, including the newly coming data batch (step 3 in Algorithm 1). Then we check whether time point t_j is an outlook $\mathcal{O}$ (step 4). If t_j is an outlook, we call Algorithm 2 to rebuild the candidate list PG (steps 5-6); otherwise, we update the frequency information of each candidate g_i ($g_i \in PG$), from the outdated and newly arriving data batches (steps 8-11). Next, if a candidate's correlation is above θ, we add it into the answer set A_g (step 12-14). Finally, we output the answer set A_g when there is a demand from the user.

Our candidate building procedure for each outlook $\mathcal{O}$ is illustrated in Algorithm 2. We use Poisson/Skellam Distribution to estimate γ, $\alpha \leq \gamma \leq \beta$ in the next outlook (line 2 of Algorithm 2). Because graph isomorphism is NP-complete,

we reduce the number of graph isomorphism test by retrieving from the projected database of query graph g_q rather than using the original windows of graph. In step 4, We mine a set of frequent subgraphs from $\mathcal{T}_q$ using a lower bound of threshold $lower(N_{g_i g_q})$, which is determined by Theorem 1. This threshold takes the possible frequency increment quantity of N_{g_q} into consideration and guarantees that the graph is a possible graph with N_{g_q} in the range of $[N_{g_q} - \alpha, N_{g_q} + \beta]$. In this way, all the potential candidates will be kept before reaching next outlook. In steps (6-12), for each candidate graph, we compute its correlation upper bound. If a graph g_i's upper correlation bound is greater than the given threshold θ, we add g_i into PG list, otherwise, we apply a heuristic rule to prune the candidates and speed up the process.

4.6 Experimental Result

4.6.1 Experiment setup

In this section, we report our experimental results. The graph data stream is collected from a real-world dataset of the NCI Open Database Compounds [1], which contains compound structures of cancer and AIDS data. The original dataset contains about 249,000 graphs. After preprocessing and removing some disconnected graphs, we have a data stream with about 233,000 graphs.

We compare our algorithms with an exhaustive search method (denoted as **rCGSearch**) in terms of system runtime[2]. When implementing **rCGSearch**, *i.e.*, whenever a new batch of graphs arrives, rCGSearch restarts to involve the CGSearch algorithm [71] in the sliding window. rCGSearch is a precise method in the sense that it can return all the true answers with zero false positives and false negatives, but it is computationally expensive which makes it unsuitable for stream based applications.

Precision and recall are widely used to measure the performance of an algorithm [11]. Suppose the true answer set of correlated graphs is T_g, and the answer

[1] http://cactus.nci.nih.gov/ncidb2/download.html

[2] Note that the memory consumption of CGStream and rCGSearch are almost the same and fixed, as they both store the window of graphs for query process.

set returned by our algorithm is A_g. The **precision** is defined as $|T_g \bigcap A_g|/|A_g|$, and the **recall** is denoted as $|T_g \bigcap A_g|/|T_g|$.

Because we maintain a candidate set PG over stream and update the frequency information of each candidate, the answer will be returned as long as it is stored in PG. In other words, the genuine correlation values of the retrieved graphs to the query graph g_q are all above θ, which asserts that the recall of our algorithm is 1.

To calculate the query precision values, we randomly select 30 graphs as the query graphs. For each selected query graph, its support value in the whole data stream is in range [0.02, 0.05]. During the query process, we sequentially move the sliding window one batch at a time and evaluate the precision of the correlated graphs in each sliding window over the whole stream. Suppose there are ψ batches of graphs over data stream. The average precision on a data stream for a query is computed as $\overline{Precision} = \frac{1}{\psi}\sum_{\iota=1}^{\psi} P_\iota$, where P_ι is the precision in window $D_{win} = \bigcup\{D_\iota | n - w + 1 \leq \iota \leq n\}$ (the most recent w batches). We calculate average runtime in a similar way. The results for 30 graphs in terms of these measures are averaged again and reported as the final results.

We study the performance of our algorithm with various parameters. Unless specify otherwise, we set the default values θ=0.7, w=20, m=10, and $|D_j|$=3000.

4.6.2 System runtime performance

Pruning Effectiveness of CGStream: In order to assess the effectiveness of different parts of pruning techniques in our CGStream algorithm, we first remove the heuristic rules, and then remove both heuristic rules and the upper correlation bound to investigate the system runtime performance. Table 4.1 summarizes the system runtime in each time point (including outlook points and non-outlook points) and the accumulative runtime of the whole stream.

The results in Table. 4.1 show that the system runtime at the outlooks is significantly larger than at the non-outlook points. This is because CGStream needs to query from the whole window D_{win} at outlooks while at non-outlook points it only needs to check the PG list. Meanwhile, it can be seen that after we remove the heuristic rule, the runtime at the outlooks increases significantly, which

Table 4.1: Effectiveness of Pruning in CGStream with $\theta = 0.8$ (seconds)[Acc. Time - accumulative runtime]

	Runtime		Acc. Time
	Outlook	Non-Outlook	
CGStream	107.0	4.9	476.2
No Rule	195.5	5.0	768.8
No Rule&UpBound	204.2	11.6	956.5

reflects the contributions of the simple heuristic rule for pruning. Furthermore, if we remove both heuristic rule and upper correlation bound from CGStream algorithm, it will not only increase the runtime at outlooks substantially, but also increase the runtime at non-outlook points. Overall, the accumulative time (column 4 in table 4.1) in the whole stream will increase if we remove either heuristic rule or upper correlation bound.

The above results conclude that the upper correlation bound and heuristics are essential for CGStream.

Algorithm performances with Different Query Thresholds: To study the performance of our algorithm with respect to different query threshold (θ) values, we vary the θ values and report the the system runtime at each time point (*i.e.* each sliding window) and total system accumulative runtime in Fig. 4.2 and Fig. 4.3.

Fig. 4.2 shows that the proposed CGStream algorithm significantly outperforms the exhaustive algorithm in terms of system runtime. Take $\theta = 0.6$ as an example, it only takes about 15 seconds for CGStream to retrieve the answers at most time stamps, whereas rCGSearch needs about 380-400 seconds. CGStream is more than 20 times efficient than rCGSearch. Even at the outlooks, CGStream needs less time (occasional a little more time) than rCGSearch to build the candidate list, this is because our loose correlation upper bound and heuristic rule can reduce the number of candidates and speed up the computation process.

We also illustrate the accumulative runtime in each time point in Fig. 4.3. It is clear that the accumulative runtime of rCGSearch increases dramatically as streaming batch data continuously arrives. In contrast, CGStream's accumulative time climbs very slowly except for outlooks, where large jumps can be observed.

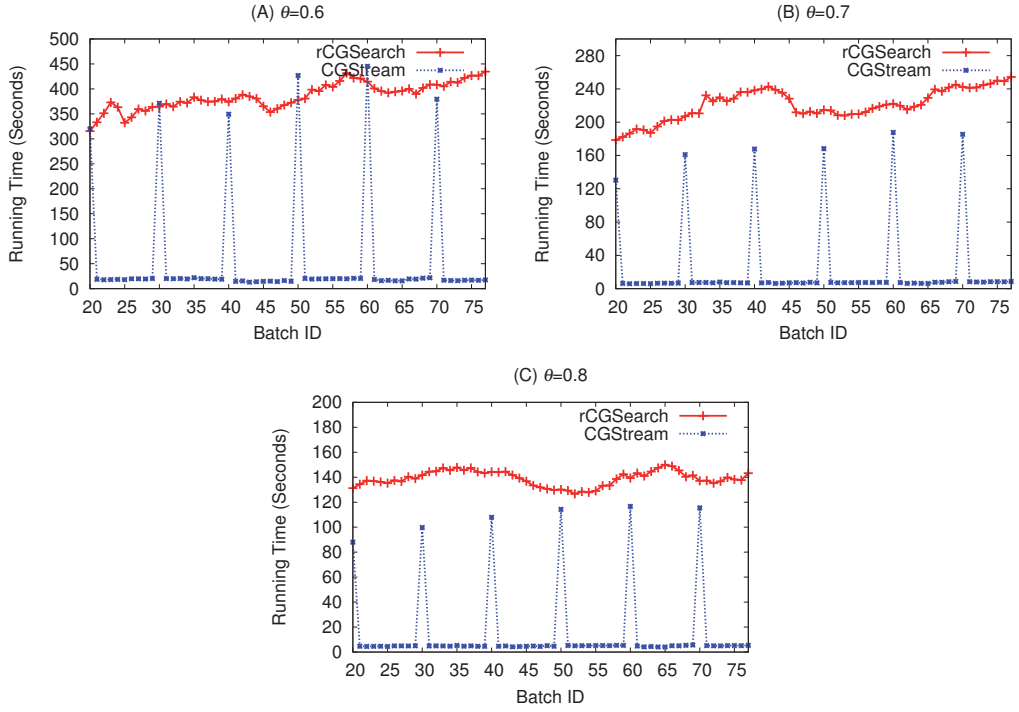


Figure 4.2: System runtime consumption with respect to different θ values.

The results in Fig 4.2 and Fig. 4.3 show that when the threshold θ is relatively small, both rCGSearch and CGStream need more time to retrieve the results in each batch. This is because a smaller threshold θ results in more candidates returned by the frequent subgraph mining step, which in turn calls for more time to check and prune the candidate list.

Algorithm performances with different window size w Values: In Fig. 4.4 and Fig. 4.5, we also report algorithm performance with respect to different sliding window sizes.

As expected, Fig. 4.4 shows that CGStream increases much more slowly than rCGSearch in terms of system accumulative runtime when we vary w values. The results are consistent with that we reported for θ previously. Meanwhile, when we increase the window size, Fig. 4.5.(A) and Fig. 4.5.(B) show that CGStream requires more time to build the candidate list at the outlooks, which results in a larger jump in accumulative system runtime at the outlooks in Fig. 4.5.(C) and Fig. 4.5.(D). This is because the algorithm needs to go through more graphs

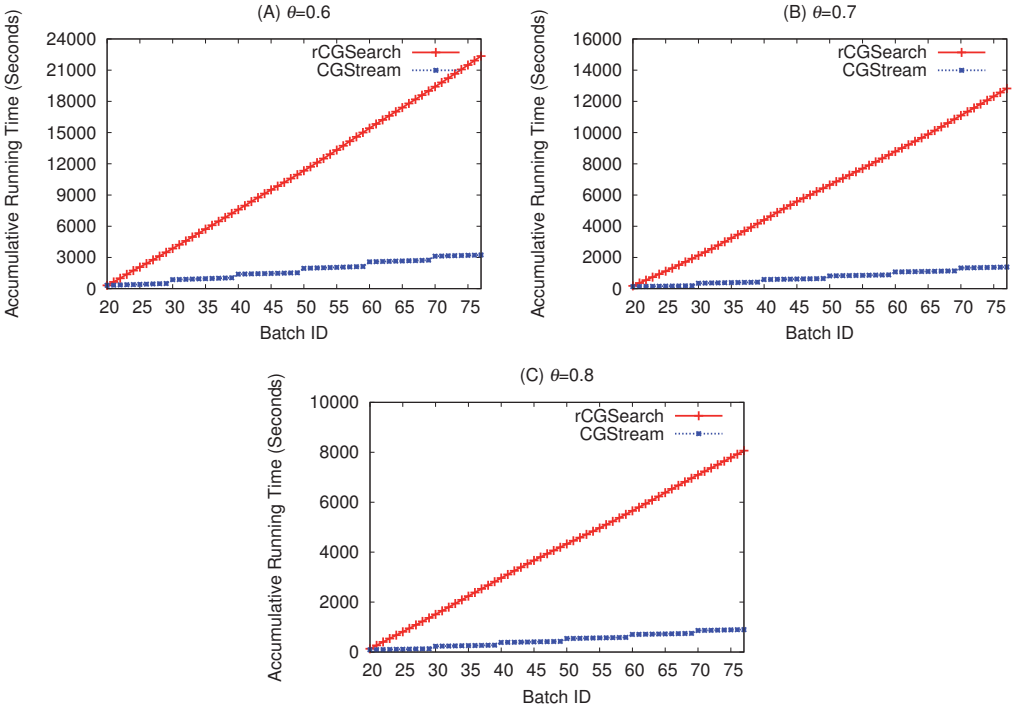


Figure 4.3: System accumulative runtime consumption with respect to different θ values.

when we increase the window size. In real applications, the appropriate size of window may depend on the domains of application and specific user settings.

Algorithm performances with Different Batch Sizes: Fig. 4.6 illustrates the results when using different batch sizes (*i.e.* different $|D_j|$ Values). The experimental results, once again, demonstrate that CGStream can greatly reduce the computation cost required by rCGSearch, because it avoids involving CGSearch algorithm repeatedly when updating the sliding window.

Algorithm Performances w.r.t. Different Outlook Frequencies In Fig. 4.7 we also report the algorithm performance with respect to different m values, *i.e.*, how frequent we set a stream outlook and rebuild the candidate list (the m value also determine the number of graphs ΔN between two outlooks). It is obvious that the larger the m values, the less accumulative time is needed by CGStream algorithm, because it involves less outlooks and less candidate rebuilding procedures (which is the most time consuming process). However, as

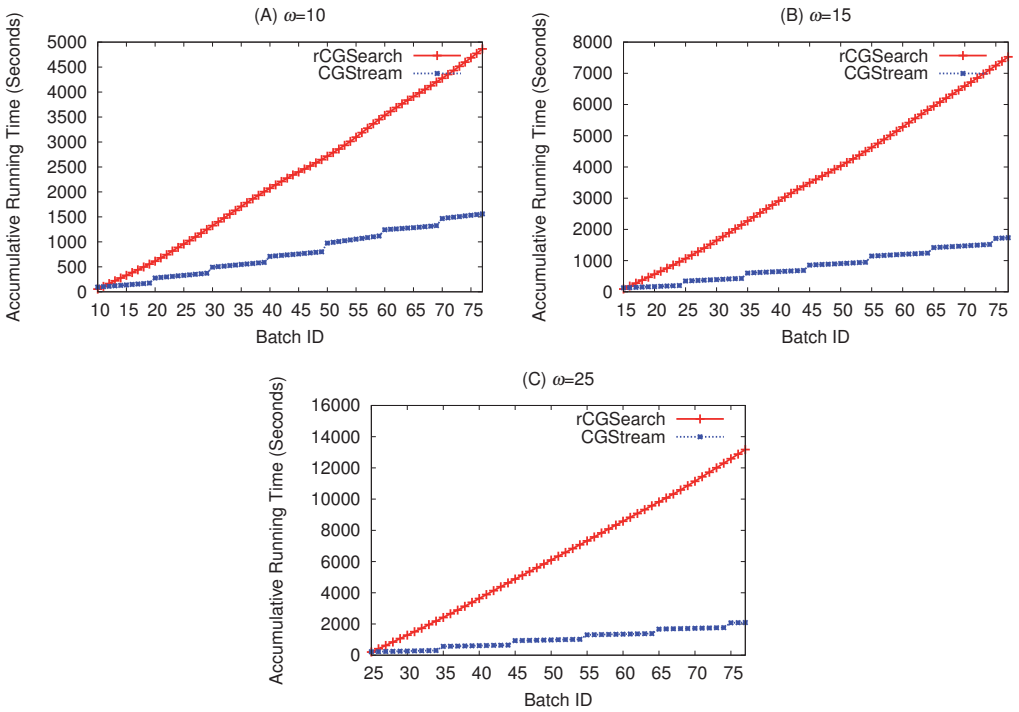


Figure 4.4: System accumulative runtime consumption with respect to different w values.

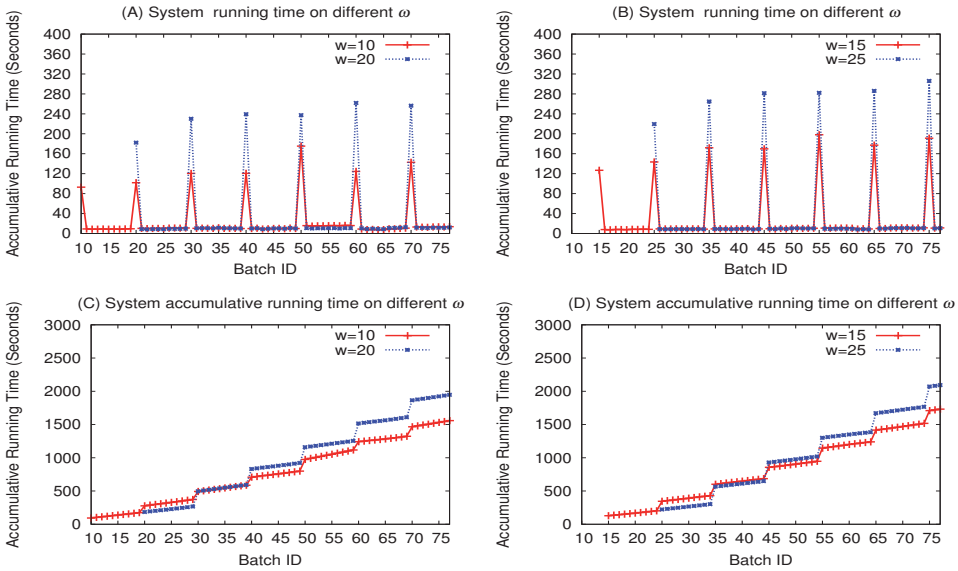


Figure 4.5: Comparison on different w values. (A) and (B), system runtime in each time point; (C) and (D), system accumulative runtime in each time point.

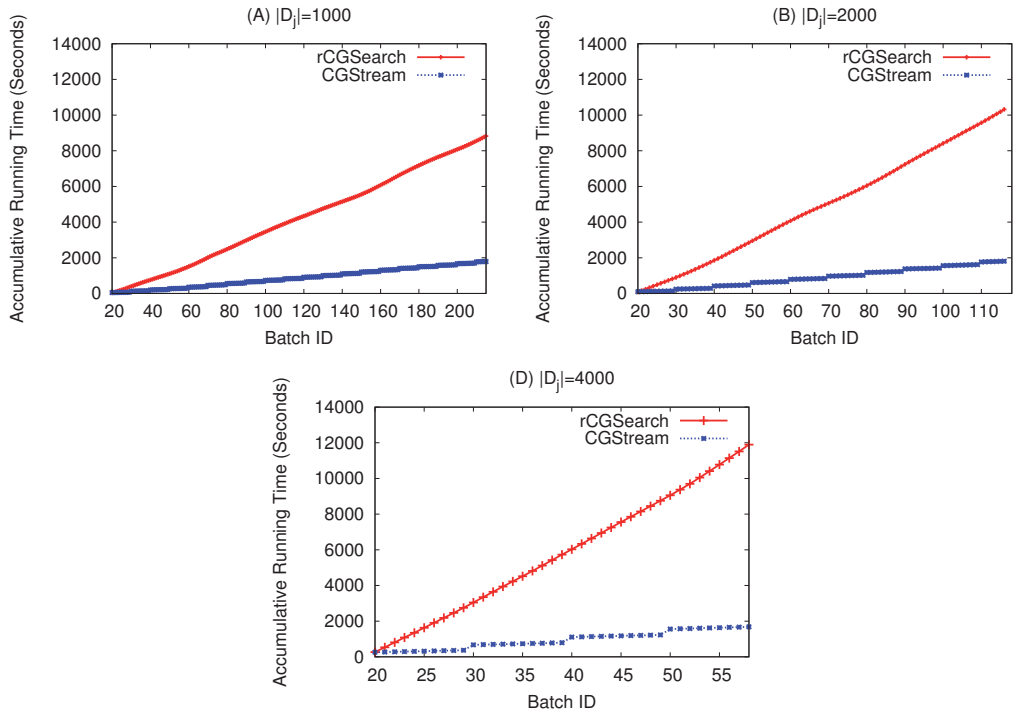


Figure 4.6: System accumulative runtime consumption with different $|D_j|$ values.

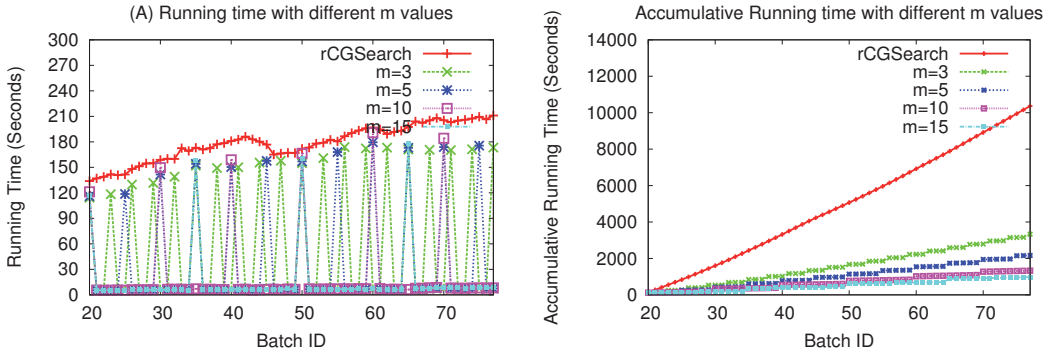


Figure 4.7: Comparison on different m values. (A) system runtime, (B) system accumulative runtime.

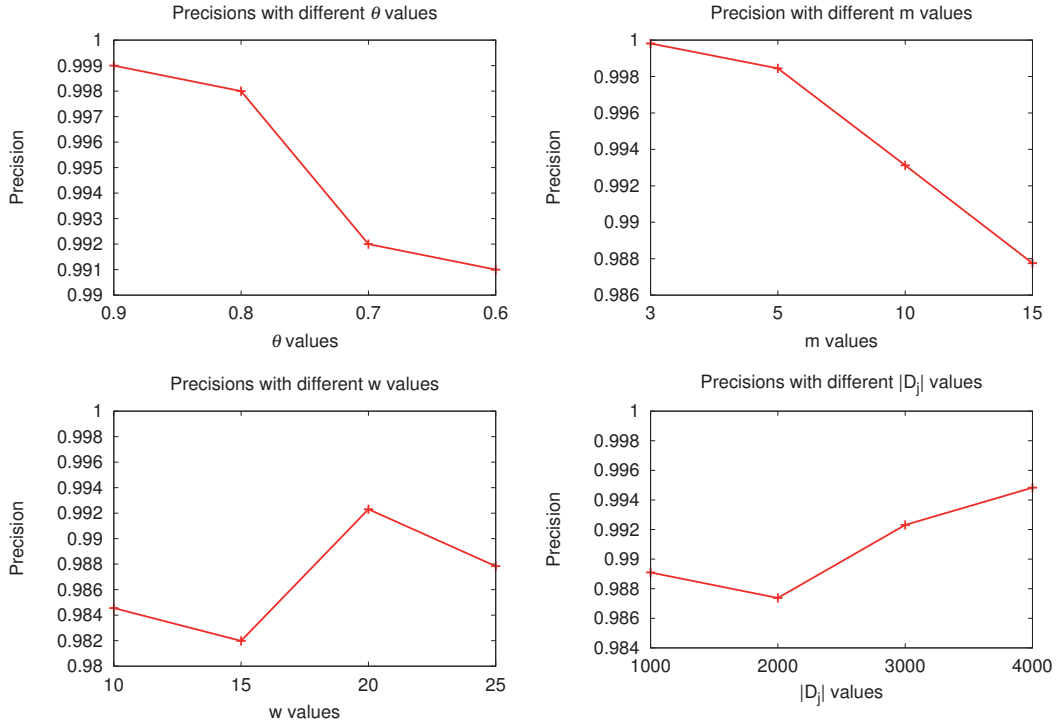


Figure 4.8: Query precisions with respect to different parameters settings.

we will see in latter section, the precision may decrease if we increase m values. There is a tradeoff between time consumptions and precision.

4.6.3 Query Precision

In Fig. 4.8, we report the query performance of the CGStream algorithm with respect to different parameters.

Fig. 4.8.(A) shows that the precision drops slightly when we decrease the θ values. However, the precision is above 0.99 in our algorithm for all θ values, which reflects the high accuracy of our algorithm.

In Fig. 4.8.(B), the results show that the increasing of m values (outlook buffer size) will decrease the query precision. As we have mentioned in Section 4.6.2, increasing the m values reduces the number of outlooks in the stream. Because outlooks consumes most of the system runtime, increasing m values reduces the accumulative time taken by CGStream algorithm. On the other hand, with a

relatively large m value, the outlook buffer size will increase accordingly, which will make the parameter estimations for $\alpha \leq \gamma \leq \beta$ to be less accurate. As a result, it will decrease the algorithm's query precision. The tradeoff between the runtime consumption and precision may be determined depending on domain applications and user requirements.

Fig. 4.8.(C) and Fig. 4.8.(D) show that the precision will change slightly when we vary either window size w or batch size $|D_j|$. Nevertheless, the query precision is always very close to 1, which indicates that our algorithm is highly accurate in practice.

4.7 Conclusions

In this chapter, we investigated the problem of query correlated graphs from data stream, by using a sliding window which covers a number of consecutive batches of stream data records. We argued that, for data streams with dynamic increasing volumes, simple exhaustive search for correlated graphs needs to repeatedly carry out the query process, which is computationally expensive. By setting stream outlooks and considering the possible increment of query graph within two adjacent outlooks, we derived a lower frequency bound to mine a set of frequent subgraphs as candidates. An upper correlation bound and a heuristic rule are also derived to prune the candidates in the process of candidate checking. Experimental results demonstrate that our proposed algorithm CGStream is several times more efficient than the exhaustive search method in terms of the system runtime consumption, and achieve high performance in terms of query precision.

Part II

Graph Stream Classification

Graph Stream Classification: Overview

Graph classification is becoming increasingly important and has attracts wide interest in recent years. However, existing studies on graph classification mainly focus on static graph dataset. In reality, graph data is becoming more and more dynamic. Instead of being a static dataset, graph data is increasing and evolving in a streaming fashion. For example, an online user's browsing pattern, with respect to all web pages, can be regarded as a graph. The browsing patterns of all users will form a graph stream. Each scientific publication and its references can be represented as a graph (see Fig. 1.3), so all scientific papers, collected in chronological order, will form a graph stream with increasing volumes.

In stream scenarios, classifying graph data is a very challenging task. This is because the decision concepts (decision boundaries) of the graph data may gradually (or rapidly) change, that is, the concept is drifting in the stream. The challenges will be further complicated when there are insufficient labeled graph data or the underlying class distributions of the graph data are imbalanced and noisy. In Part II, we will study the following problems:

- **Graph Stream Classification using Labeled and Unlabeled Graphs:** Due to the complexity of network structures, labeling graphs usually requires experts to investigate the structures carefully. To reduce the human resource of labeling graphs, a possible way is to combine both labeled and unlabeled graphs to construct classifier models. In Chapter 5, we will study how to select discriminative subgraphs with minimum redundancy for semi-supervised graph stream classification.

- **Imbalanced and Noisy Graph Stream Classification:** Meanwhile, in real-life application, especially for graph based domains, the class distributions of data are inherent imbalanced. In the NCI chemical compound database, there are only about 5% percent of chemical compounds which are active to the cancer bioassay test, whereas 95% of them are inactive to the cancer bioassay test (see table 3.2). In Chapter 6, we will study how to perform graph stream classification with imbalanced class distributions and noise.

Chapter 5

Graph Stream Classification using Labeled and Unlabeled Graphs

5.1 Introduction

Graph classification is becoming increasingly important in recent years due to rapid growth of complex data which exhibit structural and interdependent relationships.

Unlike conventional data, where each instance is represented in a feature-value vector format, graphs exhibit node-edge structural relationships and have no natural vector representation. As a result, a common practice is to select a set of discriminative subgraph as features and transfer graphs into vectors [70] in Euclidean space, so that traditional machine learning algorithms such as Support Vector Machines (SVM) and Decision Tree can be applied.

When selecting subgraph features, common methods use an evaluation metrics, such as the frequency, to select a number of important subgraph features. The graph data can then be represented by using the selected features in a vector

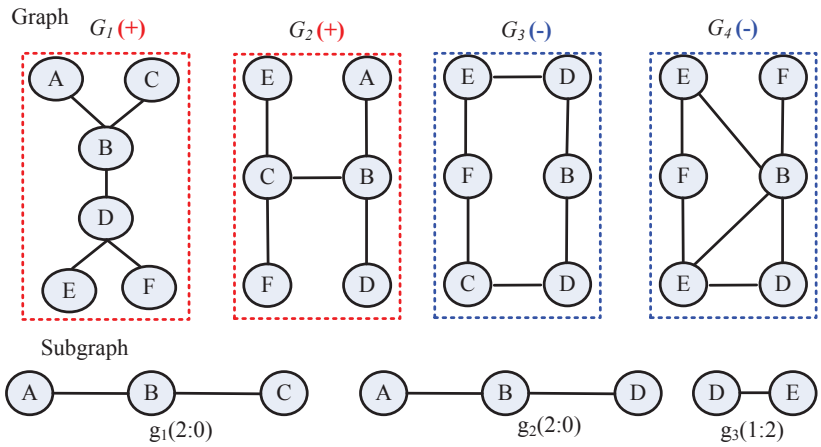


Figure 5.1: An example demonstrating subgraph correlations: G_1 and G_2 are positive graphs (+), G_3 and G_4 are negative graphs (-). Each subgraph, g_1, g_2, or g_3, is marked with its frequency occurring in positive *v.s.* negative graphs.

space (depending on whether a graph contains specific features or not). While a large number of subgraph feature selection methods exist, they consider subgraphs as independent observations without realizing that subgraph features are normally generated from the same set of graphs. As a result, subgraph features may share high correlations, which is one of the major factors attributed to the performance loss of learning methods. As shown in Fig. 5.1, subgraphs g_1 and g_2 only appear in positive graphs, whereas g_3 appears in both positive and negative graphs. If evaluated separately, g_1, g_2 are more informative than g_3. However, g_1 and g_2 are highly correlated and redundant. A good approach is to include either g_1 and g_3, or g_2 and g_3 to form a two feature set for graph classification.

In stream scenario, subgraph feature selection can be even more complicated. This is because the decision concepts (decision boundaries) of the graph data may gradually (or rapidly) change, *i.e.* the concept drifting in the stream. In order to rapidly capture the concept changes in the graph stream, the graph feature selection module should take the dynamic graph stream as a whole to select informative and less redundant features. Unfortunately, existing graph feature selection methods all work on static graph set. No effective strategy exists to select informative subgraph features from graph streams. While a trivial solution is to partition graph stream into a number of stationary subsets (or chunks) and

carry out feature selection in each individual subset, such a simple solution does not allow graph subsets to collaborate with each other to select highly effective subgraph features for graph streams.

In summary, when selecting subgraph features from graph stream for classification, we should take the following three factors into consideration to ensure that the whole framework is effective and efficient.

- **Identifying informative subgraphs with minimum redundancy:** Finding a set of informative subgraph features with minimum redundancy can ensure that the succeeding learning methods can achieve high accuracy for graph stream classification.

- **Capturing concept drifting in streams:** Concept drifting represents emerging changes in the streams. Our model must be able to capture such changes and emphasize on the instances represented by drifting concepts.

- **Combining labeled and unlabeled graphs:** The continuous increasing volumes of the graph stream makes the graph labels very difficult to obtain. Our model should take advantage of the large quantify of unlabeled graphs to boost graph stream classification.

Motivated by the above observations, we report in this chapter, gSLU, a graph stream classification framework using both labeled and unlabeled graphs. Instead of limiting subgraph features to labeled samples, gSLU combines both labeled and unlabeled graphs to generate a rich set of subgraph features for assessment. To identify informative subgraphs with minimum redundancy, we propose a subgraph assessment criterion to assess the informativeness of individual features and the redundancy of the whole feature set at the same time. To capture instances represented by emerging concept drifting in graph streams, we employ a dynamic instance weighting mechanism, where graphs misclassified by the existing model receive a higher weight so the subgraph feature selection can emphasize on difficult graphs to find effective features to represent them. Experiments on real-world applications demonstrate that gSLU is effective for selecting informative and minimum-redundancy subgraph features to build accurate classifiers.

The proposed graph ensemble model is able to tackle concept drifting in graph streams for classification.

The remainder of the chapter is structured as follows. The problem definition and the overall framework is discussed in Section 5.2. Section 5.3 reports the proposed subgraph feature assessment criterion. The gSLU framework is reported in Section 5.4, followed by the experiments in Section 5.5. We conclude the chapter in Section 5.6.

5.2 Problem Definition & Overall Framework

In this section, we first review some several important notations and discuss overall framework for semi-supervised graph stream classification.

In our study, each graph G_i is associated with a class label $y_i \in \mathcal{Y}$. For binary classification problem, we have $y_i \in \mathcal{Y} = \{-1, +1\}$. A graph G_i is either labeled (denoted by G_i^l) or unlabeled (denoted by G_i^u). In this chapter, we also use G^l and G^u to denote labeled and unlabeled graphs, respectively.

Graph Stream (Definition 4): A graph stream $\mathcal{S} = \{\cdots, G_i, G_{i+1}, G_{i+2}, \cdots\}$ contains an increasing number of graphs in a streaming fashion. In this chapter, we consider that a graph chunk $D_t = D_t^l \bigcup D_t^u$ contains a fixed number of graphs collected from a consecutive stream region, where D_t^l and D_t^u denote labeled and unlabeled graphs, respectively.

By using the graph chunk notation, a graph stream $\mathcal{S}$ can be denoted as a collection of chunks $\mathcal{S} = \{D_1, \cdots, D_t\}$.

Pearson's Correlation Coefficient (Definition 6): Given two subgraph patterns g_p and g_q, and a graph set D, the Pearson's Correlation Coefficient between g_p and g_q over the graph set D can be defined as:

$$\phi(g_p, g_q) = \frac{N^D N^D_{g_p,g_q} - N^D_{g_p} N^D_{g_q}}{\sqrt{N^D_{g_p}(N^D - N^D_{g_p}) N^D_{g_q}(N^D - N^D_{g_q})}} \tag{5.1}$$

In Eq.(5.1), N^D denotes the total number of graphs in graph set D. $N^D_{g_p}$, $N^D_{g_q}$, and $N^D_{g_p,g_q}$ denote the number of graphs in D containing g_p, g_q, and g_p and g_q,

respectively.

Subgraphs with high Pearson's correlation coefficient indicate that subgraphs co-occur in the same graphs. So Pearson's correlation provides a measure to assess the redundancy of the subgraph features, regardless of the labels of the underlying graph set.

Given a graph stream $\mathcal{S} = \{D_1, D_2, \cdots, D_t\}$ with a number of consecutive graph chunks, each chunk $D_t = \{G_1, \cdots, G_n\}$ containing some labeled and unlabeled graphs, the **aim** of the graph stream classification is to build a prediction model from the most recently observed k chunks $(D_{t-k+1}, \cdots, D_{t-1}, D_t)$ to predict graphs in the next chunk D_{t+1} with the maximum accuracy.

5.2.1 Overall Framework

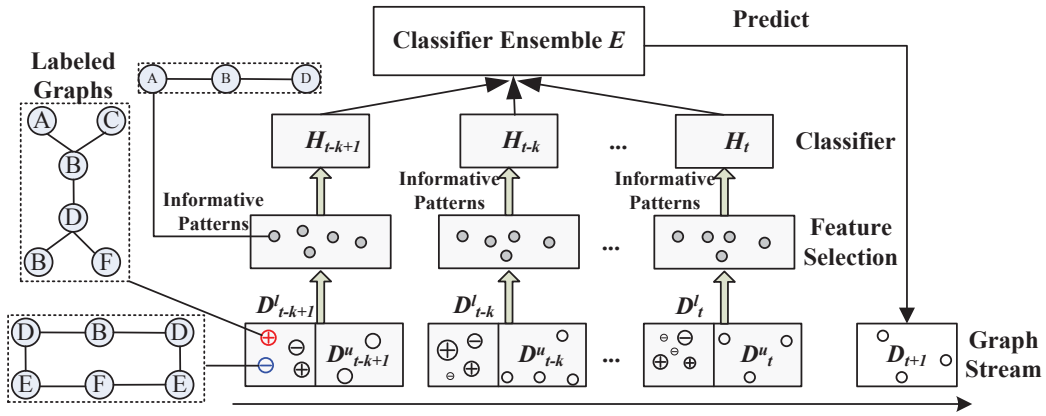


Figure 5.2: A framework for semi-supervised graph data stream classification. The graph stream is divided into chunks. In each chunk $D_t = D_t^l \cup D_t^u$, the circles with '+' indicate positive graphs, and the circles with '-' are negative graphs. The size of a circle represents its weight in the chunk. In our graph stream scenario, the weight of a graph is dynamically tuned by an ensemble of classifiers built in previous chunks. In the current chunk, by taking the weight of each graph into consideration, we select a set of optimal informative features with minimum redundancy. An ensemble of classifier is built from the most recent k chunks to predict graphs in the yet-to-come chunk.

In this chapter, we propose an ensemble framework, with a set of minimum-redundancy subgraph features being extracted from each graph chunk to train ensemble classifiers. Our framework, as shown in Fig. 5.2, contains three key

components: (1) partitioning graph stream into chunks, (2) selecting informative and minimum-redundancy subgraph features from each chunk, and (3) forming an ensemble model by combining classifiers trained from individual chunks. To put all three components into a unified framework, we employ an instance weighting mechanism to allow multiple graph chunks to work in a collaborative way to tackle concept drifting in graph streams. As soon as a graph chunk D_t is collected, the overall framework proceeds as follows:

- **Instance Weighting:** We use models trained from the past graph chunks to carefully weight graphs in the most recent chunk D_t with misclassified graphs (*i.e.* samples representing concept drifting in stream) receiving higher weight values.
- **Subgraph Feature Selection:** A set of informative subgraph features with minimum-redundancy are selected to represent the weighted graphs in the current chunk D_t.
- **Updating Ensemble:** By using selected features, a classifier $\mathcal{H}_t$, is trained from chunk D_t and is included into the ensemble to predict graphs in a future chunk D_{t+1}

In the following sections, we first propose our subgraph feature selection module, and then discuss detailed procedures of gSLU in Section 5.4.

5.3 Minimum Redundancy Subgraph Feature Selection

Given a chunk of graph data D_t, let $\mathcal{F}$ denote the complete set of subgraphs in D_t, and $\mathbf{g} = \{g_1, \cdots, g_m\}$ be a small set of subgraphs selected from $\mathcal{F}$. Our subgraph feature selection aims to simultaneously achieve two goals: (1) maximize the informativeness (discriminative power) of the selected feature set $\mathbf{g}$ for classification, and (2) minimize the redundancy between subgraph features in $\mathbf{g}$. Let $\mathcal{I}(\mathbf{g})$ be a function to measure the informativeness of $\mathbf{g}$, and $\mathcal{R}(\mathbf{g})$ be a function to

assess the redundancy in $\mathbf{g}$. The above objective can be formalized in Eq.(5.2), where $|\cdot|$ represents the cardinality of a set, and m is the number of features to selected from D_t.

$$\begin{aligned} \mathbf{g}^{\star} &= \arg\max_{\mathbf{g}\subset\mathcal{F}}(\mathcal{I}(\mathbf{g})) \\ s.\ t. \quad & (1)\ |\mathbf{g}| \leq m\ and \\ & (2)\ \mathcal{R}(\mathbf{g}) \leq \mathcal{R}(\mathbf{g}'), \forall \mathbf{g}' \subset \mathcal{F},\ |\mathbf{g}'| == |\mathbf{g}|,\ \&\ \mathbf{g}' \neq \mathbf{g} \end{aligned} \tag{5.2}$$

The objective function in Eq.(5.2) indicates that the optimal subgraph features $\mathbf{g}^{\star}$ should have (1) maximum discriminative power, *i.e.*, $\max(\mathcal{I}(\mathbf{g}))$, and (2) minimum redundancy between subgraph features, *i.e.*, $\min(\mathcal{R}(\mathbf{g}))$.

Note that there are already also some feature selection studies [30, 113] that consider maximizing the relevance and minimizing the redundancy of the feature set. However, these algorithms were not designed for graph classification. In other words, they cannot integrate feature selection with the subgraph enumeration process. In contrast, our algorithm can effectively integrate subgraph enumeration process with the feature selection module.

5.3.1 Informativeness of the Feature Set

To discover the set of informative features, we need to measure the informativeness of a feature set $\mathbf{g}$, *i.e.*, $\mathcal{I}(\mathbf{g})$. To calculate $\mathcal{I}(\mathbf{g})$, we impose constraints on the labeled graphs in D_t. For two graphs G_i and G_j, if they have the same class labels, there is a pairwise must-link constraint between G_i and G_j in a must-link set $\mathcal{M}$. If G_i and G_j have different class labels, there is a cannot-link constraint between them in a cannot-link set $\mathcal{C}$. Such an idea is previously employed in constraint based clustering [137]. If we take labeled D_t^l and unlabeled D_t^u graphs in D_t into consideration, a good feature set should satisfy constraints as follows.

- **Weighted Must Link:** if there is a must link between G_i and G_j, their subgraph feature vectors $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ should be similar to each other. In a data stream scenario, each graph G_i is associated with a weight value w_i. For each graph pair, the higher the total weight of two graphs in the must

link set, the more impact the constraint will have for selecting features to represent their resemblance.

- **Weighted Cannot Link:** if there is a cannot link between G_i and G_j, their subgraph feature vectors $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ should be distinct from each other. The higher the total weight of G_i and G_j, the more impact the constraint will have for selecting features to represent the distinctness between the two graphs.

- **Weighted Separability:** if two graphs G_i and G_j are unlabeled, their subgraph feature vectors $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ should be different. It is similar to PCA's assumption, which aims to find the components with largest possible variance.

By integrating instance weight, we can adjust the weight of the graphs to emphasize on some important samples. As a result, our framework can effectively capture the concept drifting underlying the graph stream. In this section, we take instance weight as given values. In Section 5.4, we will provide solutions for automatically calculating the instance weight.

Taking the above constraints into consideration, we derive a criterion for measuring the informativeness as follows:

$$\begin{aligned}\mathcal{I}(\mathbf{g}) = &\frac{1}{2A}\sum_{y_i y_j=-1}(\boldsymbol{x}_i-\boldsymbol{x}_j)^2(w_i+w_j)\\ &-\frac{1}{2B}\sum_{y_i y_j=1}(\boldsymbol{x}_i-\boldsymbol{x}_j)^2(w_i+w_j)\\ &+\frac{1}{2C}\sum_{G_i,G_j\in D_t^U}(\boldsymbol{x}_i-\boldsymbol{x}_j)^2(w_i+w_j)\end{aligned} \tag{5.3}$$

where w_i and w_j are the weights for G_i and G_j, respectively. $A = \sum_{y_i y_j=-1}(w_i + w_j)$, $B = \sum_{y_i y_j=1}(w_i + w_j)$, and $C = \sum_{G_i,G_j\in D_t^U}(w_i + w_j)$. A, B, and C assess the total weights of constraints in the must-link, cannot-link, and unlabeled set. Combining weighted must link, cannot link and separability can better capture the underlying distribution of graph data.

Significance of Unlabeled Graphs: A key property of Eq. (5.3) is that it considers both labeled and unlabeled graphs. The benefit of unlabeled graphs is two-fold: (1) because incorporating unlabeled graphs significantly increases the total number of graphs, it will alleviate the graph data sparsity issue and help enrich the frequent subgraphs. As a result, the subgraph feature space becomes more dense, through which a good set of subgraph features can be discovered; (2)because we emphasize on features which can better separate unlabeled graphs, we can collect a set of more informative subgraph features out of the frequent features.

By integrating weight values in Eq.(5.3), we define a weighted similarity matrix $W = [W_{ij}]^{n \times n}$ as follows,

$$W_{ij} = \begin{cases} \frac{w_i + w_j}{A} & : \quad y_i y_j = -1 \\ -\frac{w_i + w_j}{B} & : \quad y_i y_j = 1 \\ \frac{w_i + w_j}{C} & : \quad G_i, G_j \in D_t^U \\ 0 & : \quad otherwise \end{cases} \tag{5.4}$$

Accordingly, Eq. (5.3) can be rewritten as follows,

$$\begin{aligned} \mathcal{J}(\mathbf{g}) &= \frac{1}{2} \sum_{y_i y_j} (-\boldsymbol{x}_j)^2 W_{ij} \\ &= tr(X(T_d - W)X^T) \\ &= tr(X L_d X^T) \\ &= \sum_{g_p \in \mathbf{g}} f_{g_p} L_d f_{g_p}^T \end{aligned} \tag{5.5}$$

In Eq.(5.5), tr denotes the trace of a matrix. $X = [x_1, x_2, \cdots, x_n]$ is the matrix consisting binary feature vectors represented D_t. T_d is diagonal weighted degree matrix of W, *i.e.*, $(T_d)_{ii} = \sum_j W_{ij}$. $L_d = T_d - W$ is known as a Laplacian matrix. f_{g_p} is an indicator vector of subgraph g_p with respect to all graphs G_i in chunk D_t, *i.e.*, $f_{g_p} = [f_{g_p}^{G_1}, f_{g_p}^{G_2}, \cdots, f_{g_p}^{G_n}]$, where $f_{g_p}^{G_i} = 1$ iff $g_p \subseteq G_i$; otherwise $f_{g_p}^{G_i} = 0$.

Definition 7. ***gScore:*** *Given a weighted matrix W as defined in Eq.(5.4) and a graph chunk D_t, the informativeness score of a subgraph g_p is*

$$\mathbb{i}(g_p) = f_{g_p} L_d f_{g_p}^T \tag{5.6}$$

where f_{g_p} is an indicator vector of g_p in D_t, $T_d = diag(d_i)$ is a diagonal matrix with $d_i = \sum_j W_{ij}$, and $L_d = T_d - W$ is a Laplacian matrix.

In order to find the subgraph feature set $\mathbf{g}$ which maximizes the informativeness $\mathcal{I}(\mathbf{g})$ as defined in Eq. (5.5), we can calculate gScore values of all subgraphs in $\mathcal{F}_s$ and sort them, according to their gScore, in a descending order, *i.e.*, $\mathbb{i}(g_1) \geq \mathbb{i}(g_2) \cdots \geq \mathbb{i}(g_{|\mathcal{F}|})$. By using the top-$m$ features $\mathbf{g} = \{g_1, g_2, \cdots, g_m\}$, we can maximize $\mathcal{I}(\mathbf{g})$.

5.3.2 Informative Subgraph Feature Selection

To obtain frequent subgraph set $\mathcal{F}_s$ from D_t, various approaches [14, 58, 79, 151, 154] has been proposed for frequent subgraph mining. In our thesis, we employ a Depth-First-Search (DFS) based algorithm gSpan [151] to enumerate subgraphs. The key idea of gSpan is that each subgraph has a unique DFS Code, which is defined by a lexicographic order of the discovery time during the search process. Two subgraphs are isomorphism iff they have the same minimum DFS Code. By employing a depth first search strategy on the DFS Code tree (where each node is a subgraph), gSpan can effectively enumerate all frequent subgraphs efficiently.

Intuitively, to maximize $\mathcal{I}(\mathbf{g})$ for subgraph feature selection, a simple solution is to use gSpan to discover frequent subgraph set $\mathcal{F}_s$ from each graph chunk D_t, and constantly maintain the feature set $\mathbf{g}$ with the maximum gScore during the frequent subgraph search process. In other words, assume the feature set $\mathbf{g}$ already contains m subgraph features with g_{min} being the subgraph having the minimum gScore in $\mathbf{g}$. For each newly explored subgraph g_p, if g_p's gScore is larger than $\mathbb{i}(g_{min})$, the algorithm will replace g_{min} with g_p to ensure that $\mathcal{I}(\mathbf{g})$ is maximized. The similar approach has, in fact, been employed in a previous work [75].

Algorithm 3 Minimum Redundancy Subgraph Feature Selection

Require:
D_t: A graph data chunk;
min_sup: The threshold of the frequent subgraph;
m: the number of features to be selected;
Ensure:
$\mathbf{g} = \{g_1, g_2, \cdots, g_m\}$: A set of features;
1: $\mathbf{g} = \emptyset$, $\tau = 0$;
2: **while** Recursively visit the DFS Code Tree in gSpan **do**
3: $g_p \leftarrow$ current visited subgraph in DFS Code Tree;
4: **if** $freq(g_p) < min\_sup$ **then**
5: **continue**;
6: Compute the gScore $\mathbb{i}(g_p)$ for subgraph g_p;
7: **if** $|\mathbf{g}| < m$ or $\mathbb{i}(g_p) > \tau$ **then**
8: $\mathbf{g} \leftarrow \mathbf{g} \bigcup g_p$;
9: **if** $|\mathbf{g}| > m$ **then**
10: $g_q \leftarrow Subgraph\_Redundancy\_check(\mathbf{g})$; //Algorithm 4
11: $\mathbf{g} \leftarrow \mathbf{g}/g_q$;
12: $\tau = \min_{g_i \in \mathbf{g}} \mathbb{i}(g_i)$;
13: Depth-first search the subtree rooted from node g_p;
14: **return** $\mathbf{g}$;

Take Fig. 5.3 as an example, because g_{12} has the lowest gScore (the value showing in the circle), it will be replaced by subgraphs with a higher gScore value to maximize $\mathbb{I}(\mathbf{g})$ for feature selection. In fact, there are a number of highly correlated subgraph features, such as g_5, g_6, g_7 and g_{11}, and a better approach is to replace one of the highly correlated subgraph features to ensure high informativeness and low redundancy features to be included in $\mathbf{g}$.

5.3.3 Minimum Redundancy Subgraph Feature Selection

In order to discover the most informative subgraph feature subset $\mathbf{g}$ with minimum redundancy, as defined in Eq.(5.2), we take feature correlations into consideration, and aim to minimize the correlation between features during the subgraph feature exploration process. This objective is achieved through a two-step optimization process as follows,

- **Maximize** $\mathcal{I}(\mathbf{g})$: Explore new informative subgraph g_p whose gScore value is greater than the minimum gScore of the current feature set $\mathbf{g}$.

- **Minimize** $\mathcal{R}(\mathbf{g})$: If some highly correlated subgraphs exist, replace g_p with a subgraph having the highest redundancy; otherwise, replace g_p with the subgraph having the minimum gScore.

Algorithm 3 lists the proposed minimum redundancy subgraph feature selection method. The algorithm starts with an empty feature set $\mathbf{g}$ and the minimum gScore $\tau = 0$, and continuously enumerates subgraphs by recursively visiting the DFS Code Tree in gSpan algorithm. If a subgraph g_p is not a frequent subgraph, both g_p and its subtree will be pruned (line 4-5). Otherwise, we calculate g_p's gScore value $\mathbb{i}(g_p)$. If $\mathbb{i}(g_p)$ is larger than τ or the feature set $\mathbf{g}$ has less than m subgraphs (*i.e.* $\mathbf{g}$ is not full), we add g_p to the feature set $\mathbf{g}$ (lines 7-8). Meanwhile, if the size of $\mathbf{g}$ exceeds the predefined size m, we need to remove one feature g_q which is highly correlated to other features in $\mathbf{g}$ and has less discriminative power (lines 9-11) (the detailed procedures to discover g_q is discussed in Section 5.3.3.1 and Algorithm 4). After that, the algorithm continues the depth-first search by following the children of g_p (line 13), and continues until the frequent subgraph mining process by gSpan is completed.

The above process not only maintains the set of subgraphs with high discriminative power, but also minimizes the feature redundancy. As a result, more useful subgraph features can be used to cover graphs for classification. In the example showing in Fig. 5.3, instead of removing g_{12} which has the lowest gScore value, our method will remove g_6 to minimize the redundancy in the feature set $\mathbf{g}$.

5.3.3.1 Subgraph Redundancy Check

When a new informative subgraph is added into feature set $\mathbf{g} = \{g_1, g_2, \cdots, g_m\}$, we need to remove a subgraph g_q from $\mathbf{g}$ if $|\mathbf{g}| > m$. To minimize the redundancy, the subgraph g_q should satisfy the following two conditions: (1) *high correlation:* g_q should be highly correlated to other subgraphs, and (2) *low informativeness:* g_q should have low discriminative power for classification.

In order to discover a subgraph g_q highly correlated to other graphs, we employ

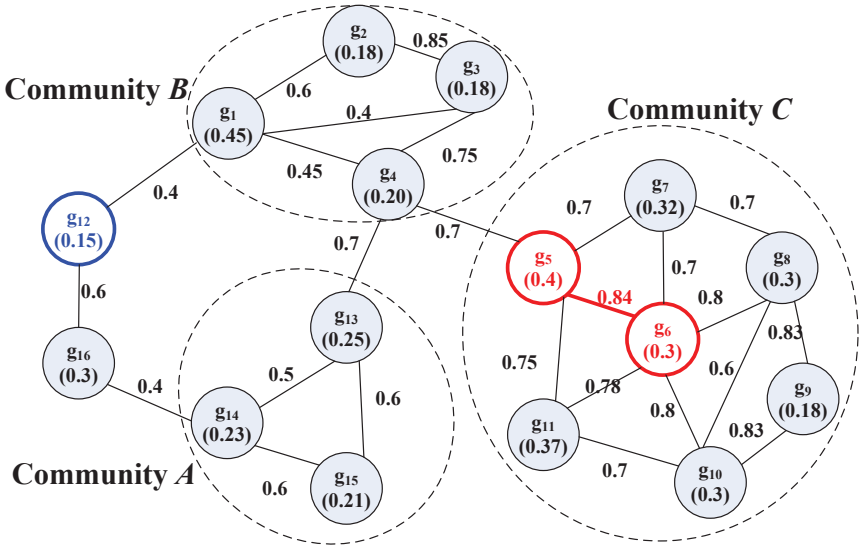


Figure 5.3: An illustrated example for subgraph selection with minimum redundancy. Given a subgraph set $\mathbf{g} = \{g_1, g_2, \cdots, g_m\}$, each node represents a subgraph g_i and its gScore value $\mathbb{i}(g_i)$, and each edge denotes the Pearson's correlation coefficient between two subgraphs. Edges with low value are omitted for clear presentation. Without considering the correlation among features, one will delete the feature g_{12}(blue color) as it has the minimum gScore value $\mathbb{i}(g_{12})$. By considering feature correlations, feature g_6 (red color) will be deleted, because (1) g_6 is located in a dense and highly correlated group, and (2) g_6 has the highest correlation with its neighbor g_5 and its informativeness value is smaller than g_5.

the community detection principle in the social network analysis [99] to discover groups (or communities) from the subgraph features in $\mathbf{g} = \{g_1, g_2, \cdots, g_m\}$. Because each community represents a group of subgraphs sharing high correlations, we can select g_q from these communities to minimize the redundancy of the feature set. Accordingly, our strategy for redundant feature deletion follows two steps:

a) **Subgraph Community Discovery:** Find dense and highly correlated subgraph communities.

b) **Redundant Subgraph Deletion:** Locate the most correlated subgraph pair $< g_i, g_j >$ in the most dense community A, and remove the least informative subgraph from the pair $< g_i, g_j >$.

Subgraph Community Discovery: To discover subgraph community from $\mathbf{g}$, we construct a correlation graph, with each node denoting a subgraph g_i, and each edge between g_i and g_j indicating their Pearson's correlation $\phi(g_i, g_j)$ (defined in Eq. (5.1)). A correlation graph example is shown in Fig. 5.3.

To form communities in the correlation graph, we further construct a k_{nn} *correlation graph matrix* $\mathcal{Q} = [\mathcal{Q}_{ij}]^{m \times m}$. More specifically, $\mathcal{Q}_{ij} = 1$ iff subgraph g_j is among one of the k_{nn}-nearest-neighbors (k_{nn} largest correlated subgraphs) of subgraph g_i or g_i is among one of the k_{nn}-nearest-neighbors of g_j; otherwise $\mathcal{Q}_{ij} = 0$.

Definition 8. ***Clique and Maximum Clique.*** *A graph (subgraph)* $G = (\mathcal{V}, E, \mathcal{L})$ *is a clique if for* $\forall v_i, v_j \in \mathcal{V}$, $< v_i, v_j > \in E$. *A clique is a maximum clique if it is not a subgraph of any other clique.*

In the example showing in Fig. 5.3, group A denotes a maximum clique with size 3 (*i.e.* a 3−clique).

Our community discovery algorithm is based on the maximum clique finding in the correlation graph matrix. More specifically, we first discover the maximum γ-cliques in the correlation graph (we use Bron-Kerbosch algorithm [19] in our experiments), and then overlap these maximum cliques to form communities. For instance, in Fig. 5.3, the community B is jointed by two 3-cliques $g_1 - g_2 - g_3$ and $g_3 - g_1 - g_4$. Using 3-cliques, we can find three communities in Fig. 5.3.

After discovering the subgraph communities, we need to select the most dense and correlated community.

Definition 9. ***Community Redundancy:*** *Given a community* $A = (\mathcal{V}_A, E_A, \mathcal{L}_A)$, *where each vertices in* $\mathcal{V}_A$ *is a graph feature, the Community redundancy (CR) for A is defined as:*

$$CR(A) = \frac{1}{|E_A|} \sum_{\mathcal{Q}_{ij}=1} \phi(g_i, g_j) \tag{5.7}$$

The community redundancy (CR) assesses the average correlations between linked community members. The higher the CR value, the more redundancy

Algorithm 4 *Subgraph_Redundancy_check*($\mathbf{g}$)

Require:
$\mathbf{g} = \{g_1, g_2, \cdots, g_m\}$: A set of subgraph features;
k_{nn}: Number of neighbors;
γ: The minimum clique size;

Ensure:
g_q: A highly correlated and less informative feature in $\mathbf{g}$;

$G \leftarrow$ Construct a correlation graph with nodes as features (i.e., $g_i \in \mathbf{g}$), and edges as Pearson's correlation coefficients between nodes;
$\mathcal{Q} \leftarrow$ Construct an k_{nn} graph matrix;
$\mathcal{C}_{liques} \leftarrow$ find all maximum cliques from $\mathcal{Q}$;
$\mathcal{C}_{om} \leftarrow$ build communities by jointing $\mathcal{C}_{liques}$ with minimum size γ;
if $\mathcal{C}_{om} \neq \emptyset$ **then**
  $Z = \arg\max_{A \in \mathcal{C}_{com}} CR(A)$;
  $< g_i, g_j >= \arg\max_{<g_p, g_q> \in Z} \phi(g_p, g_q)$;
  $g_q = \arg\min(\mathbb{i}(g_i), \mathbb{i}(g_j))$;
else
  $g_q = \arg\min_{g_x \in \mathbf{g}} \mathbb{i}(g_x)$;
return g_q;

exists in the community. Given a feature set $\mathbf{g} = \{g_1, g_2, \cdots, g_m\}$, its redundancy ($\mathcal{R}(\mathbf{g})$) is the maximum CR value among all communities $\mathcal{C}_{om}$ discovered from $\mathbf{g}$,

$$\mathcal{R}(\mathbf{g}) = \max_{A \in \mathcal{C}_{om}} CR(A) \tag{5.8}$$

To minimize the redundancy of $\mathbf{g}$ as defined in Eq.(5.8), we need to delete the subgraph feature with the largest correlation and small informativeness in the most redundant community.

Redundant Subgraph Deletion: After locating the most redundant community, the feature pair (g_i and g_j) with the highest correlation value is identified as the most correlated feature pair in the the community. The redundant subgraph feature is the one (g_i or g_j) which has a smaller gScore value. Then the least informative redundant feature is removed from $\mathbf{g}$. In example showing in Fig. 5.3, community C is the most redundant community (with highest CR value). g_5 and g_6 are the most correlated feature pair in community C. Because $\mathbb{i}(g_6) < \mathbb{i}(g_5)$, g_6 is identified as the least informative redundant feature to be removed from $\mathbf{g}$.

Algorithm 4 lists detailed procedures of Subgraph_Redundancy_Check() for finding the most correlated and the least informative feature in a set of features $\mathbf{g} = \{g_1, g_2, \cdots, g_m\}$. It is worth noting that if the community discovery fails to find any community from the feature graph $\mathcal{Q}$ (*i.e.* $\mathcal{C}_{om} = \emptyset$), it means that subgraph features have low correlations with each other. The algorithm will return the subgraph with the smallest gScore value as the least informative most redundant subgraph feature (line 10).

5.4 gSLU Algorithm

In graph stream settings, graph data may constantly evolve (*i.e.*, concept drifting), which makes existing models incapable of classifying instances representing emerging/changing concepts. Accordingly, we define *disturbing graph samples* by using models trained from historical data as follows,

Definition 10. ***Disturbing Graph Samples:*** *Given a classifier trained from historical graph data, disturbing graph samples (or instances) are the ones which are incorrectly classified by the given classifier.*

Because disturbing graph samples are "difficult" instances and an existing model is incapable of classifying them, they need to be emphasized during the stream learning process. In our system, we use an instance based weighting method to capture difficult graph samples, and combine the instance weight with the feature selection module.

Instance Weighting: The idea of our weighting scheme is as follows: as soon as a new graph chunk $D_t = D_t^l \bigcup D_t^u$ is collected for processing, we use an ensemble of classifiers $E = \{\mathcal{H}_{t-k}, \mathcal{H}_{t-k+1}, \cdots, \mathcal{H}_{t-1}\}$ which is trained from historical chunks to predict labeled graphs in D_t^l. If a graph is miss-classified by E, we increase the graph's weight because it is a difficult sample, and if a graph is correctly classified we decrease its weight. This weighting mechanism is also similar to the Adaboost algorithm [42]. By doing so, we can integrate the instance weight to the succeeding subgraph feature selection procedure, so gSLU can emphasize on

Algorithm 5 gSLU Algorithm

Require:
$\mathbb{S} = \{D_1, D_2, \cdots\}$: Graph Stream
k: The maximum capacity of the ensemble

Initialize $E = \emptyset, t = 0$;
while $\mathbb{S}! = \emptyset$ **do**
  // Training Phase:
  $D_t \leftarrow$ A new graph chunk;
  $D_t = D_t^l \bigcup D_t^u$; $\mathbb{S} \leftarrow \mathbb{S}/D_t$; $t = t + 1$;
  if $(t == 1)$ **then**
    $\mathbf{g} \leftarrow$ minimum redundancy features in D_t; //Algorithm 3
    $H_t \leftarrow$ classifier built from D_t and $\mathbf{g}$;
    $E \leftarrow E \bigcup \mathcal{H}_t$
  else
    $\xi \leftarrow \frac{\sum_{G_i \in D_t^l} (h(G_i|E) \neq y_i)}{|D_t^l|}$;
    $w_i = \begin{cases} w_i \sqrt{\frac{1-\xi}{\xi}} & : \quad h(G_i|E)! = y_i, G_i \in D_t^l \\ w_i \sqrt{\frac{\xi}{1-\xi}} & : \quad h(G_i|E) = y_i, G_i \in D_t^l \end{cases}$;
    $w_i = \frac{w_i}{\sum_{G_i \in D_t^l \bigcup D_t^u} w_i}$;
    $\mathbf{g} \leftarrow$ minimum redundancy features in D_t; //Algorithm 3
    $\mathcal{H}_t \leftarrow$ classifier built from D_t and $\mathbf{g}$;
    $E \leftarrow E \bigcup \mathcal{H}_t$
    if $|E| > k$ **then**
      $E \leftarrow E/\mathcal{H}_{t-k}$
  // Testing Phase:
  $D_{t+1} \leftarrow$ A new graph chunk;
  $h(G_i|E) = \arg\max \sum_{i=t-k-1}^{t} h(G_i|\mathcal{H}_i)$

difficult samples and select a set of informative features to represent disturbing graph samples.

gSLU Algorithm: Algorithm 5 lists detailed procedures of the proposed gSLU framework which combines instance weighting and minimum redundancy subgraph feature selection for graph stream classification.

The "while" loop in Algorithm 5 represents a stream processing cycle which repeats as long as new graph data continuously arrive. Once a new chunk D_t (including labeled D_t^l and unlabeled graphs D_t^u) is collected, gSLU first checks whether D_t is the first chunk (line 5). For the first chunk D_1, gSLU simply retrieve a set of features $\mathbf{g}$ from D_1 by using Algorithm 3, and add the classifier $\mathcal{H}_t$ built from $\mathbf{g}$ to initialize the ensemble E (lines 5-8).

For any succeeding chunks (except the first chunk) $D_t, t = 2, 3, \cdots$, gSLU uses the ensemble E to tune the weight of each graph in D_t by using ensemble error rate ξ (line 10), where $h(G_i|E)$ returns the class label of G_i predicted by the ensemble E. If the ensemble E misclassifies a graph $G_i \in D_t^l$, gSLU increases the weight of G_i by $\sqrt{\frac{1-\xi}{\xi}}$, otherwise gSLU decreases the weight of G_i by $\sqrt{\frac{\xi}{1-\xi}}$ (line 11). On line 12, gSLU normalizes the weight values for all instances in D_t, followed by line 13 which retrieves a set of minimum redundancy subgraph features $\mathbf{g}$ from D_t. By using features in $\mathbf{g}$, a classifier $\mathcal{H}_t$ is trained and is used to update the ensemble E (lines 14-15). If the number of classifiers in E exceeds the predefined size k, we discard the oldest classifier in E (line 17).

At the testing phase, gSLU uses the majority vote of all classifiers in E to predict graphs in the new graph chunk D_{t+1}.

In our algorithm, we used Adaboosting-like algorithm [42] to update the graph weights. As on each chunk, we do not perform subgraph selection iteratively like Adaboost algorithm does, we do not have similar accuracy guarantee as Adaboost. We use this scheme because it can help us to capture the "difficult" graph samples that may be caused by concept drift.

5.5 Experiments

We report our experiments on real-world graph data, with emphasis on (1) effectiveness of the proposed minimum redundancy subgraph feature selection module, and (2) the efficiency and effectiveness of gSLU for graph stream classification.

5.5.1 Experimental Settings

Two graph streams, DBLP and NCI graph streams (please refer to table 3.2 and Section 3.3 for more details), collected from real-world applications are used in our experiments.

- **DBLP Graph Stream:** We used the DBLP-balanced graph stream in tabel 3.2 for graph stream classification. It is worth noting that the DBLP graph stream consists of two classes, *i.e.*, DBDM (database and data mining) or CVPR (computer vision and pattern recognition). As DBDM and CVPR are overlapping in many aspects, such as machine learning and visual information retrieval. The shifting of the research focus makes DBLP stream an ideal benchmark for concept drifting graph stream classification.
- **NCI Graph Stream:** For NCI graph streams, because the original NCI datasets are highly imbalanced, with about 5% positive graphs, so we used the NCI-balanced collection in table 3.2, which shares balanced class distributions of two classes, for graph stream classification. We concatenate nine datasets into a stream with 33,028 graphs in total. In the NCI graph stream, the bioassay task changes from time to time (from one dataset to another), which simulates the concept drifting in the graph stream.

Baseline Methods To evaluate the effectiveness of our minimum redundancy feature selection and instance weighting based graph stream classification framework, we compare the proposed gSLU with both supervised and semi-supervised feature selection methods (all of them are based on the same ensemble framework as gSLU).

- **Information gain based method (IG+Stream).** In each chunk, we first retrieve a set of frequent subgraph features from labeled graphs, and

rank the features according to their Information Gain (IG) [115] value. The top-m subgraphs with the highest IG are used to build the classifier. This framework only uses labeled graph in the stream and does not consider feature redundancy at all.

- **gSemi based method (gSemi+Stream).** In each chunk, we employ gSemi algorithm [75] to select a set of informative features to train a classifier. This framework combines labeled and unlabeled graphs to maximize the total informativeness score for a set of features, without considering feature redundancy.

For fair comparisons, all three algorithms (gSLU, IG+Stream, and gSemi+Stream) use Nearest Neighbor (NN) classifier (trained from features extracted from each chunk by using different methods) to form an ensemble and predict graphs in a future chunk. We used the NN classifier because the NN classifier is the most simple and based classifier for classification. Similar study [75] on feature selection is also based on the NN classifier. For IG+Stream and gSemi+Stream, there is no instance weighting, and only gSLU updates the graph sample weight over stream. The ensemble method is based on a majority voting approach.

Unless specified otherwise, the default parameter settings are as follows: Ensemble size k=15, chunk size $|D_t| = 800$, number of neighbors to build correlated graphs k_{nn}=5, maximum cliques $\gamma = 4$ (for DBLP) and 6 (for NCI), minimum support threshold $min\_supp = 1\%$ (for DBLP) and 30% (for NCI) of the chunk size $|D_t|$.

All experiments are collected from a linux cluster computing node with an Interl(R) Xeon(R) @3.33GHZ CPU and 3GB fixed memory size.

5.5.2 Experimental Results

In this section, we first compare the feature selection component of each algorithm, without considering the ensemble framework and graph weighting. Then we integrate ensemble into each algorithm to compare the overall performance of different methods for graph stream classification.

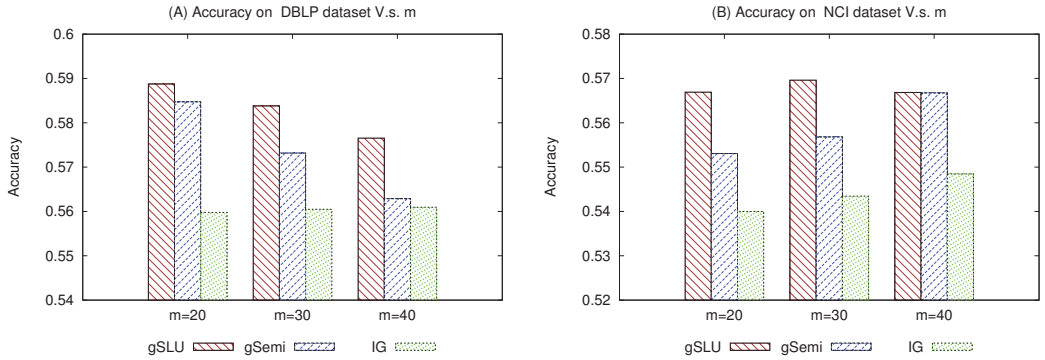


Figure 5.4: Comparison of the proposed minimum redundancy subgraph feature selection with other algorithms. For each graph stream and a chunk D_t, we build a classifier $\mathcal{H}_t$ from chunk D_t, by using features selected from different methods, to predict graphs in D_{t+1}. The results represent the average classification accuracy over all chunks in the graph stream. (A) Results on DBLP stream, with chunk size $|D_t| = 1000$, labeled graphs in each chunk $|D_t^l|$=30; (B) Results on NCI stream, with $|D_t| = 800$, $|D_t^l|$=30.

5.5.2.1 Minimum Redundancy Subgraph Feature Selection Results

To report our minimum redundancy subgraph feature selection results, for each stream, we built a nearest neighbor (NN) classifier $\mathcal{H}_t$ from the current chunk D_t, by using different feature selection methods, to predict graphs in the next chunk D_{t+1}. There is no instance weighting and ensemble framework involved in the experiment. Then we report the average classification accuracy of different methods over the whole stream in Fig. 5.4.

The results in Fig. 5.4 demonstrate that our subgraph feature selection module in gSLU outperforms its peers on both DBLP and NCI streams. Among all three methods, IG only uses labeled graphs to generate feature candidates, and its results are inferior to other methods which confirm that using unlabeled graphs can tackle the data sparsity issues and help generate useful feature patterns for graph classification. Meanwhile, although both gSLU and gSemi combine labeled and unlabeled graphs for feature selection, our results show that combining each feature's informativeness score and the redundancy of the feature set, like gSLU does, is superior to gSemi which only assesses each feature's informativeness score without considering their correlations.

In the following subsections, we report results that incorporate subgraph fea-

ture selection, graph weighting, and ensemble for graph stream classification.

5.5.2.2 Graph Streams Classification Accuracies

Results on different sizes of labeled graphs $|D_t^l|$: In Figs. 5.5 and 5.6, we vary the number of labeled graphs in each graph chunk, and report the results on DBLP and NCI streams.

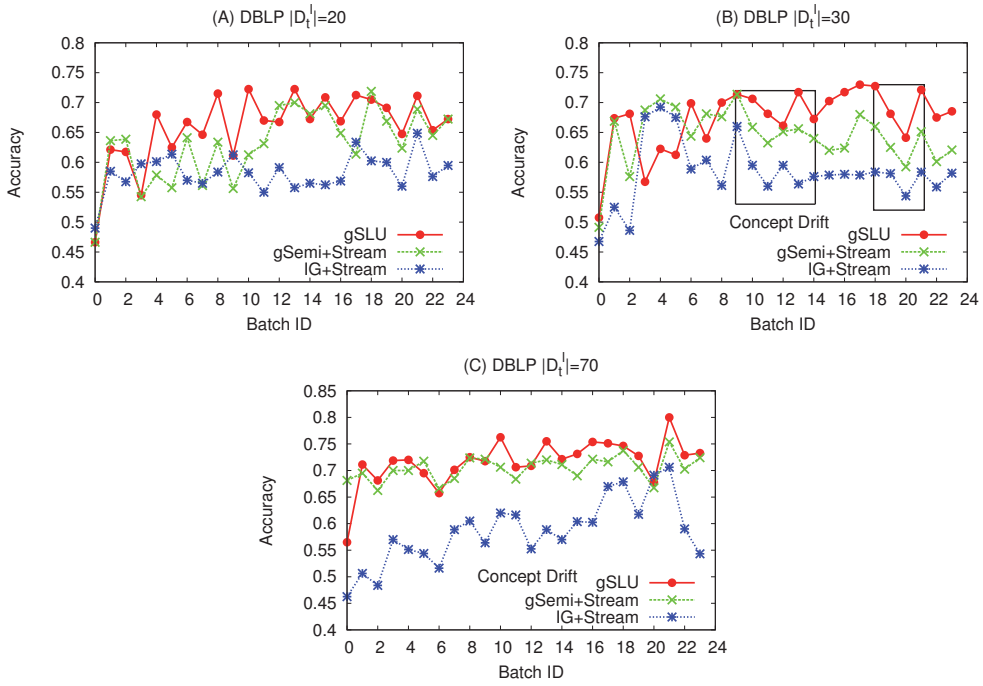


Figure 5.5: Accuracy *w.r.t.* different sizes of labeled graphs on *DBLP stream* with each chunk containing 800 graphs, and the number of features in each chunk is 20. The number of labeled graphs in each chunk: (A) 20; (B) 30; (C) 70.

The results in Figs. 5.5 and 5.6 show that gSLU consistently outperforms IG+stream and gSemi+Stream across the whole stream. Without utilizing unlabeled data, IG+Stream's performance is significantly worse than gSLU and gSemi+Stream, especially in Fig. 5.5.(C). This is because that labeled graphs in each chunk are very limited. By using rich unlabeled graphs, it is possible to discover a set of dense and informative subgraph patterns that better differentiate labeled graphs. Meanwhile, gSLU outperforms gSemi+Stream in data stream

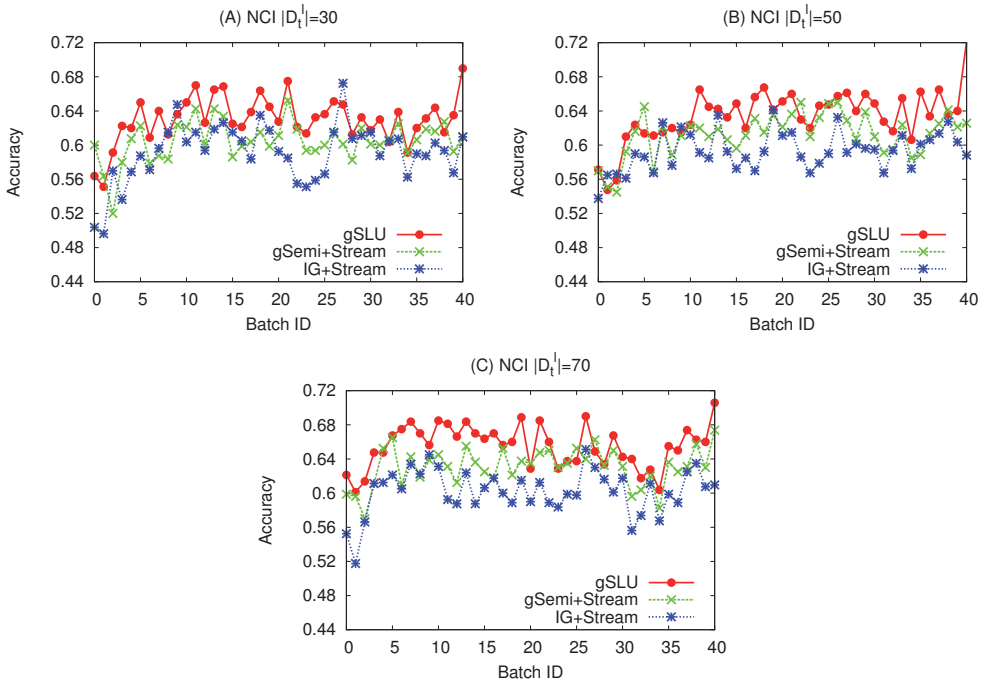


Figure 5.6: Accuracy *w.r.t.* different seizes of labeled graphs on *NCI stream* with each chunk containing 800 graphs, and the number of features in each chunk is 20. The number of labeled graphs in each chunk: (A) 30; (B) 50; (C) 70.

classification. This is mainly attributed to gSLU's two key components, including minimum redundancy subgraph feature selection and instance weighting. The former selects a set of highly informative and low redundancy features to build classifiers, and the latter allows multiple chunks (classifiers) to work in a collaborative way to form an accurate ensemble model. As a result, gSLU achieves good performance in classifying graph streams with dynamic changes. For example, in Fig. 5.5.(B), there are noticeable concept drifting from chunks 10-12 and from 18-20 (marked by the rectangle boxes). As a result, all three methods experience performance loss. By employing instance weighting to tackle the concept drifting, gSLU receives much less loss than gSemi+Stream and IG+Stream.

The average accuracies over the whole graph stream, as shown in Fig. 5.7, demonstrate that increasing the number of labeled graphs in each chunk can benefit all three methods. This result indicates that a large number of labeled graphs in each chunk can enhance the feature selection in a semi-supervised graph

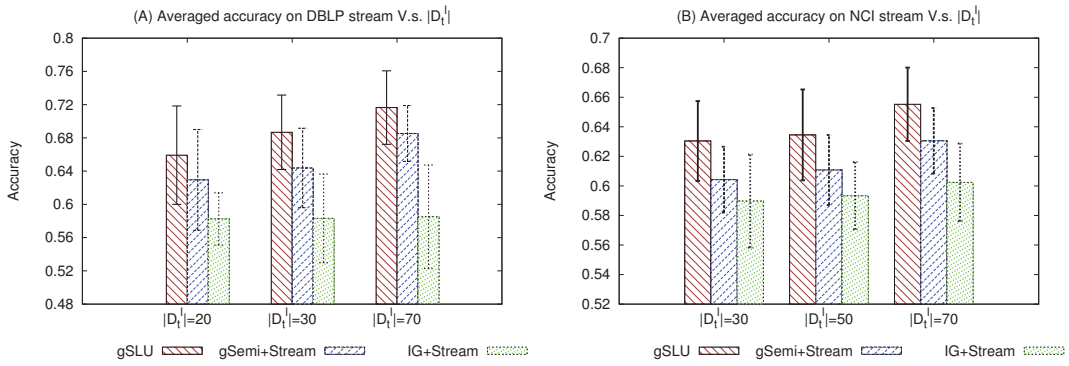


Figure 5.7: Average accuracy (and standard deviation) *v.s.* labeled graph sizes $|D_t^l|$ with chunk size $|D_t|$=800, feature size $m = 20$.

stream setting. Overall, gSLU is the best among the three methods.

In Table 5.1, we report the pairwise t-test (with confident level $\alpha = 0.05$) to validate the statistical significance between three methods. Each entry (value) denotes the p-value for a t-test between two algorithms, and a p-value less than $\alpha = 0.05$ indicating that the difference is statistically significant. From Table 5.1, gSLU statistically outperforms IG+Stream in all cases, and is superior to gSemi+Stream for eight out of nine trials.

Table 5.1: Pairwise t-test results with labeled graph sizes $|D_t^l|$. A, B, and C denote gSLU, gSemi+Stream, and IG+Stream, respectively.

DBLP				NCI			
D_t^l	A-B	A-C	B-C	D_t^l	A-B	A-C	B-C
20	1.7E-03	3.0E-07	9.0E-04	30	3.1E-09	1.4E-11	1.3E-02
30	2.4E-02	7.0E-06	3.7E-09	50	5.7E-07	4.2E-11	1.2E-06
70	**8.6E-02**	1.7E-12	3.2E-10	70	4.2E-09	5.3E-19	1.1E-10

Results on different number of features m: In Figs. 5.8 and 5.9, we report the algorithm performance with respect to different number of subgraph features in each chunk. The average results over the whole stream are also reported in Fig. 5.10. As expected, gSLU has the best performance among the three algorithms for both DBLP (Fig. 5.8) and NCI (Fig. 5.9) streams. In Fig. 5.8.(A), there is a significant concept drifting from chunks 2-3, where gSemi+Stream and IG+Stream both experience sharp accuracy drop. In contrast, gSLU's perfor-

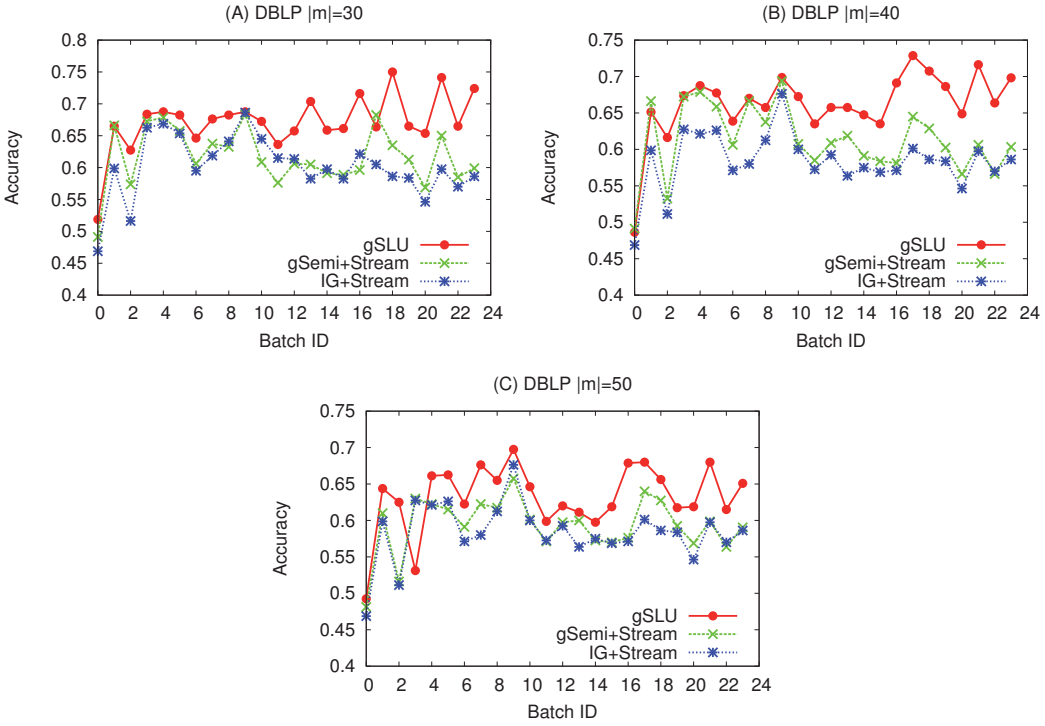


Figure 5.8: Accuracy *w.r.t.* different number of features on *DBLP stream* with each chunk containing 800 graphs, and the size of labeled data in each chunk is 30. The number of feature selected in each chunk: (A) 30; (B) 40; (C) 50.

mance loss is much smaller than other two methods. This also demonstrates the effectiveness of the instance weighting scheme for capturing emerging disturbing graph samples. By adjusting graph weight values, gSLU can immediately select subgraph features to represent emerging samples and force the whole classification framework to adapt to the changing concepts in the graph stream.

Interestingly, results in Fig. 5.10 show that increasing the number of features in DBLP stream actually reduces the accuracy of all three algorithms. This may suggest that for DBLP classification task, a small number of features (such as keywords or simple citation relationships) may have enough discriminate power for classification. Increasing subgraph features may introduce redundant features into the learning process, which deteriorates the accuracy of the classifier. This is, however, not the case for NCI stream where increasing the number of features consistently help improve the classifier.

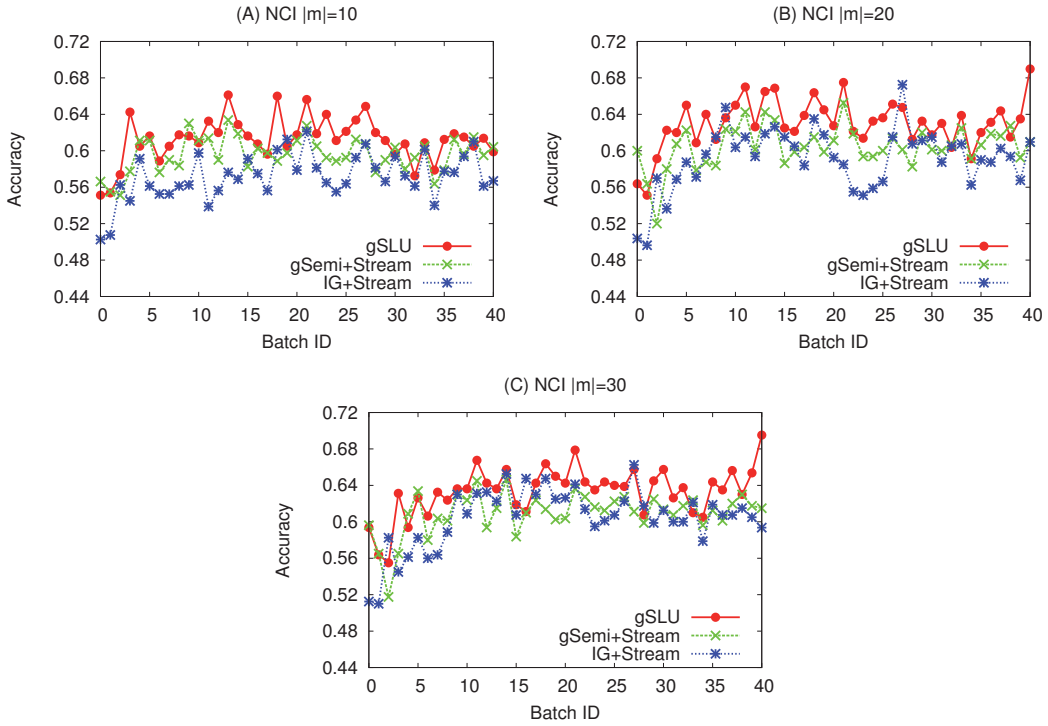


Figure 5.9: Accuracy *w.r.t.* different number of features on *NCI stream* with each chunk containing 800 graphs, and the size of labeled data in each chunk is 30. The number of features selected in each chunk: (A) 10; (B) 20; (C) 30.

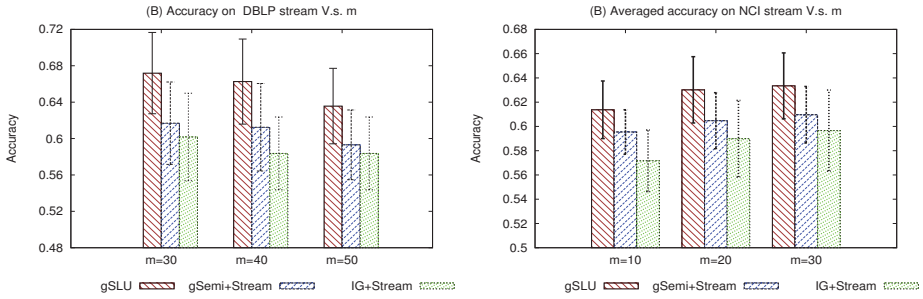


Figure 5.10: Averaged accuracy (and standard deviation) *v.s.* number of features m, with chunk size $|D_t|$=800, feature size $m = 20$.

Table 5.2: Pairwise t-test results with various feature size m. A, B, and C denote gSLU, gSemi+Stream, and IG+Stream, respectively.

DBLP				NCI			
m	A-B	A-C	B-C	m	A-B	A-C	B-C
30	5.4E-07	6.1E-08	2.1E-02	10	7.4E-06	1.7E-13	1.7E-08
40	1.9E-06	1.2E-11	1.1E-06	20	3.1E-09	1.4E-11	1.3E-02
50	5.3E-05	6.9E-06	9.7E-03	30	4.9E-09	5.2E-08	**2.7E-01**

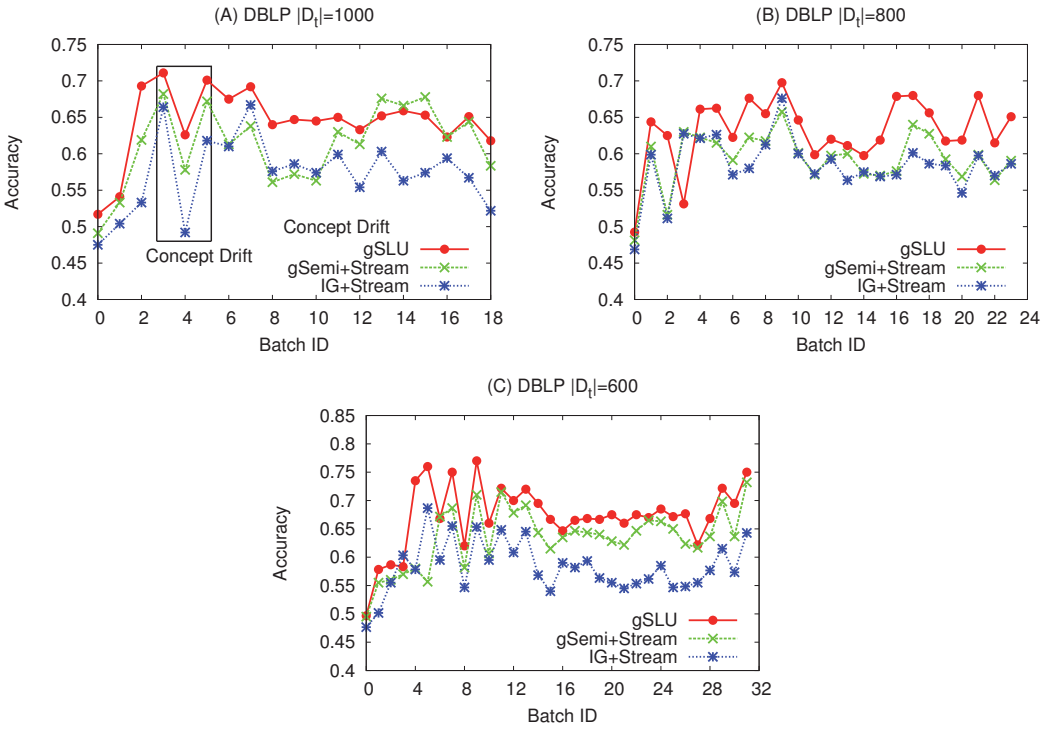


Figure 5.11: Accuracy *w.r.t.* different chunk sizes on *DBLP stream* with each chunk containing 30 labeled graphs, and the number of features in each chunk is 50. The batch sizes vary as: (A) 1000; (B) 800; (C) 600.

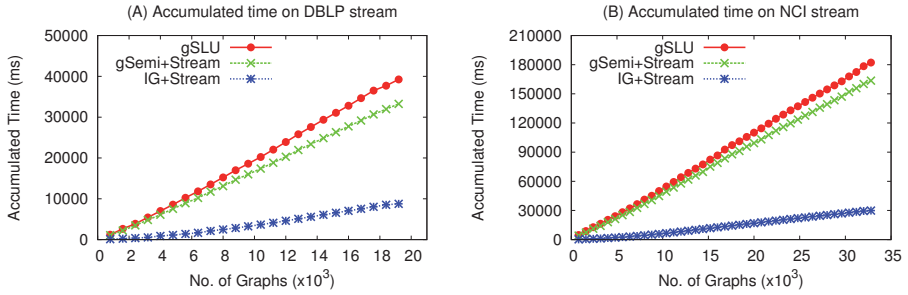


Figure 5.12: System accumulated runtime *v.s.* number of graphs processed over stream. ($|D_t| = 800$, $|D_t^l| = 10\%|D_t|$, and m=20).

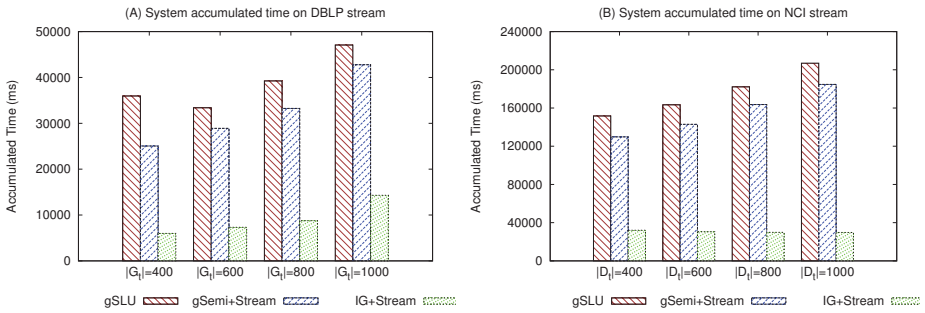


Figure 5.13: System accumulated runtime *v.s.* different chunk sizes $|D_t|$, $|D_t^l| = 10\%|D_t|$. m=20; (A) Results on DBLP stream; (B) Results on NCI stream.

In Table 5.2, we report the pairwise t-test with confident level $\alpha = 0.05$. The p-values (less than 0.05) in each entry assert that gSLU statistically significantly outperforms gSemi+Stream and IG+Stream.

Results on different chunk sizes $|D_t|$: In Fig. 5.11, we report the algorithm performance by using different number of graphs in each chunk $|D_t|$ (varying from 1000, 800, to 600).

Overall, the results in Fig. 5.11 show that the accuracies of all methods fluctuate to a large extent across the whole stream. Because we fixed the number of labeled graphs in each chunk to 30, the overall trend shows that a smaller chunk size may slightly outperform the settings with a larger chunk size.

5.5.2.3 Graph Streams Classification Efficiency

In Figs. 5.12 and 5.13, we report the system runtime performance (efficiency) for graph stream processing. The results show that IG+Stream algorithm takes much less time than semi-supervised algorithms (gSLU and gSemi+Stream). This is mainly because IG+Stream carries out frequent subgraph mining on a small set of labeled graphs whereas both gSLU and gSemi+Stream need to handle labeled and unlabeled data. In our experimental settings, the labeled graphs is less than 10% of the unlabeled graphs, so IG+Stream shows much better runtime performance. When comparing gSLU and gSemi+Stream, gSLU requires slightly more time than gSemi+Stream because of the minimum-redundance checking during the feature selection process. Meanwhile, the accumulated system runtime *w.r.t.* different chunk sizes, as shown in Fig. 5.13, also indicate that system runtime remains relatively stable for various chunk sizes. Overall, the results show that gSLU linearly scales to the number of graphs and chunks, which means that gSLU is capable of handling real-world high speed graph streams.

In the experiments, we also tried to collect system runtime by treating each stream as one single chunk. Unfortunately, both streams ended up with insufficient memory for processing (using a 3GB Memory cluster computing node). This also asserts that a stream based processing framework is potentially useful for handling large scale graph datasets.

5.6 Conclusion

In this chapter, we investigated graph stream classification using limited labeled graphs and abundant of unlabeled graphs. We argued that existing methods for subgraph feature selection fail to consider redundancy between selected subgraphs, and dynamic graph streams require solutions to capture emerging concept drifting examples so the overall framework can adapt to the changes in the streams. In the chapter, we proposed unique measures to discover informative subgraph features with minimum redundancy, through which we can build classifiers from graph streams. To capture emerging difficult graphs in the stream, we proposed an instance weighting mechanism to force the subgraph feature selec-

tion module to emphasize on emerging concept drifting graphs, so our model can quickly adapt to the changes in the stream. Experiments validated the proposed design for effective graph stream classification.

Chapter 6

Imbalanced and Noisy Graph Stream Classification

Although graph classification has been advanced greatly in recent years, no studies have paid attention to imbalanced graph classification in streaming scenarios. In this chapter, we will investigate the problem of graph stream classification with imbalanced class distributions and noise.

6.1 Introduction

6.1.1 Imbalanced Graph Classification

Existing graph classification algorithms [29, 40, 70, 75, 120, 144, 168] mainly focus on graph applications with balanced class distributions (*i.e.* the percentages of samples in different classes are close to each other). In reality, balanced class distribution is rarely the case and for many applications interesting samples only form a small percentage of the whole population. For instance, in NCI chemical compound graph datasets, only about 5% percent of molecules are active to the anti-cancer bioassay test, and the remaining 95% are inactive to the test (see Table 3.2). Learning from datasets with imbalanced class distributions has been

widely studied in past years. Popular techniques include sampling [85], ensemble learning [45, 80], and SVM adapting [6, 135], and a recent monograph has discussed many methods for imbalanced data classification [56]. Unfortunately, these learning methods for imbalanced data are designed and evaluated only for data with vector representations, without considering complex structure information of graphs. As a result they may have sub-optimal performance when applied to graph data.

When dealing with imbalanced graph data, a simple solution is to apply existing methods for imbalanced data [85] to under-sample graphs in the majority class to obtain a relatively balanced graph dataset, and then apply graph classification methods [75, 117]. Such a trivial treatment not only ignores the structure information in the graph datasets, but may be also subject to the risk of losing valuable information in the sampled data, and results in poor algorithm performance. This problem will be further aggravated with the presence of noise (*i.e.* mislabelled samples). For graph applications, it is an inherent complex process to examine and label structured data, which may result in mislabelled samples (or noise). Because noise accounts for a small portion of the whole dataset, they are similar to instances in the minority class. As a result, solutions which try to emphasize on minority class samples to improve the performance gain may falsely emphasize on noise and incur significant performance loss instead.

6.1.2 Graph Stream Classification

The second challenge arisen in real-life applications is the dynamic increase and change of structural information over time, *i.e.*, graph streams [2, 101, 107, 108, 110]. For example, an online user's browsing pattern, with respect to all web pages, can be regarded as a graph. The browsing patterns of all users will form a graph stream. Each scientific publication and its references can be represented as a graph [101], so all scientific papers, collected in chronological order, will form a graph stream with increasing volumes.

In stream scenarios, classifying noisy and imbalanced graphs is a very challenging task. This is because the decision concepts (decision boundaries) of the graph data may gradually (or rapidly) change, *i.e.* the concept drifting in the

stream. In order to tackle the concept drifting, the subgraph feature selection and classification modules should take the dynamic graph stream as a whole to achieve maximum benefits. Unfortunately, existing graph classification methods all work on static datasets with balanced class distributions. No effective strategy exists to support classification for imbalanced graph streams. Intuitively, a trivial solution is to partition graph stream into a number of stationary subsets (or chunks) and carry out classifier learning in each individual subset. This simple solution, however, does not allow graph subsets to collaborate with each other to train robust models. More effective solution to capture dynamic changes in graph stream is highly desired.

In summary, when classifying noisy and imbalanced graph streams, major challenges exist for subgraph feature selection, noise handling, and concept drift modeling. More specifically,

- **Bias of learning models:** Low presence of minority (positive) graphs will make learning models biased to the majority class and result in inferior performance on the minority class. In extreme cases (*e.g.*, the minority samples are extremely rare), the classifier may ignore minority samples and classify all graphs as negative.

- **Impact of noise:** Most learning algorithms (such as boosting) are sensitive to noise, because in their designs if an instance's predicted label is different from its original label, the instance will receive a larger weight and plays a more important role in the learning process. As a result, the decision boundaries of the classifiers may be misled by noise and eventually result in deteriorated accuracy.

- **Concept drifting:**. In graph streams, the data volumes and the decision boundaries of the graph data are constantly changing, which impose difficulty for finding effective subgraph features to capture the concept drifting and train classifiers with high accuracy.

To solve the above challenges, we propose, in this chapter, an ensemble boosting algorithm, **gEboost**, for imbalanced graph stream classification. Our theme is to employ a divide-and-conquer approach to partition graph stream into chunks.

In each chunk, we formulate the learning task as a margin maximizing problem, and employ a linear programming boosting algorithm to integrate subgraph feature selection and classifier learning process as a unified framework. To capture graphs represented by drifting concepts in graph streams, we employ a dynamic weighting mechanism, where graphs misclassified by the existing model will receive a higher weight so the boosting procedure (including subgraph feature selection and model learning) can emphasize on difficult graphs for learning. In summary, the key contributions of the chapter is threefold:

1. To the best of our knowledge, gEboost is the first algorithm with capability to handle graph streams with both imbalanced class distribution and noise.

2. While existing graph classification methods consider subgraph feature selection and model learning as two separated procedures, we provide an effective design to integrate subgraph mining (feature selection) and model learning (margin maximization) into a unified framework, so two procedures can mutually benefit each other to achieve a maximization goal.

3. We propose an effective weighting strategy to model dynamic changes of concept drifting graph stream. Our approach, which tunes the weights of misclassified graphs to support graph stream classification, can be easily generalized to stream data with rich structure information.

The remainder of the chapter is structured as follows. Our overall framework are discussed in Section 6.2. Section 6.3 reports the proposed algorithm for learning from noisy and imbalanced graph data. Our gEBoost algorithm is detailed in Section 6.4. Experimental results are presented in Section 6.5, and we conclude the chapter in Section 6.6.

6.2 Overall Framework

In this chapter, we propose an ensemble classification framework, with a linear boosting procedure in each chunk to select discriminative subgraph features and train ensemble based classifiers. The framework, as shown in Fig. 6.1, contains

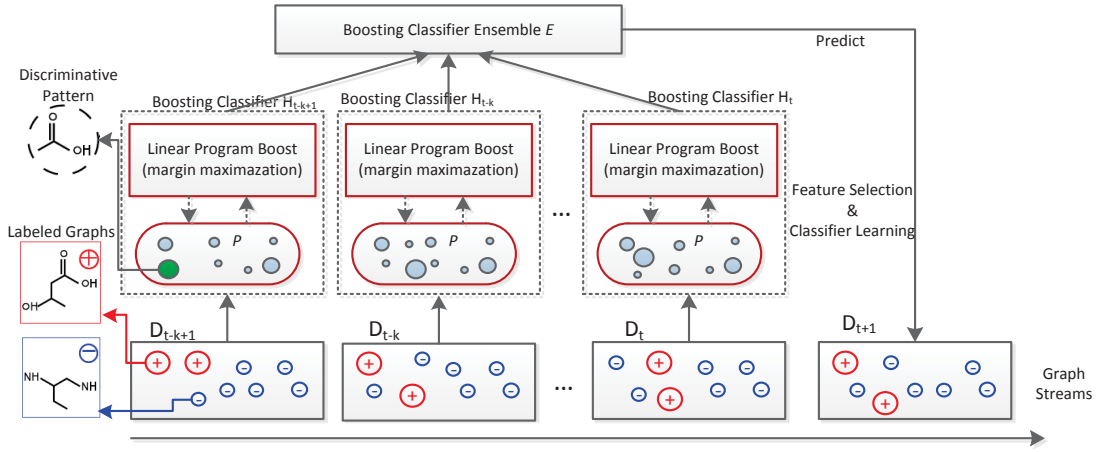


Figure 6.1: A framework for imbalanced noisy graph stream classification. The graph stream is divided into chunks. In each chunk D_t, circles with '+' represent positive graphs, and circles with '-' are negative graphs. The size of a circle represents sample weight in each chunk. The weight of a positive graph is initialized as β times larger than negatives, and it will be fine tuned by the concept drifting weights. In each chunk, we combine the discriminative subgraph feature selection and classifier learning (margin maximization) into a unified framework. This process will return an optimal classifier from each chunk after the linear boosting algorithm is converged (Detailed in Fig. (6.2) and Section 6.3). A classifier ensemble E is built from the most recent k chunks to predict graphs in a yet-to-come chunk D_{t+1}.

three key components: (1) partitioning graph stream into chunks; (2) selecting discriminative subgraph features iteratively and learning a classification model in each chunk; and (3) forming an ensemble model by combining classifiers trained from individual chunks. As soon as a graph chunk D_t is collected, the overall framework proceeds as follows:

- **Instance Weighting for Data Imbalance and Concept Drifting:** To address data imbalance and concept changes in the graph stream, we propose to adjust weight values of graphs in each chunk and use models trained from the graph chunks to pinpoint "difficult" samples in stream. To tackle data imbalance, the initial weight value of each positive graph in the most recent chunk D_t is much larger than negative graphs. Meanwhile, to handle concept drifting, each graph G_i's weight is adaptively updated to accommodate changes in the stream.

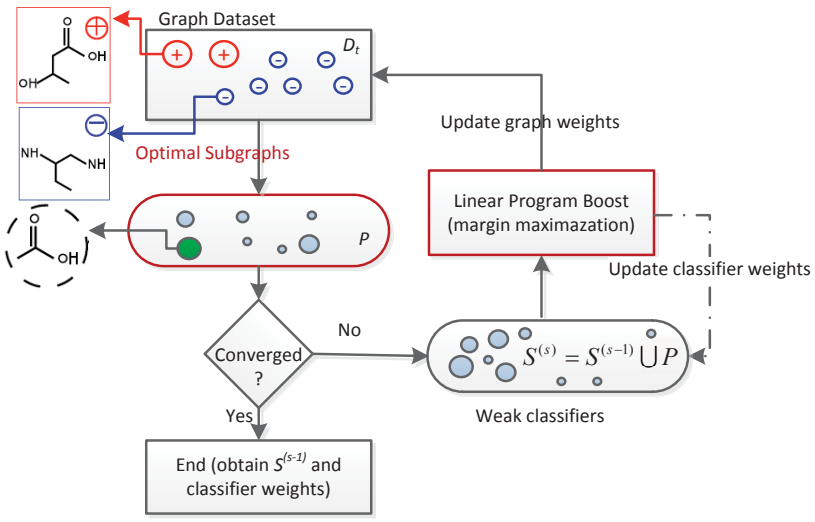


Figure 6.2: The proposed boosting framework for learning from noisy and imbalanced graphs in each chunk. The initial weight of each positive graph in D_t is β times larger than a negative graph (the circle size corresponds to graph weight), and the weight will be further adjusted to capture "difficult" samples (detailed in Section 6.4). In each chunk, our algorithm iteratively selects optimal subgraph features P from D_t, and adds P into a global set S. Afterwards, the algorithm solves a linear programming problem to get two sets of weights: (1) weights for training graphs D_t, and (2) weights for weak classifiers (subgraph decision stumps). The loop between feature selection and margin maximization continues until boosting converges.

- **Subgraph Feature Selection and Classifier Learning:** For graph classification, the weighted graphs in each chunk D_t will help iteratively extract a set of discriminative subgraph features to learn a boosting classifier $\mathcal{H}_t$. The iterative subgraph feature selection and model learning process can mutually benefit each other to achieve maximum performance gain.

- **Updating Ensemble:** The newly learned classifier $\mathcal{H}_t$ from chunk D_t is included into the ensemble to predict graphs in a future chunk D_{t+1}.

In the following sections, we first propose our boosting algorithm for imbalanced and noisy graph classification in a local chunk, and then propose solutions to handle graph stream in Section 6.4.

6.3 Learning from a Local Chunk with Noisy and Imbalanced Graphs

Given a graph chunk $D_t = \{(G_1, y_1), \cdots, (G_n, y_n)\}$, which contains a number of graphs, let $\mathcal{F} = \{g_1, \cdots, g_m\}$ denote the full set of subgraphs in D_t. We can use $\mathcal{F}$ as features to represent each graph G_i into a vector space as $x_i = \{x_i^{g_1}, \cdots, x_i^{g_m}\}$, where $x_i^{g_i} = 1$ if $g_i \in G_i$, and otherwise 0. An example is shown in Fig. 3.1, where we can use three subgraph g_1, g_2, and g_3 to represent graph G_2 as $\boldsymbol{x}_2 = [1, 1, 0]$.

To build weak classifiers for boosting, we can use each subgraph g_j as a decision stump classifier (g_j, π_j), as follows:

$$\hbar(G_i; g_j, \pi_j) = \begin{cases} \pi_j & : \quad g_j \in G_i \\ -\pi_j & : \quad g_j \notin G_i \end{cases} \tag{6.1}$$

where $\pi_j \in \mathcal{Y} = \{-1, +1\}$ is a parameter controlling the label of the classifier. We use decision stumps because they are commonly used in boosting classification of graph data [120]. In addition, (1) it is easy to cast the stumps into a linear program framework, and (2) it can help facilitate the derivation of pruning bounds for subgraph enumeration.

The prediction rule in a local chunk D_t for a graph G_i is a linear combination of all weak classifiers:

$$\mathcal{H}_t(G_i) = \sum_{(g_j, \pi_j) \in \mathcal{F} \times \mathcal{Y}} w_j \hbar(G_i; g_j, \pi_j) \tag{6.2}$$

where w_j is the weight of weak classifier $\hbar(G_i; g_j, \pi_j)$. If $\mathcal{H}_t(G_i) \geq 0$, it is a positive graph (+1), or negative (-1) otherwise.

6.3.1 Framework of Linear Boosting Algorithm

Our linear boosting algorithm for noisy and imbalanced graphs is shown in Fig. 6.2. The framework combines subgraph feature selection and graph classification

into a boosting process as follows:

- **Subgraph Feature Selection:** Given a chunk D_t, with each graph in the chunk being carefully weighted, we need to select a set of subgraph features P to help learn the graph classification models. It will be detailed in Algorithm 7.
- **Margin Maximization:** Based on selected subgraph patterns $S = S \bigcup P$, we learn a classifier by maximizing margins between positive and negative examples. The margin maximization can be formulated as a mathematical problem.
- **Weight Updating for Weak Classifiers and Training Graphs:** By solving margin maximization problem, we can obtain two set of weights: (1) weights for weak classifiers $\boldsymbol{w} = \{w_1, \cdots, w_{|S|}\}$, and (2) weights for training graphs $\boldsymbol{\mu} = \{\mu_1, \cdots, \mu_n\}$.

The above boosting process will continue until the algorithm converges. In the following subsections, we first show how to formulate boosting learning as a mathematical maximization problem, and then combine subgraph selection and model learning (margin maximization) into one framework.

6.3.2 gBoost Algorithm for Balanced graph classification

Our algorithm is extended from the classic gBoost algorithm [120], which is designed for graph classification with balanced graph classification. For gBoost, its learning objective function is

$$\begin{array}{ll} \max\limits_{\rho,\boldsymbol{w},\boldsymbol{\xi}} & \rho - \frac{1}{vn}\sum_{i=1}^{n}\xi_i \\ s.t. & y_i\sum_{k=1}^{m}\hbar_{g_k}(G_i)w_k + \xi_i \geq \rho; \\ & \sum_{k=1}^{m} w_k = 1; \\ & w_k \geq 0, \xi_i \geq 0; \end{array} \tag{6.3}$$

To handle imbalanced and noisy data, we assigned different weights for different classes, as discussed in next subsection.

6.3.3 Objective Function for Imbalanced and Noisy Data

Our boosting algorithm, which considers noisy and imbalanced graph stream, is formulated as the following linear programming optimization problem:

$$\begin{array}{ll} \max\limits_{\boldsymbol{w},\rho,\boldsymbol{\xi}\in\Re_+^N} & \rho - C(\beta \sum_{\{i|y_i=+1\}}^{n^+} \delta_i\varphi_i\xi_i + \sum_{\{i|y_i=-1\}}^{n^-} \delta_i\varphi_i\xi_i) \\ s.\ t. & y_i \sum_{j=1}^{m} w_j \cdot \hbar(G_i; g_j, \pi_j) + \xi_i \geq \rho,\ \ i = 1, 2 \cdots n \\ & \sum_{j=1}^{m} w_j = 1, \quad w_j \geq 0 \quad j = 1, 2 \cdots m \end{array} \tag{6.4}$$

The above objective function aims to maximize the margin ρ between positive and negative graphs. The first set of constraints enforce that both positive and negative graphs are beyond the margin. A misclassified graph G_i (*i.e.*, inside the margin) will be penalized by ξ_i. Here, n^+ and n^- denote the number of graphs in positive and negative classes ($n = n^+ + n^-$) in chunk D_t. $C = \frac{1}{vn}$ is a parameter controlling the magnitude of misclassification in the algorithm. The idea of margin maximization is similar to gBoost [120] and SVM formulation. To handle graph streams with imbalanced distributions and noise, we incorporate three key components: δ_i, β, and φ_i in the objective function. δ_i indicates the weight factor for handling *disturbing graph samples* (Definition 10) in a data stream setting (In this section, δ_i is set as a fixed value, and Section V will show that δ_i can be dynamically adjusted in graph stream). The other two key components in our objective function include:

- **Weight values of graphs in different classes:** In each imbalanced graph chunk, positive graphs are much fewer than negative graphs. So positive graphs should carry larger misclassification penalties to prevent them from being overlooked for learning. Inspired by imbalanced SVM formulation [9, 135] for vector data, in our formulation, the weights of positive graphs are β times higher than the weights of negative graphs. The weight adjustment, with respect to the class distributions, can help alleviate the class imbalance and prevent learning models from being biased towards the majority class (which dominates the graph chunk).

- **Weight values for graphs within the same class:** To handle noise, we introduce a membership value φ_i, for each graph G_i, to indicate how likely

G_i is a noisy graph. By using φ_i to adjust the weight of each graph, we can reduce the impact of noisy graphs on the learning process.

To calculate φ_i in Eq.(6.4), we use the density of each graph G_i to determine its likelihood score of being an noisy graph. Intuitively, if G_i is located far away from its class center, it is more likely being mislabeled (so φ_i will have a smaller value). Therefore, our approach to calculate φ_i is given as follows:

$$\varphi_i = \frac{2}{1 + e^{\tau d(G_i)}}; \tag{6.5}$$

In Eq.(6.5), $d(G_i)$ denotes the distance of graph G_i to its class center in the vector space, and $\tau \in [0, 1]$ is a decay factor controlling the magnitude of the change of the distance.

6.3.4 Linear Boosting with Graph Data

The objective function in Eq.(6.4) requires a feature set $\mathcal{F} = \{g_1, \cdots, g_m\}$ being used to represent graph data for learning and classification. In reality, this feature set is unavailable unless all possible structures of graphs in D_t are enumerated. Enumeration of subgraph patterns is NP-complete. Therefore, Eq.(6.4) cannot be solved directly. *Column generation (CG)* [97], a classic optimization technique, provides an alternative solution to solve this problem. Instead of directly solving the primal problem in Eq.(6.4), CG works on the dual problem by starting from an empty set of constraints, and iteratively selects the most violated constraints until no more violated constraint exists. The final optimal solution, under the iteratively selected constraints, is equal to the optimal solution under all constraints.

We can write the dual problem of Eq.(6.4) as follows [1]:

$$\begin{aligned} &\min_{\boldsymbol{\mu},\gamma} \gamma \\ &s.\ t. \quad \sum_{i=1}^{n} y_i \mu_i \cdot \hbar(G_i; g_j, \pi_j) \leq \gamma, \quad j = 1, 2 \cdots m \\ &\quad 0 \leq \mu_i \leq \beta \delta_i \varphi_i C \qquad if \quad y_i = +1 \\ &\quad 0 \leq \mu_i \leq \delta_i \varphi_i C \qquad if \quad y_i = -1 \\ &\quad \sum_{i=1}^{n} \mu_i = 1. \end{aligned} \tag{6.6}$$

According to the duality theory [13], Eq.(6.4) and Eq.(6.6) have the same solution (objective values) though they have different predicted variables ($\boldsymbol{w}, \rho, \boldsymbol{\xi}$ in Eq.(6.4) and $\boldsymbol{\mu}, \gamma$ in Eq.(6.6)).

For the dual problem, each constraint $\sum_{i=1}^{n} y_i \mu_i \cdot \hbar(G_i; g_j, \pi_j) \leq \gamma$ in Eq.(6.6) enforces restriction on a subgraph pattern (g_j, π_j) over all graphs in D_t. In other words, the m constraints are equivalent to the total subgraphs in D_t, which is practically very large or even infinite. In the primal problem defined in Eq.(6.4), there are only n constraints (which are equal to the number of training graphs in D_t). As a result, we have $m >> n$. To solve the problem (Eq.6.6) in an effective way, we can combine subgraph mining and CG techniques as follows: (1) first discover the top-l subgraph pattern that violates the constraints most in each iteration; and (2) solve the sub-problem based on the selected top-l constraints. After solving Eq.(6.6) based on selected constraints, we can obtain $\boldsymbol{\mu} = \{\mu_1, \cdots, \mu_n\}$, which can be regarded as the new weights for training graphs, so that we can iteratively perform subgraph feature selection in the next round (See Fig.6.2). Such a top-l constraint technique is known as *Multiple Prices* [88] in column generation.

To apply multiple prices, we first define the discriminative score for each subgraph based on the constraints in Eq.(6.6).

Definition 11. ***Discriminative Score:*** *for a subgraph decision stump* (g_j, π_j),

[1] The derivation from Eq.(6.4) to Eq.(6.6) is illustrated in Appendix A.1.

its discriminative score is defined as:

$$\dot{\mathbb{1}}(g_j, \pi_j) = \sum_{i=1}^{n} y_i \mu_i \cdot \hbar(G_i; g_j, \pi_j) \tag{6.7}$$

We can sort subgraph patterns according to their discriminative scores in a descending order, and select the top-l subgraphs to form the most violated constraints.

Suppose $\mathcal{S}^{(s)}$ is the set of decision stumps (subgraphs) discovered by column generation so far at s^{th} iteration. Let $\gamma^{(s)}$ and $\boldsymbol{\mu}^{(s)} = \{\mu_1^{(s)}, \cdots, \mu_n^{(s)}\}$ be the optimal solution for the s^{th} iteration, our algorithm will try to solve linear problem in s^{th} iteration as follows:

$$\begin{aligned} & \min_{\gamma^{(s)}, \boldsymbol{\mu}^{(s)}} \ \gamma^{(s)} \\ s.\ t. \quad & \textstyle\sum_{i=1}^{n} y_i \mu_i^{(t)} \hbar(G_i; g_j, \pi_j) \leq \gamma^{(t)}, \forall (g_j, \pi_j) \in \mathcal{S}^{(s)} \\ & 0 \leq \mu_i^{(s)} \leq \beta \delta_i \varphi_i C \qquad if \quad y_i = +1 \\ & 0 \leq \mu_i^{(s)} \leq \delta_i \varphi_i C \qquad if \quad y_i = -1 \\ & \textstyle\sum_{i=1}^{n} \mu_i^{(s)} = 1. \end{aligned} \tag{6.8}$$

The solutions to Eq.(6.8) and its Lagrange multipliers will result in $\boldsymbol{\mu}^{(s)}$ and $\boldsymbol{w}^{(s)}$ which correspond to (1) new weights for graphs ($\boldsymbol{\mu}^{(s)}$), and (2) new weights for decision stump classifiers ($\boldsymbol{w}^{(s)}$). By using updated weight values, the algorithm will continue and proceed to the $s + 1^{th}$ iteration.

Note that in Eq.(6.8), φ_i changes in each iteration, because the class centers for positive and negative graphs are calculated by using current selected subgraphs $S^{(s)}$ (transfer each graph as a vector based on $S^{(s)}$). The changing subgraph features will result in updated class centers, and result in new φ_i value according to Eq.(6.5).

Our graph boosting framework is illustrated in Algorithm 6. To handle class imbalance, the weight of each positive graph μ_i is set to be β times larger than the weights of negative graphs (step 1). The weight value is further updated by δ_i (step 2), which takes the concept drifting in streams into consideration (detailed in Section 6.4). After that, the boosting algorithm iteratively selects

Algorithm 6 Boosting for Noisy and Imbalanced Graph Classification in a Local Chunk

Require:

$D_t = \{(G_1, y_1), \cdots, (G_n, y_n)\}$: Graph Datasets

Ensure:

$\mathcal{H}_t(G_i) = \sum_{(g_j,\pi_j)\in S^{(s-1)}} w_j^{(s-1)} \hbar(G_i; g_j, \pi_j)$: Classifier;

$\delta_i, i = 1, \cdots, n$: **Concept drifting weights**;

$\mu_i = \begin{cases} \varsigma^+ & : \quad y_i = +1 \\ \varsigma^- & : \quad y_i = -1 \end{cases}$, where $\frac{\varsigma^+}{\varsigma^-} = \beta$, $\sum_{i=1}^n \mu_i = 1$;

$\mu_i \leftarrow \mu_i \delta_i$, where $\sum_{i=1}^n \mu_i = 1$;

$S^{(0)} \leftarrow \emptyset$; $\gamma^{(0)} \leftarrow 0$;

$s \leftarrow 0$;

while true **do**

  Obtain top-l subgraph decision stumps $P = \{(g_i, \pi_i)\}_{i=1,\cdots,l}$; //Algorithm 7;

  $\mathbb{i}(g^\star, \pi^\star) = \max_{(g_j,\pi_j)\in P} \mathbb{i}(g_j, \pi_j)$

  if $\mathbb{i}(g^\star, \pi^\star) \leq \gamma^{(s-1)} + \varepsilon$ **then**

    break;

  $S^{(s)} \leftarrow S^{(s-1)} \bigcup P$;

  Obtain the membership value φ_i for each graph example G_i based on $S^{(s)}$ and Eq. (6.5);

  Solve Eq. (6.8) to get $\gamma^{(s)}$, $\boldsymbol{\mu}^{(s)}$, and Lagrange multipliers $\boldsymbol{w}^{(s)}$;

  $s \leftarrow s + 1$;

return $\mathcal{H}_t(G_i) = \sum_{(g_j,\pi_j)\in S^{(s-1)}} w_j^{(s-1)} \hbar(G_i; g_j, \pi_j)$;

top-l subgraphs $P = \{(g_i, \pi_i)\}_{i=1,\cdots,l}$ in each round (step 6). At step 7, we obtain the most optimal score $\mathbb{i}(g^\star, \phi^\star)$. If the best pattern in the current round no longer violates the constraint, the iteration process stops (steps 8-9). To speed up the boosting process, we relax the stopping condition and terminate the loop as soon as the change of the optimal value becomes subtle (ε). On steps 10-12, the linear programming problem in Eq.(6.8) is solved based on the selected subgraphs using the open source software CVX, a package for specifying and solving convex programs [50, 51]. After Eq.(6.8) is solved, we obtain two sets of weights: (1) $\boldsymbol{\mu}^{(s)} = \{\mu_1^{(s)}, \cdots, \mu_n^{(s)}\}$, the weights of training graph for optimal subgraph mining in the next round; and (2) $\boldsymbol{w}^{(s)} = \{w_1^{(s)}, \cdots, w_{|S^{(s)}|}^{(s)}\}$, the weights for subgraph decision stumps in $S^{(s)}$, which can be obtained from the Lagrange multipliers of dual problem in Eq.(6.8). Once the algorithm converges, the final classification model $\mathcal{H}(G_i)$ is returned on step 14.

Subgraph mining: In order to mine the top-l subgraphs (step 6 of Algorithm 6), we need to enumerate the entire set of subgraph patterns, with respect to a given threshold, from the training graphs D_t. In our boosting algorithm, we employ a Depth-First-Search (DFS) based algorithm gSpan [151] to enumerate subgraphs. The key idea of gSpan is that each subgraph has a unique DFS Code, which is defined by the lexicographic order of the time the subgraph is discovered during the search process. By employing a depth first search strategy on the DFS Code tree (where each node is a subgraph), gSpan can enumerate all frequent subgraphs efficiently. To speed up the enumeration, we utilize a *branch-and-bound pruning rule* [120] to prune the search space.

Theorem 3. *Given a subgraph feature g_j, let*

$$\begin{aligned} \dot{\mathbb{i}}_+^{(g_j)} &= 2\sum_{\{i|y_i=+1, g_j \in G_i\}} \mu_i^{(t)} - \sum_{i=1}^n y_i \mu_i \\ \dot{\mathbb{i}}_-^{(g_j)} &= 2\sum_{\{i|y_i=-1, g_j \in G_i\}} \mu_i^{(t)} + \sum_{i=1}^n y_i \mu_i \\ \dot{\mathbb{i}}(g_j) &= \max\left(\dot{\mathbb{i}}_+^{(g_j)}, \dot{\mathbb{i}}_-^{(g_j)}\right) \end{aligned} \tag{6.9}$$

If $g_j \subseteq g'$, the discriminative score $\mathbb{i}(g', \pi') \leq \dot{\mathbb{i}}(g_j)$.

Because a subgraph decision stump may have a positive or a negative label

Algorithm 7 Imbalanced Subgraphs Mining

Require:

$D_t = \{(G_1, y_1), \cdots, (G_n, y_n)\}$: Graph Datasets;

$\boldsymbol{\mu} = \{\mu_1, \cdots, \mu_n\}$: Weights for graph example;

l: Number of optimal subgraph patterns;

min_sup: The minimum support for optimal subgraphs;

Ensure:

$P = \{(g_i, \pi_i)\}_{i=1,\cdots,l}$: The top-$l$ subgraphs;

1: $\eta = 0$, $P \leftarrow \emptyset$;
2: **while** Recursively visit the DFS Code Tree in gSpan **do**
3: $g_p \leftarrow$ current visited subgraph in DFS Code Tree;
4: **if** g_p has been examined **then**
5: **continue**;
6: Compute score $\mathbb{i}(g_p, \pi_p)$ for subgraph g_p according Eq.(6.7);
7: **if** $|P| < l$ or $\mathbb{i}(g_p, \pi_p) > \eta$ **then**
8: $P \leftarrow P \bigcup (g_p, \pi_p)$;
9: **if** $|P| > l$ **then**
10: $(g_q, \pi_q) \leftarrow \arg\min_{(g_x,\pi_x)\in P} \mathbb{i}(g_x, \pi_x)$;
11: $P \leftarrow P/(g_q, \pi_q)$;
12: $\eta \leftarrow \min_{(g_x,\pi_x)\in P} \mathbb{i}(g_x, \pi_x)$
13: **if** $sup(g_p) > min\_sup$ & $\mathbb{i}(g_p) > \eta$ **then**
14: Depth-first search the subtree rooted from node g_p;
15: **return** $P = \{(g_i, \pi_i)\}_{i=1,\cdots,l}$;

$\mathcal{Y} = \{+1, -1\}$, we calculate its maximum score based on each possible value, and select the maximum one as the upperbound.

According to Theorem 3, once a subgraph g_j is generated, all its super-graphs are upperbounded by $\mathbb{i}(g_j)$. Therefore, this theorem can help reduce the search space.

Our branch-and-bound subgraph mining algorithm is listed in Algorithm 7. The minimum value η and subgraph set P are initialized on step 1. We prune the duplicated subgraph features on steps 4-5, and compute the discriminative score $\mathbb{i}(g_p, \pi_p)$ for g_p on step 6. If $\mathbb{i}(g_p, \pi_p)$ is larger than η or the current set P has less than l subgraph patterns, we add (g_p, π_p) to the feature set P (steps 7-8). If the size of P exceeds the predefined size l, the subgraph with the minimum discriminative score is removed (steps 9-11). We use two metrics, the minimum support for subgraph g_p and a branch-and-bound pruning rule, similar to the

rule in [120], to prune search space on steps 13-14. The optimal set P is returned on step 15. It is worth noting that our algorithm is efficient in the sense that even if there is no minimum support threshold min_sup for subgraph mining, the algorithm can still function properly by only relying on the pruning rule.

6.4 gEBoost algorithm

In this section, we discuss the proposed ensemble framework, which combines classifiers trained from local chunks (as described in Section 6.3) to handle graph stream with dynamic changing volumes and concepts, *i.e.* concept drifting.

In graph stream settings, the correct classification of graphs with imbalanced class distributions are challenged by several key factors. First, noise presenting in the stream will deteriorate existing learned model and reduce the classification accuracy. Second, graph data may constantly evolve (*i.e.*, concept drifting) which will introduce misclassifications because the existing models do not have the knowledge of the emerging new concepts. Third, even within the observed concepts, there are always some "difficult" samples which cannot be correctly classified by current models. Accordingly, we used disturbing graph samples to capture the "difficult" samples, as defined in Definition 10.

The disturbing graph samples may be introduced by noise, concept drifting, or genuinely difficult samples on which existing imperfect model fail to handle. Because the existing model is incapable of classifying them, they need to be emphasized during the stream learning process. In our system, we use an instance based weighting method to capture disturbing graph samples, and further include the instance weight into the local classifier learning process (the objective function Eq.(6.4) and step 2 of Algorithm 6).

Instance Weighting: The idea of our weighting scheme is as follows: as soon as a new graph chunk D_t is collected for processing, we use an ensemble of classifiers $E = \{\mathcal{H}_{t-k}, \mathcal{H}_{t-k+1}, \cdots, \mathcal{H}_{t-1}\}$ trained from historical chunks to predict labeled graphs in D_t. If a graph is misclassified by E, we increase the graph's weight because it is a difficult sample for the current model E. If a graph is correctly classified by E, we decrease its weight because model E already has

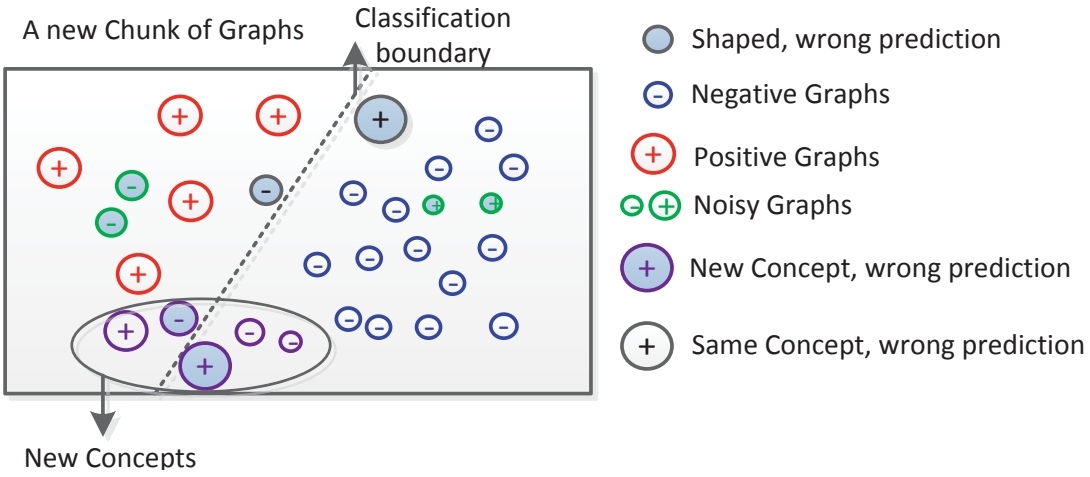


Figure 6.3: A conceptual view of graph weighting scheme for imbalanced graph stream classification. Given a new chunk of graphs with positive and negative graphs (circle sizes indicate the weights), the current classifier may make incorrect prediction on three kinds of disturbing graph samples: (1) For a noisy graph G_i (green circles), its weight will be first increased inversely proportional to the accuracy of the current ensemble classifier (measured by δ_i), and be further decreased according to G_i's distance to the class centres (correspond to φ_i). (2) For emerging new concept graphs (purple circles), if the current ensemble makes an incorrect prediction, their weights will be increased by δ_i because current model needs to emphasis on these samples with new concepts. (3) For graphs sharing the same concepts as previous chunk (black circles), their weights will also increase (by δ_i) because they are difficult instances and the current classifier can not correctly classify them. The weight updating scheme will help differentiate different types of disturbing graphs for training effective graph stream classifier.

sufficient knowledge to correctly classify this graph. This weighting mechanism is similar to Adaboost [42] and our semi-supervised graph stream classification method [101]. By doing so, we can tune instance weights to capture disturbing samples (including concept drifting under neath the stream), so gEBoost can emphasize on difficult samples and select a set of informative features to build better models. Our weighting scheme is illustrated in Fig. 6.3.

gEBoost Algorithm: Algorithm 8 lists detailed procedures of gEBoost framework which combines instance weighting and graph boosting for graph stream classification.

The "while" loop in Algorithm 8 represents a stream processing cycle which repeats as graph data continuously arrive. For the first graph chunk D_1, gEBoost

Algorithm 8 gEBoost

Require:
$\mathbb{S} = \{D_1, D_2, \cdots\}$: A Graph Stream
k: The maximum capacity of the ensemble

1: Initialize $E = \emptyset, t = 0$;
2: **while** $\mathbb{S}! = \emptyset$ **do**
  // Training Phase:
3: $D_t \leftarrow$ A new graph chunk;
4: $\mathbb{S} \leftarrow \mathbb{S}/D_t$; $t = t + 1$;
5: **if** $(t == 1)$ **then**
6: $\delta_i = 1, i = 1, \cdots, n$;
7: $\mathcal{H}_t \leftarrow$ classifier built from D_t and $\{\delta_i\}_{i=1,\cdots,n}$; //Algorithm 6;
8: $E \leftarrow E \bigcup \mathcal{H}_t$
9: **else**
10: $err \leftarrow \frac{\sum_{G_i \in D_t^l}(E(G_i) \neq y_i)}{|D_t^l|}$;
11: $\delta_i = \begin{cases} \delta_i \sqrt{(1-err)/err} & : \quad E(G_i)! = y_i, G_i \in D_t \\ \delta_i \sqrt{err/(1-err)} & : \quad E(G_i) = y_i, G_i \in D_t \end{cases}$;
12: $\delta_i = \frac{\delta_i}{\sum_{G_i \in D_t} \delta_i}$;
13: $\mathcal{H}_t \leftarrow$ classifier built from D_t and $\{\delta_i\}_{i=1,\cdots,n}$; //Algorithm 6;
14: $E \leftarrow E \bigcup \mathcal{H}_t$
15: **if** $|E| > k$ **then**
16: $E \leftarrow E/\mathcal{H}_{t-k}$
  // Testing Phase:
17: $D_{t+1} \leftarrow$ A new graph chunk;
18: $\alpha_i \leftarrow \mu_p I(H_i(G_p) == y_p), G_p \in D_t$;
19: $\mathcal{H}(G_p|E) = \arg\max \sum_{i=t-k-1}^{t} \alpha_i \mathcal{H}_i(G_p)$

simply builds a linear boosting classifier using Algorithm 6 without considering concept drifting ($\delta_i = 1$), and adds classifier $\mathcal{H}_t$ to initialize the ensemble E (lines 5-8).

For each of the succeeding chunks $D_t, t = 2, 3, \cdots$, gEBoost uses ensemble E and its error rate err to tune the weight of each graph in D_t (line 10), where $E(G_i)$ returns the class label of G_i predicted by E. If a graph G_i's label is different from the one predicted by the ensemble classifier E, gEBoost increases the weight of G_i by $\sqrt{(1-err)/err}$, otherwise gEBoost decreases the weight of G_i by $\sqrt{err/(1-err)}$ (line 11). On lines 12-14, gEBoost normalizes the weight

values for all graphs in D_t, and builds a boosting classifier from D_t and $\{\delta_i\}_{1,\cdots,n}$ to update the ensemble E.

During the classification phase (lines 17-19), gEBoost first calculates the weighted accuracy α_i on the most recent chunk D_t ($I(x)$ returns 1 if x is true, otherwise 0), and then uses weighted voting to assemble all classifiers in E to predict graphs in a new graph chunk D_{t+1}. Note that μ_p (line 18) is determined in Algorithm 6, representing the weight for graph G_p.

6.5 Experiments

We report our experiments on real-world graph streams to validate (1) the effectiveness of the proposed algorithm for handling noisy and imbalanced graphs; and (2) the efficiency and effectiveness of gEBoost for graph stream classification. The source code, benchmark data, and detailed results can be downloaded from our online report [111].

6.5.1 Experimental Settings

Three graph streams, DBLP, NCI, and Twitter graph streams (please refer to table 3.2 and Section 3.3 for more details), collected from real-world applications are used in our experiments.

- **DBLP Graph Stream:** The imbalanced version of DBLP graph stream, DBLP-imbalanced (detailed in table 3.2), is used here for studying the effectiveness of our algorithm.

- **NCI Graph Stream:** The NCI collection (shown in table 3.2), is naturally imbalanced in its class distributions, with less than 5% graphs in the positive class. In our experiments, we concatenate two datasets (detailed in table 6.1), NCI-1 and NCI-33, as one stream with 74,371 graphs in total (4.38% samples belonging to positive class). In this case, the bioassay task changes from NCI-1 to NCI-33, which simulates the concept drifting in the stream

Table 6.1: NCI cancer screen datasets used in the experiments

Bioassay-ID	Original Compounds		New Compounds		$\|Pos\|\%$
	#Pos	#Total	#Pos	#Total	
NCI1	2295	42324	1793	37349	4.80
NCI33	1857	41971	1467	37022	4.00

(*i.e. sudden drift*). We sequentially construct graph chunks such that each chunk consists of 4.38% positive graphs and others are negative.

- **Twitter Stream** We also use the twitter graph stream (shown in table 3.2) in our experiments. Note that to investigate algorithm performance in handling concept drifts, we synthetically control the prior distribution of positive graphs at several fixed time stamps. Specifically, 20% of positive graphs are randomly selected on Monday and Tuesday over time before June 2. By doing so, we use sudden changes of priori distributions to inject concept drifting on Monday.

Noise and class imbalance in graph chunk: In our experiments, each chunk D_t in graph streams has a small number of positive graphs D_t^p, and a large number of negative graphs D_t^n, where $|D_t^p| = |D_t| * |Pos|\%$, and $|D_t^n| = |D_t| - |D_t^p|$. For instance, for the NCI graph stream (with $|Pos|\% = 4.38\%$), if the chunk size $|D_t| = 1500$, then there are 1500 * 4.38%=66 positive and 1434 negative graphs, respectively. For Twitter graph stream, the graph chunks on Monday and Tuesday are imbalanced (with 20% of positive graphs), whilst graphs on other days are relatively balanced.

To systematically study the algorithm performance in noisy data environments, we introduce noise to each stream as follows. Given a graph chunk D_t with $|D_t^p|$ positive graphs, we randomly select $|D_t^p| * Z\%$ positive graphs and $|D_t^p| * Z\%$ negative graphs, and flip their class labels (*i.e* change a positive graph as negative, and vice versa). Because majority graphs are negative, this random approach will have a severer impact on positive class than negative class.

Baselines: There are few existing methods for graph stream classification [2][25], but they are incremental learning approaches, whereas our method, gEBoost, is an ensemble framework. Because they are two types of methods, it is very

difficult to make direct comparisons with these methods. More specifically, the algorithms in [2][25] employ hashing for classification. Whenever a graph arrives, the hashed values of graph edges are used to build a classifier. For new graphs in the stream, they continuously use the hashed values of the graph to update their classifier. In their experiments, the validation of the stream classification models was done by evaluating the accuracy of the model on a separated test set. In the proposed gEBoost method, we use a divide-and-conquer based ensemble framework, which partitions stream into small chunks, and uses classifiers trained from graph chunks to form an ensemble for prediction. The validation was done by evaluating the accuracy of the model on the next available future graph chunk. Another clear advantage of our method is that we extract sub-graph features to represent graphs in a vector format, so any learning algorithm can be used for graph stream classification. Whereas [2][25] are limited to their own learning algorithms (*e.g.*, [2] can only use k-NN classifier).

For gBoost [120], we implement two variants of gBoost to handle class imbalance. Because gBoost is designed for static datasets, we incorporate gBoost into our ensemble framework (like gEBoost does, *i.e.*, setting $\delta_i = 1$) for graph stream classification. The detailed baselines are as follows:

- **gBoost-b+Stream** first under-samples graphs in the majority (negative/inactive) class in each chunk to create a balanced graph set to train a gBoost classifier [120]. The most recent k chunks form an ensemble to predict graphs in a future chunk.

- **gBoost+Stream** applies the gBoost algorithm [120] in each graph chunk directly. An ensemble of gBoost classifiers (like gEBoost) is used to classify graphs in a future chunk.

- **Learn++.CDS-DT** first mines a set of frequent subgraphs as features, and then transfer graphs into vector format. The Learn++.CDS [31] is performed on the transferred vector data with Decision Tree (DT) as a based classifier on each chunk.

- **Learn++.NIE-gBoost** learns gBoost classifiers in each chunk with a Bagging strategy to combat data imbalance, and then we apply Learn++.NIE

[31] algorithm to k consecutive chunks to form an ensemble classification for graph stream classification.

Note that Learn++.CDS cannot combines with gBoost algorithm, because it needs to generate synthetic positive samples based on the vector data to handle data imbalance. By contrast, Learn++.NIE employs a bagging strategy to combat data imbalance, so we integrate gBoost with Learn++.NIE as a baseline. It is worth noting that Learn++.NIE-gBoost works on graphs directly (as gEBoost does). However its subgraph feature exploration process (gBoost) does not take class imbalance and noise into consideration.

Measurement and Parameter Setting For imbalanced data, accuracy is no longer an effective metrics to assess the algorithm performance, so we use AUC (Area Under the ROC Curve) as the performance measurement in our experiments.

Unless specified otherwise, we use following default parameter settings in the experiments: Ensemble size k=10, chunk size $|D_t| = 800$ (for DBLP) and 1500 for (NCI). For gEBoost, we set $\beta = \frac{|D_t^n|}{|D_t^p|}$ as the imbalance ratio, and the decay factor $\tau = 0.1$, the relax factor $\epsilon = 0.01$ for DBLP and Twitter, and 0.05 for NCI streams, respectively. The number of top-l subgraphs selected in each round is 25. For parameter v ($C = \frac{1}{vn}$), we set different values for different algorithms. Specifically, v is set to 0.2 for DBLP graph streams for all boosting algorithms. For NCI and Twitter graph streams, we set v=0.05 for gBoost+Stream and gBoost-b+Stream, and $v = 0.5$ for gEBoost. For NCI data stream, we set $min\_sup = 15\%$ and 0 for DBLP and Twitter graph stream, which means no support threshold is provided in these two streams for subgraph mining. We use parameters suggested in [31] for both Learn++.CDS-DT and Learn++.NIE-gBoost algorithms.

6.5.2 Experimental Results

6.5.2.1 Performance on Noisy and Imbalanced Graph Chunks

To report our boosting modules for noisy and imbalanced graph data, we built a classifier $\mathcal{H}_t$ from the current chunk D_t (as discussed in Section 6.3) to predict graphs in the next chunk D_{t+1}. In this experiment, no instance weighting and

ensemble framework are involved, because we want to know whether gEBoost's objective function in Eq.(6.4) can indeed handle data imbalance and noise in the graph chunks. We report the average classification accuracy of different methods *w.r.t.* different levels of noise over the whole stream in Fig. 6.4.

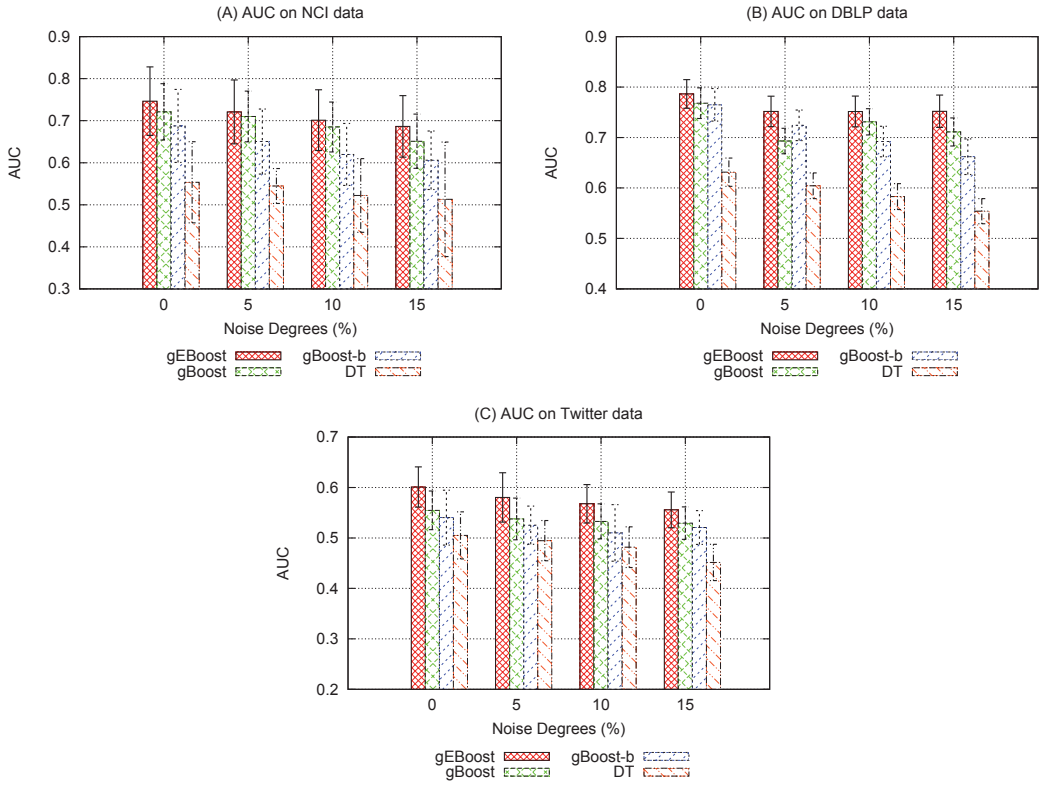


Figure 6.4: Comparison of different algorithms for imbalanced graph stream classification. For each chunk D_t, we build a classifier $\mathcal{H}_t$ from chunk D_t to predict graphs in D_{t+1}. The results represent the average AUC values and standard deviation over all chunks in each graph stream. (A) NCI stream; (B) DBLP stream; (C) Twitter stream.

The results in Fig. 6.4 demonstrate that gEBoost outperforms its peers on all graph streams. Among all boosting methods, gBoost-b under-samples graphs from the majority (negative) class to create a balanced graph chunk before applying gBoost, and its results are inferior to gBoost and gEBoost. This confirms the hypothesis of information loss during the under-sampling process, which results in low quality discriminative subgraph features for classification. Meanwhile, although both gBoost and gEBoost directly work on imbalanced graph data, gE-

Boost considers weights for samples in different classes. The results show that gEBoost is superior to gBoost, which ignores the class imbalance and noise issues, and treats all samples equally. Note that in our experiment, both gBoost+Stream and Learn++.NSE-gBoost algorithms use gBoost as base classifiers. The DT base classifier, which is built on the vector data and employed in Learn++.CDS-DT algorithm, is worse than any other boosting algorithm.

The results in Fig. 6.4 also validate the effectiveness of our algorithm in handling noise. It is clear that noise deteriorates AUC values of all algorithms. This is because noise (*i.e* incorrectly labeled graphs) does not comply with the distributions of majority samples in the same class, and makes a learning algorithm difficult to separate positive and negative classes. The results show that our algorithm has much less performance loss when a higher degree of noise is imposed. This is mainly attributed to the distance penalties in the objective functions (φ_i for graph G_i of Eq. (6.4)) in gEBoost. More specifically, a negative graph, say G_i, is close to negative class center in the feature space. So even if G_i is incorrectly labeled as positive (*i.e.* a noise), it still has a large distance to the positive class center (because G_i is close and similar to negative graphs in the feature space). By using G_i's distance to the class center to adjust its role in the objective function, Fig. 6.4 confirms that combining class distributions and distance penalties of individual graph indeed help gEBoost effectively handle graph data with severely imbalanced class distributions and noise.

6.5.2.2 Performance on Graph Streams

In this subsection, we report the performance of the proposed ensemble framework for graph stream classification.

Results with Noise Degrees Z. In Figs. 6.5, 6.6, and 6.7, we vary the noise levels in each graph chunk, and report the results on NCI, DBLP, and Twitter streams.

The results in Figs. 6.5, 6.6, and 6.7 show that gEBoost consistently outperforms all other algorithms across the whole stream for all noise levels. In our experiments, Learn++.CDS-DT has the worst performance because: (1) it uses a set of frequent subgraph as features, which may fail to obtain genuine discrimi-

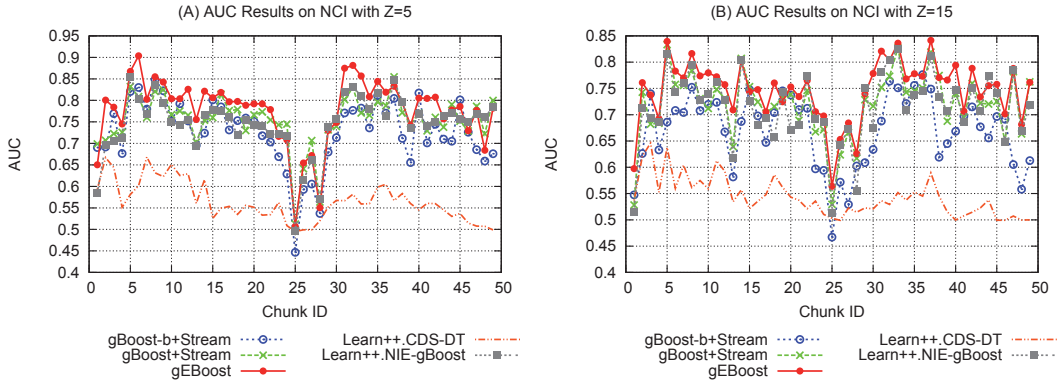


Figure 6.5: AUC *w.r.t.* different noise levels on *NCI stream* with ensemble size k=10 and chunk size D_t= 1500. (A)$Z = 5$; (B) $Z = 15$.

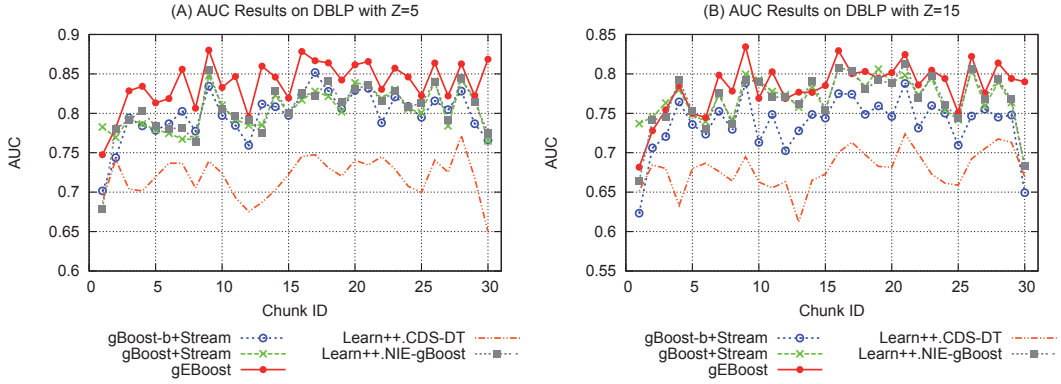


Figure 6.6: AUC *w.r.t.* different noise levels on *DBLP stream* with ensemble size k=10 and chunk size D_t= 800. (A)$Z = 5$; (B) $Z = 15$.

native subgraphs to build classification models; (2) it over-samples minority class samples to handle class imbalance, which may introduce ambiguousness to the sampled data; and (3) Learn++.CDS in each chunk use a single weak classifier (DT) while other algorithms assemble a set of decision stumps for graph classification. It is generally believed that an ensemble often outperforms a single classifier.

The results also show that gBoost-b+Stream, which under-samples graphs in each chunk to alleviate the data imbalance, is significantly inferior to gBoost+Stream, Learn++.NIE-gBoost, and gEBoost, especially in Fig. 6.6 (B). This is because each graph chunk is extremely imbalanced (*e.g.*, only 66 positive graphs out of

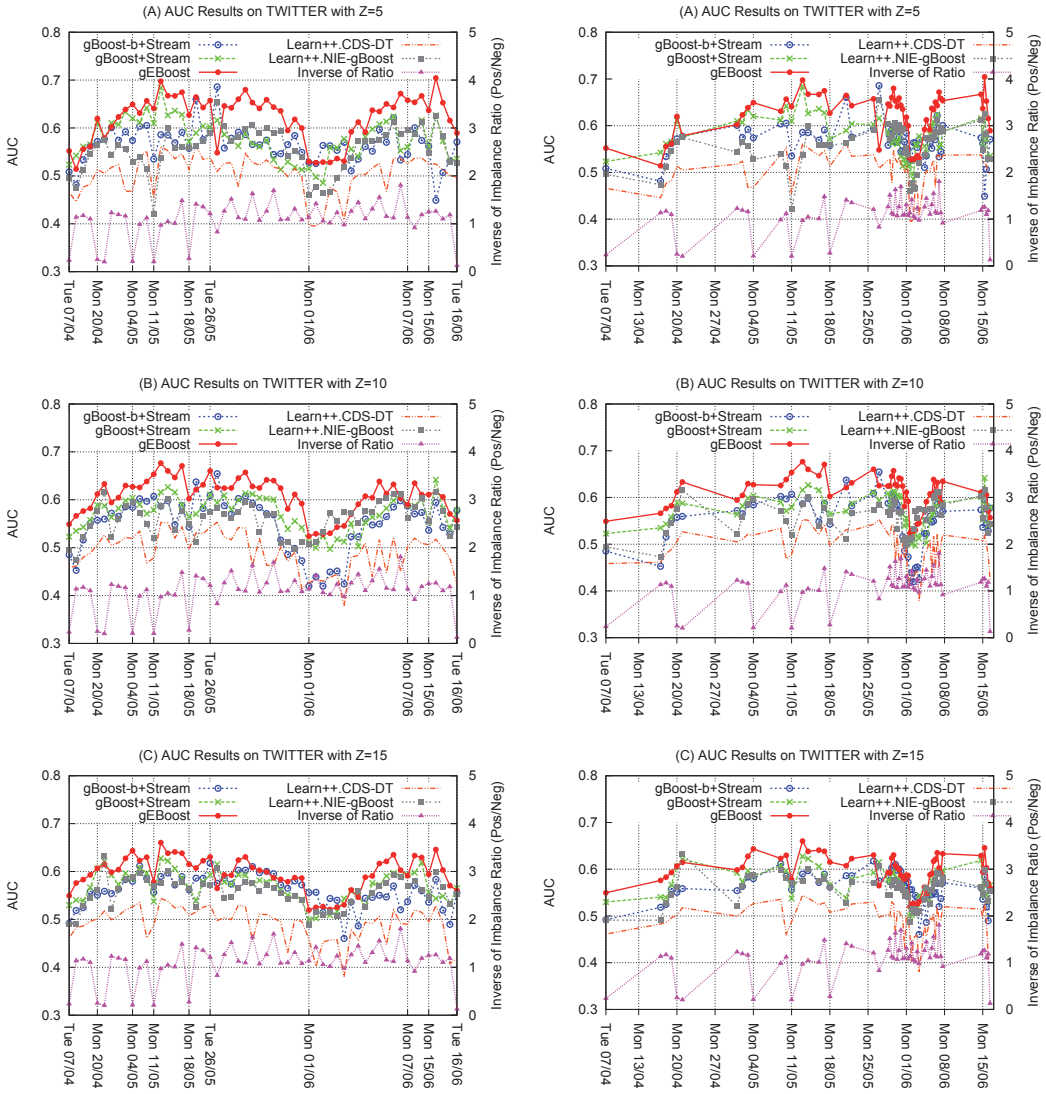


Figure 6.7: AUC *w.r.t.* different noise levels on *Twitter stream*. Figures on the left panel are plotted with respect to uniform intervals of chunks in the x-axis, and figures on the right panel are plotted with respect to uniform intervals of weeks in the x-axis.

1500 graphs for NCI stream), under-sampling will result in balanced graph chunks with significantly smaller sizes, which makes subgraph feature selection and margin learning process very ineffective. gEBoost demonstrates a better performance than gBoost+Stream and Learn++.NIE-gBoost in all streams. This is mainly attributed to gEBoost's two key components, including (1) boosting framework for feature selection and margin maximization, and (2) weighting to tackle concept drifting. The former iteratively selects a set of discriminative features and maximizes the margin sequentially, and the latter allows multiple chunks (classifiers) to work in a collaborative way to form an accurate ensemble model. As a result, gEBoost achieves good performance in classifying graph streams with dynamic changes. For example, in Fig. 6.5, there are sudden concept drifting from chunks 25-30, where the bioassay task changes from NCI-1 to NCI-33, and all three methods experience performance loss. By employing instance weighting to tackle the concept drifting, gEBoost incurs much less loss than gBoost+Stream and Learn++.NIE-gBoost.

It is worth noting that Learn++.NIE-gBoost is a specially designed algorithm for imbalanced data streams. In our experiments, the results in Figs. 6.5, 6.6, and 6.7 show that Learn++.NIE-gBoost is only comparable to gBoost+Stream, yet significantly worse than gEBoost algorithm. Indeed, for noisy and imbalanced graph streams, finding most effective subgraph features plays an essential role. This is a major challenge for graph streams, whereas Learn++.NIE-gBoost may fail to explore high quality subgraphs under imbalanced and noisy scenarios for graph stream classification.

Twitter Stream: We investigate the algorithm performance in handling concept drifting in Twitter stream in Fig. 6.7. The inverse of imbalance ratio ($\frac{|Pos|}{|Neg|}$) provides an indicator of the change of prior class distributions (y-axis on right side) over time (x-axis). Specifically, there are significant concept drifts on Monday and Tuesday before June 2, whilst class distribution (concept drift) remains relatively stable from June 2 and afterwards. The results show that for some concept drifting points, there are indeed noticeable performance drops for most algorithms. For instance, in Fig. 6.7 (A), the AUC values slightly decrease on May 11 and 18, and have a significant drop on June 1. The proposed gEBoost outperforms all other algorithms in handling sudden concept drifting. Another

interesting observation is that not all concept drift points will result in drop of AUC for gEBoost. For instance, on May 4 of Fig. 6.7 (A), while Learn++.CDS-DT witnesses a performance loss, gEBoost has an increase of AUC index, which shows gEBoost's good ability in handling concept drifts.

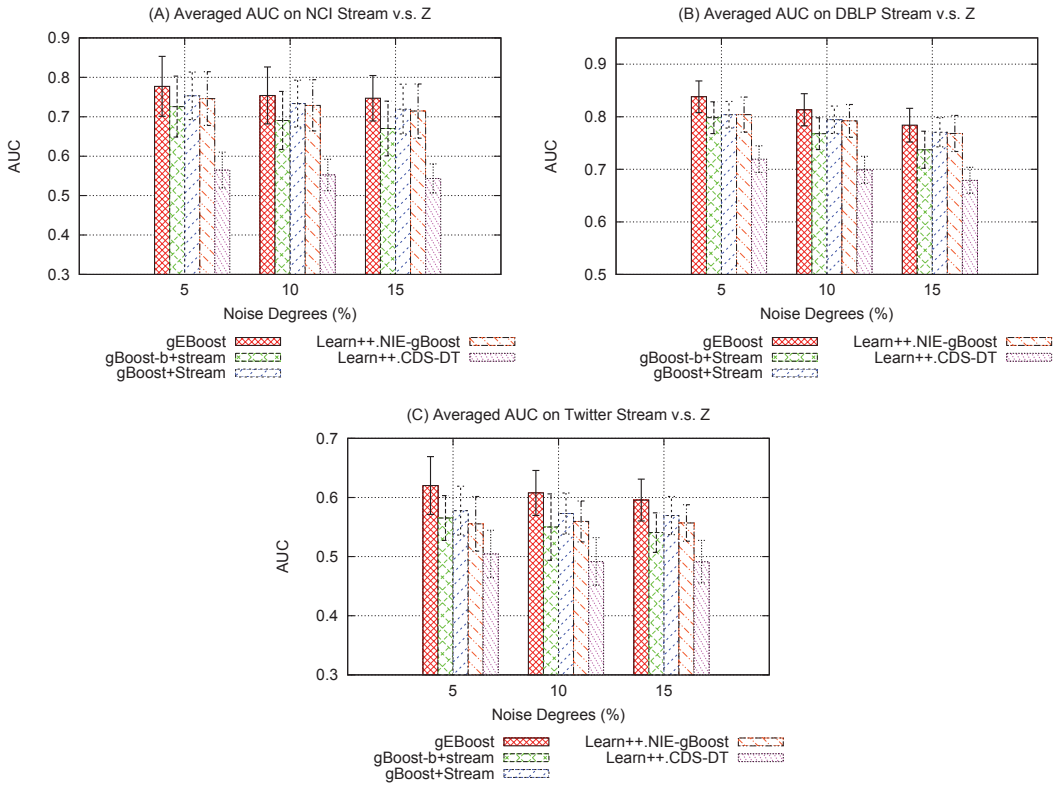


Figure 6.8: Averaged AUC values (and standard deviation) *v.s.* different noise degrees Z, with ensemble size k=10.

The average accuracies over the whole graph stream, in Fig. 6.8, show that increasing the noise degree in each chunk deteriorates the performance of all algorithms (which is consistent with our previous discussions). We also conducted pairwise t-test to validate the statistical significance of comparisons. The results show that gEBoost outperforms others significantly.

Results on Ensemble Size k. In Figs. 6.9 and 6.10, we report the algorithm performance by using different ensemble size k (varying from 5, 10, to 15) for DBLP and Twitter streams. Similar result for NCI Streams is obtained for NCI streams.

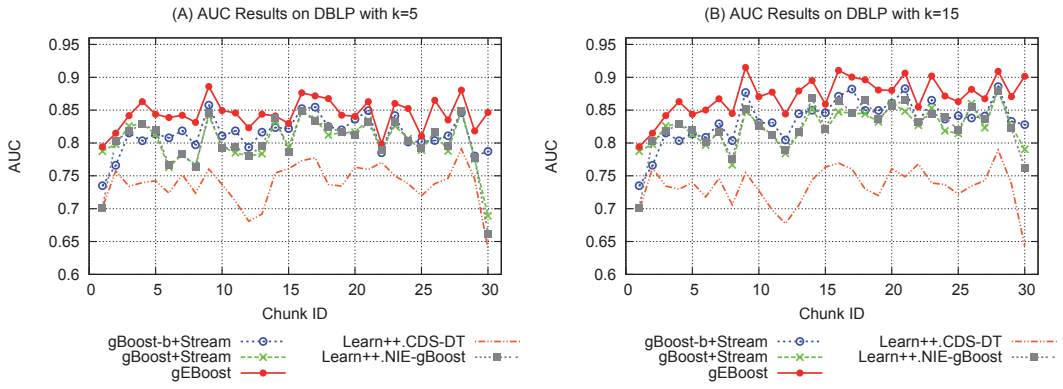


Figure 6.9: AUC *w.r.t.* different ensemble sizes on *DBLP stream* with chunk size $|D_t|$=800.

The results show that increasing ensemble size results in improved algorithm performance. For instance, when k=5 (in Fig. 6.9 (A)), all algorithms have low AUC values in DBLP graph stream. When increasing ensemble size from 5 to 15, each algorithm experiences steady improvement across the whole DBLP stream. This is mainly because a larger ensemble involves more classifier models and more knowledge for prediction. However, for large ensemble size, it will also increase computational complexity to predict graphs. In the remaining experiments, we set k=10.

Results on chunk size D_t: In Figs. 6.11 and 6.12, we report the algorithm performance with respect to different numbers of graphs in each chunk $|D_t|$.

As expected, gEBoost has the best performance among three algorithms for NCI (Fig. 6.11) and DBLP (Fig. 6.12) streams. When varying the chunk sizes, the concept drifting may occur at different locations for NCI streams. Nevertheless, our results show that gEBoost can adapt to the concept drift very quickly in most cases (Figs. 6.11 (A) and (B)), which validates the effectiveness of gEBoost in handling concept drift. In practice, the chunk size should be a moderate value. For small chunk sizes, the models trained from each chunk will be inaccurate, because no sufficient information is available for extracting discriminative subgraph features to train classifiers. For large chunk sizes, a graph chunk may include several changing concepts, which will deteriorate the learner performance.

Results on Imbalanced Degree $|Pos|$%. To study the algorithm performance

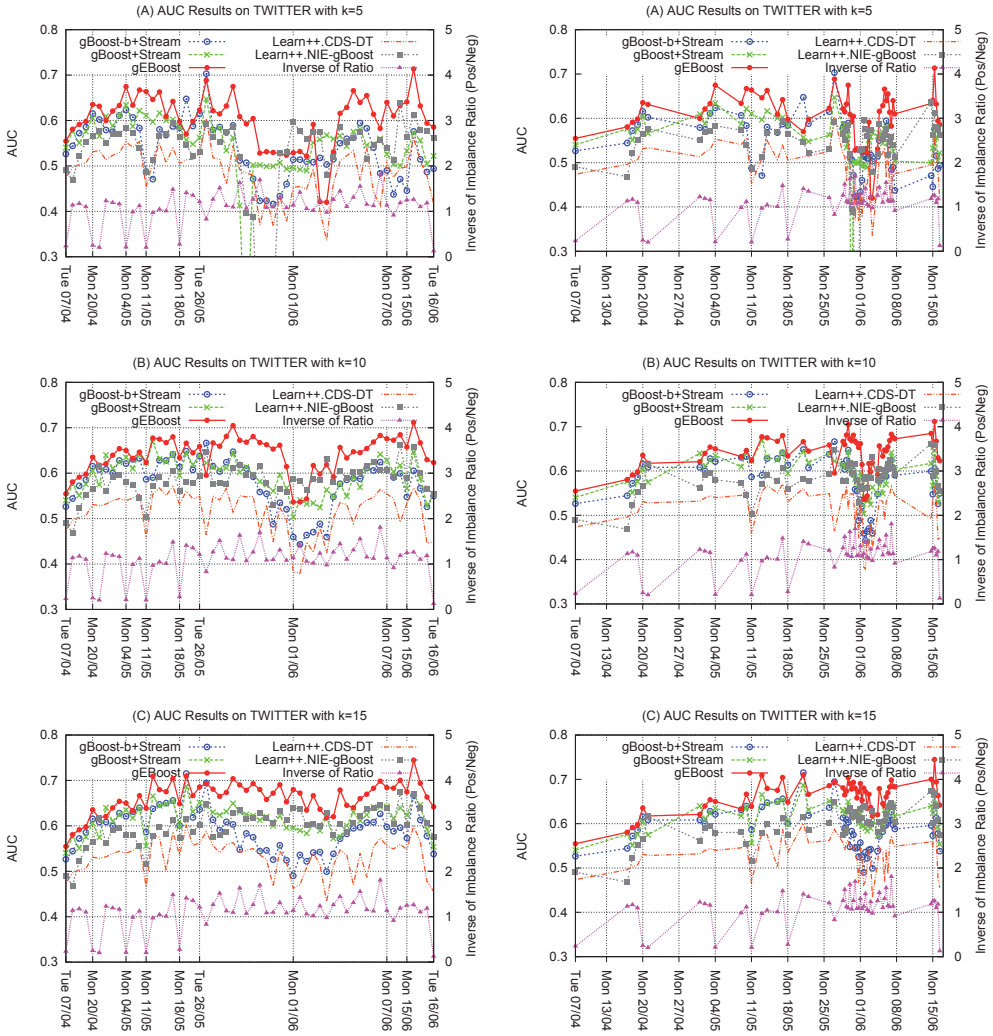


Figure 6.10: AUC *w.r.t.* different ensemble sizes on *Twitter stream*. Figures on the left panel are plotted with respect to uniform intervals of chunks in the x-axis, and figures on the right panel are plotted with respect to uniform intervals of weeks in the x-axis.

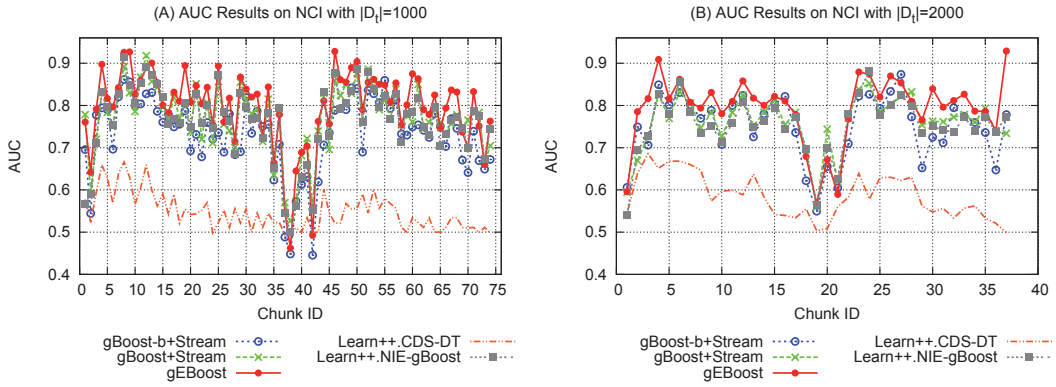


Figure 6.11: AUC *w.r.t.* different chunk size on *NCI stream* with ensemble size k=10. (A) $|D_t| = 1000$; (B) $|D_t| = 2000$.

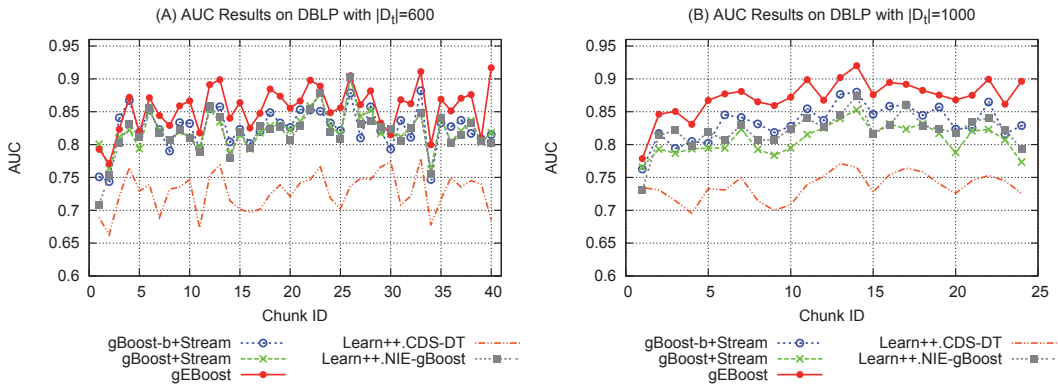


Figure 6.12: AUC *w.r.t.* different chunk size on *DBLP stream* with ensemble size k=10. (A) $|D_t| = 600$; (B) $|D_t| = 1000$.

w.r.t. different data imbalance degrees, we change the percentage of positive graphs ($|Pos|\%$) on DBLP streams. In previous experiments, $|Pos|\%$ in each chunk is 16.3. So we under-sample positive graphs in each chunk to create streams with different imbalance levels. The averaged experimental results over streams are reported in table 6.2.

Table 6.2 shows that with the increase of data imbalance degrees (changing $|Pos|\%$ from 16.3 to 5), the performance of all algorithms deteriorate in terms of averaged AUC values. This is because reducing the number of positive graphs increases the difficulty to learn good classification models in each chunk. Nevertheless, the proposed gEBoost outperforms all other algorithms under all levels

of degrees, which demonstrates the robustness of our algorithm.

Table 6.2: Average AUC values and standard deviations on DBLP Streams w.r.t Different Imbalance Degrees

$\|Pos\|$%	gBoost-b +Stream	gBoost +Stream	gEBoost	Learn++ .CDS-DT	Learn++ .NIE-gBoost
16.3	$0.830_{\pm 0.032}$	$0.823_{\pm 0.030}$	$\mathbf{0.867}_{\pm 0.028}$	$0.736_{\pm 0.028}$	$0.819_{\pm 0.039}$
10	$0.798_{\pm 0.031}$	$0.819_{\pm 0.032}$	$\mathbf{0.836}_{\pm 0.028}$	$0.723_{\pm 0.029}$	$0.812_{\pm 0.037}$
5	$0.775_{\pm 0.035}$	$0.798_{\pm 0.030}$	$\mathbf{0.818}_{\pm 0.031}$	$0.710_{\pm 0.031}$	$0.801_{\pm 0.039}$

6.5.2.3 Time and Memory Comparisons

Time Efficiency. The runtime efficiency in Figs. 6.13 and 6.14 shows that Learn++.CDS-DT consumes least time among these algorithms. This is because Learn++.CDS-DT builds a simple decision tree in each chunk whereas other methods involve a boosting process. Meanwhile, gBoost-b+Stream requires much less runtime than other boosting algorithms. This is mainly because gBoost-b carries out boosting procedure on a small subset of under-sampled graphs whereas gEBoost, gBoost+Stream, and Learn++.NIE-gBoost directly work on all graphs in each chunk. In our experiments, the down-sampled (and balanced) graphs for gBoost-b is less than 10% of each chunk, so gBoost-b+Stream has much better runtime performance. When comparing gEBoost, gBoost+Stream, and Learn++.NIE-gBoost, an interesting finding is that gEBoost requires much less runtime than gBoost+Stream and Learn++.NIE-gBoost on NCI and Twitter stream, but consumes more runtime than other two approaches on DBLP streams. This observation may be caused by different properties of different graph datasets. Meanwhile, the accumulated system runtime *w.r.t.* different chunk sizes, as shown in Fig. 6.14, also indicate that system runtime remains relatively stable for different chunk sizes. Overall, gEBoost linearly scales to the number of graphs and chunks, which makes it capable of handling real-world high speed graph streams.

Memory Consumption: The main memory consumption of gEBoost is spent on the subgraph enumeration procedure. As each chunk is a relative small graph

set, only a small amount of memory is required for our subgraph mining component. Meanwhile, because our algorithm utilizes an ensemble based framework, all graphs flows in a "one-pass" fashion, *i.e.* historical graphs are discarded after being processed, only a set of discriminative features (decision stumps) and a gEBoost classifier are kept in the memory. The obsolete classifiers are removed whenever the ensemble size is full. As a result, the memory consumption for stream classification is relatively constant for our algorithm. We never experienced any out of memory errors on a computer with 8GB memory.

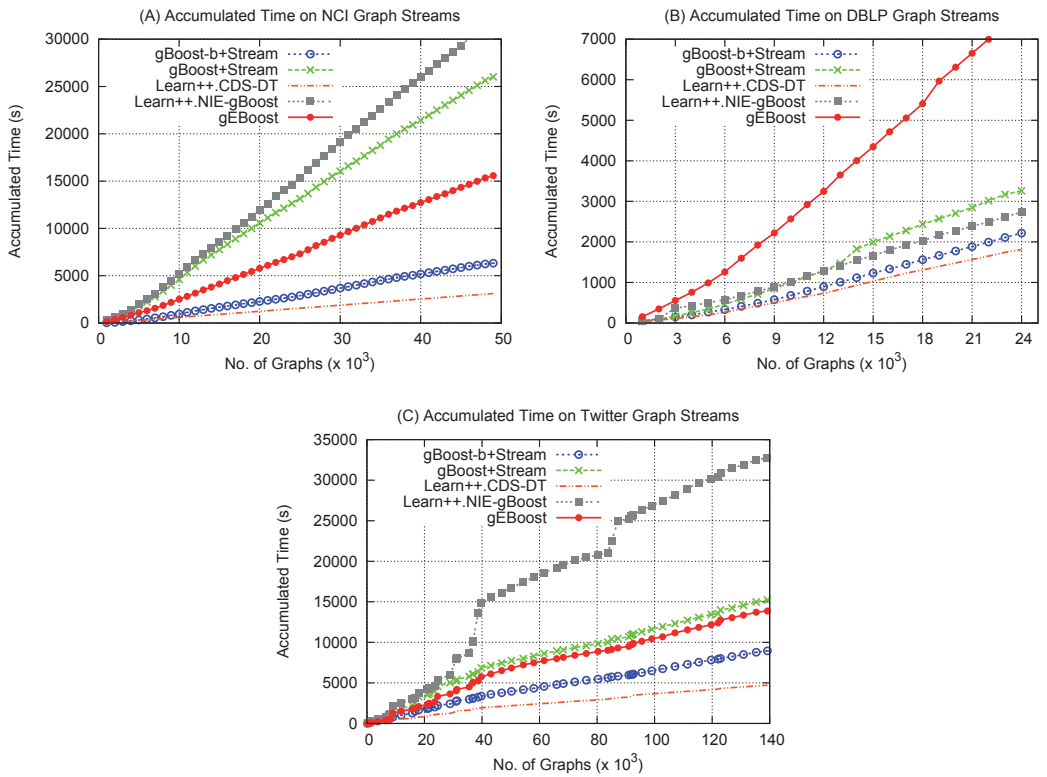


Figure 6.13: System accumulated runtime *v.s.* number of graphs processed over stream. (A) NCI stream; (B) DBLP stream; (C) Twitter stream.

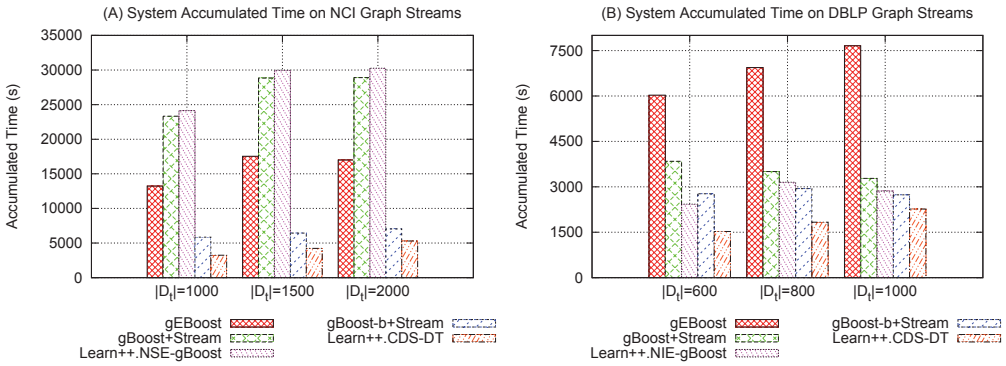


Figure 6.14: System accumulated runtime *v.s.* different chunk sizes $|D_t|$.

6.6 Conclusion

In this chapter, we investigated graph stream classification with imbalanced class distributions. We argued that existing work inherently overlooked the class distributions in the graph data, so the selected subgraph features are biased to the majority class, which makes algorithms vulnerable to imbalanced class distributions and noise. The concept drifting over stream further complicates the learning task for graph classification. In the chapter, we proposed an ensemble based framework to partition graph stream into chunks, with a boosting classifier being learnt from each chunk. The boosting procedure considers class distributions to weight individual graphs, so the selected subgraph can help find optimized margins, which further help explore new subgraph features. To handle concept drifting in the stream, each graph is carefully weighed by using classifiers learnt from previous stream. We believe our graph stream model is more useful comparing to non-stream technique, this is because the runtime for finding subgraph features from the whole graph set can be very expensive. Unless we use a very large support value, it will be very time consuming to find frequent subgraphs from a large graph set. Moreover for a graph stream, like Twitter stream, the concepts may gradually change, so the stream classification model is able to adapt to such changes for accurate prediction. Our experimental results validate the effectiveness of our algorithm.

Part III

Complex Task Graph Classification

Complex Task Classification: Overview

Apart from insufficiently labeled data or imbalanced class distributions of the graph objects, real-life applications of graph classification usually have various complex constraints. For instance, the misclassification cost for different graph samples may be different, the data size may be particularly large, and there may be several similar graph classification tasks co-existing while each has only a limited number of labeled graphs. In Part III, we will study the following two complex task graph classification problems:

Cost-sensitive Learning for Large Scale Graph Classification: All existing graph classification algorithms assume, explicitly or implicitly, that misclassifying instances in different classes incurs an equal amount of cost/risk, which is often not the case in real-life applications. For instance, misclassifying a certain class of samples, such as diseased patients, is subject to more expensive costs than others.

Another challenge of existing graph classification algorithms is that they are only designed for small size graph datasets and are very inefficient to scale up to large scale graph datasets. As big data applications [148] are becoming increasingly popular for different domains and result in graph datasets with large volumes, finding effective algorithms for large scale graph classification is highly desired.

Consider cost-sensitive learning and large scale graph classification as a whole; in the thesis, we will study cost-sensitive learning for large scale graph classification (Chapter 7).

Multi-task Graph Classification: Although existing methods have advanced the graph classification from different perspectives, they typically share similar disadvantages in their designs: (1) in order to explore graph structures for training good classification models, they require a large number of training graphs; and (2) they can only work on one single learning task. In reality, due to the inherent complexity of the graph data and the costs involved in the labeling process, collecting a large number of labeled graphs for a specific task is difficult. However, it is quite common that multiple similar graph classification tasks, each

with a limited number of training samples, may co-exist and need to be handled.

In this thesis, we will explore how to learn from multiple graph classification tasks to improve the generalization ability of classifier models (Chapter 8).

Complex Task Graph Stream Classification: It is worth noting that in Chapter 7 and Chapter 8 we mainly focus on how to perform graph classification with complex tasks on static datasets. However, these methods can be easily extended to streaming scenarios. Specifically, we can easily modify the framework used for semi-supervised graph stream classification (Section 5.2 and Fig 5.2) or for imbalanced graph stream classification (Section 6.2 and Fig 6.1), so as to enable cost-sensitive or multi-task graph stream classification, with the methods proposed in Chapter 7 and Chapter 8 being as plug-ins (components) for local graph chunks.

Chapter 7

Cost-sensitive Learning for Large Scale Graph Classification

7.1 Introduction

When considering complex task senarios, existing graph classification methods [40, 75, 101, 120, 126, 144, 168] may suffer two fundamental issues, *i.e.*, ineffective for cost-sensitive classification and inefficient for large graphs.

7.1.1 Cost-Sensitive Graph Classification

For graph classification, all existing methods assume, explicitly or implicitly, that misclassifying a positive graph incurs an equal amount of cost/risk to the misclassification of a negative graph, *i.e.*, all misclassifications have the same cost (In this chapter, positive class and minority class are equivalent, and they both denote the class with the highest misclassification cost). The induced decision rules are commonly referred to as *cost-insensitive*. In real-life graph applications, the equal-cost assumption is mostly invalid (or at least too strong). Some examples are given as follows.

Biological Domains: In structure based medical diagnose [29, 131], chemical

compounds active against cancer are very rare and are expected to be carefully monitored and identified. A false negative identification (*i.e.* predicting an active compound to be inactive) has a much more severe consequence (*i.e.* a higher cost) than a false positive identification (predicting an inactive compound to be active). Therefore, a false negative and a false positive are inherently different and a false negative prediction may result in the delay and wrong diagnose, leading to severe complications (or extra costs) at a later stage.

Cyber Security Domains: In intrusion detection systems, each traffic flow can be represented as a graph by presenting traffic destinations (such as IP addresses and ports) as nodes. Malicious traffics may impose threat or damage to computer servers, leading to severe security issues in our social life, such as private information leak or internet breakdown. Therefore, misclassification of a malicious traffic (graph) would have a much higher economic and social costs in terms of its potential impacts.

Motivated by its significance in practice, cost-sensitive learning has established itself as an active topic in data mining and machine learning areas [1, 9, 32, 34, 87, 91, 92, 135, 156, 160, 167] in the last decade. Common solutions to cost-sensitive problem includes sampling [156], decision tree modelling [86, 160], boosting [87, 92], and SVM adaptations [91, 121, 135]. However, all these methods are only dedicated to generic datasets with feature-vector representation, whereas graphs do not have features immediately available and only contain nodes and their dependency structure information. Simply enumerating subgraph structures as features is clearly a suboptimal solution for cost-sensitive learning, mainly because substructure space is potentially infinite and we need a good strategy to find high quality features to help avoid misclassifications on positive classes.

Recently, an igBoost [109] algorithm has been proposed to handle imbalanced graph datasets. The igBoost approach, extended from a standard cost-insensitive graph classification algorithm gBoost [120], assigns proper weight values to different classes by taking data imbalance into consideration, so the algorithm is potentially useful to tackle the cost-sensitive learning problem for graphs. However, the loss function defined in igBoost is not cost-sensitive but only aims to minimize the misclassification errors. As a result, if the training data are separable [91], the algorithm will have limited power to enforce the cost-sensitivity learning be-

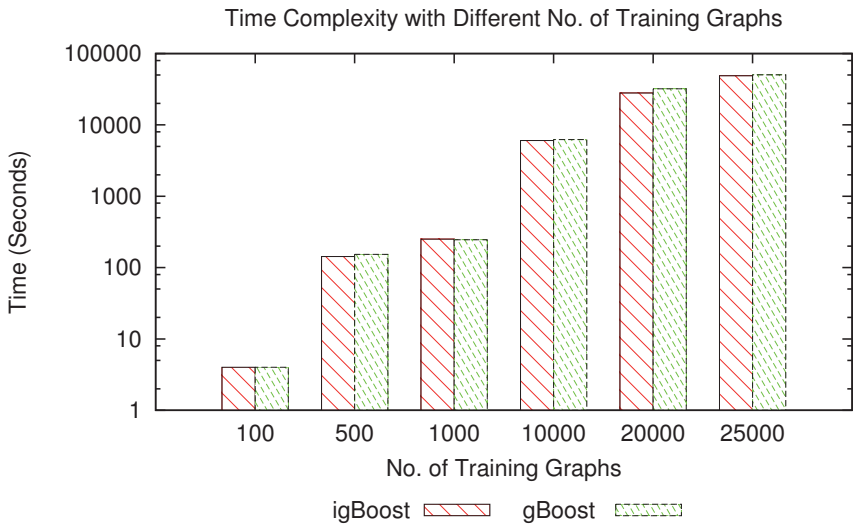


Figure 7.1: Training time *w.r.t.* different number of graphs on NCI-1 dataset for gBoost [120] and igBoost algorithm [109]. Runtime of existing graph classification algorithms exponentially grows *w.r.t.* the increase of the training set size.

cause it only tries to separate training samples without using costs associated to different classes to tune the decisions for minimum costs. From a statistical point of view, the minimum risk could be achieved by following Bayes decision rules to predict graph samples. The objective function in [109] is non-optimal because it simply employs some heuristic schemes, rather than implements the Bayes decision rules to minimize the conditional risk for cost-sensitive setting. In other words, the current boosting style algorithms are not targeting cost-sensitive learning problems for graph data.

7.1.2 Fast Training for Large Scale Graphs

Another challenge of existing graph classification algorithms is that they are only designed for small size graph datasets and are very inefficient to scale up to large size graph datasets. Taking existing boosting-based graph classification algorithms [120] as examples, a boosting algorithm iteratively selects the most discriminative subgraphs from the graph dataset and then solves a linear programming problem for graph classification. In practice, although one may use an

appropriate support value in the first step to find subgraphs, by using subgraph mining based algorithm such as gSpan [151], the linear programming solving in each iteration in the second step is a very time-consuming process, which prevents the algorithms from scaling up to large scale graph datasets.

In Fig. 7.1, we report the runtime of boosting-based graph classification algorithms with respect to different numbers of training graphs. For a small number of graphs, *e.g.* 100 to 1000, both gBoost [120] and igBoost [109] are relatively efficient (requiring 5 to 300 seconds for training). However, when the number of training graphs is considerably large (25,000 graphs or more), the training time for both gBoost and igBoost increase dramatically (about 50,000 seconds), and requires over 13 hours to complete the training task. As big data applications [148] are becoming increasingly popular for different domains and result in graph datasets with large volumes, finding effective boosting algorithms for large scale graph datasets is highly desired.

If we consider both cost-sensitive learning and large scale graph classification as a whole, the following issues should be taken into consideration to ensure the efficiency and the effectiveness of the algorithm:

1. *Cost-Sensitive Subgraph Selection:* In a cost-sensitive setting, we are given a cost matrix representing misclassification costs. To ensure minimum costs for graph classification, we should take cost of individual samples into consideration to cost-sensitively select discriminative subgraphs. A cost-sensitive subgraph exploration process is, therefore, essential, but has not been addressed by existing research.

2. *Model Learning for Cost-Sensitivity:* For existing boosting-based graph classification algorithms, they all have non-optimal loss function, and therefore have limited capability to handle cost-sensitive problems. Alternatively, we should employ a proper loss function, which not only implements the cost-sensitive Bayes decision rule, but also approximates the Bayes risk. By doing this, the induced model will have maximum power for cost-sensitive graph classification.

3. *Fast Training on Large-scale Graphs:* All boosting algorithms are difficult to scale to large graphs because the optimization procedures involved in

each iteration needs to resolve a large scale linear programming problem, which is typically time-consuming. We need new optimization techniques to enable fast training on large scale graph datasets.

Motivated by the above observations, we report in this chapter, CogBoost, a fast cost-sensitive learning algorithm for graph classification. Instead of simply assigning weights to different classes, we derive a loss function which implements the Bayes decision rule and guarantees minimum risk for prediction. To identify discriminative subgraphs for cost-sensitive graph classification, we consider individual cost of each graph sample, which is completely model driven, *i.e.*, we progressively select the most informative subgraphs based on current learned model, and the newly selected subgraph is added to current feature set to refine the classifier model. These two steps are mutually beneficial to each other. To enable fast training on large graph datasets, an advanced optimization technique, *cutting plane algorithm*, is derived to solving linear program efficiently. Experiments on real-word large graph datasets demonstrate CogBoost's perofrmance.

The remainder of the chapter is structured as follow. The problem definition and overall framework are discussed in Section 7.2. Section 7.3 reports our CogBoost algorithm for cost-sensitive graph classification. The cutting plane algorithm for fast training is reported in Section 7.4, followed by the time complexity analysis in Section 7.5. The experiments are reported in Section 7.6, and we conclude the chapter in Section 7.7.

7.2 Problem Definition and Overall Framework

Classifier Model for Graphs: A classifier model $f(\cdot)$, which is learned from a set of training graphs $T = \{(G_1, y_1), \cdots, (G_l, y_l)\}$, is a function to map a connected graph G_i from graph space $\mathcal{G}$ ($G_i \in \mathcal{G}$) to the label space $\mathcal{Y} = \{+1, -1\}$. For cost-sensitive learning, the classifier $f(\cdot)$ is required to minimize the expected misclassification cost/risk $R = E_{G_i,y_i}[L(f(G_i), y_i)]$, where $L(f(G_i), y_i)$ is a non-negative loss function with respect to the misclassification cost. A typical loss

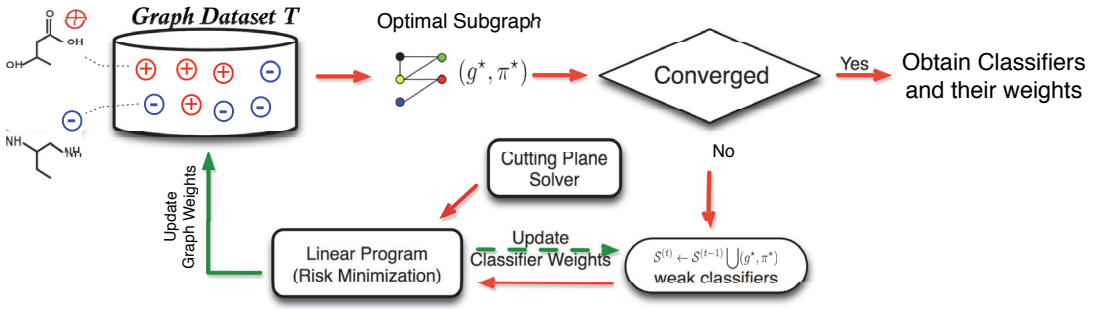


Figure 7.2: The proposed fast cost-sensitive boosting for graph classification framework. In each iteration, CogBoost selects an optimal subgraph featin this chapterure $(g^\star, \pi^\star)$ based on current learned model and weights of training graphs. Then $(g^\star, \pi^\star)$ is added to the current selected set S. Afterwards, CogBoost solves a linear programming problem to achieve cost/risk minimization. To enable fast training on large scale graph datasets, a cutting plane solver is used in this step to improve the algorithm efficiency. After solving the linear program, two sets of weights (green arrows) are updated: (1) weights for training graphs, and (2) weights for weak learners (subgraphs). The feature selection and risk minimization procedures continue until CogBoost converges or reaches the predefined number of iterations.

function is as follows:

$$L(f(G_i), y_i) = \begin{cases} 0 & : \quad f(G_i) = y_i \\ C_1 & : \quad f(G_i) = -1, y_i = 1 \\ C_{-1} & : \quad f(G_i) = 1, y_i = -1 \end{cases} \tag{7.1}$$

The loss function in Eq. (7.1) is extended from the standard 0-1 loss function, *i.e.*, $L(f(\cdot), y) = I(f(\cdot) \neq y)$, where $I(\cdot)$ is an indicator function, $I(a) = 1$ if a holds, or $I(a) = 0$ otherwise. In Eq. (7.1), when $C_1 = C_{-1} = 1$, the function $L(f(G_i), y_i))$ is cost-insensitive and degenerates to the standard 0-1 loss. For cost-sensitive learning, a false negative $(f(G_i) = -1, y_i = 1)$ prediction usually incurs a larger cost than a false positive $(f(G_i) = 1, y_i = -1)$ prediction, *i.e.*, $C_1 > C_{-1}$.

Given a set of training graphs $\mathcal{T} = \{(G_1, y_1), \cdots, (G_l, y_l)\}$, and C_1 and C_{-1} for the cost of misclassification, cost-sensitive graph classification **aims** to build an optimal classification model $f(\cdot)$ from T to minimize the expected misclassi-

fication loss (also known as risk) $R = E_{G_i,y_i}[L(f(G_i), y_i)]$.

7.2.1 Overall Framework

In this chapter, we propose a boosting framework for cost-sensitive graph classification. Our framework (Fig. 7.2) mainly consists of three steps.

1. *Optimal Subgraph Exploration:* One optimal subgraph is selected each step, and the cost-sensitive discriminative subgraph exploration is guided by the model learnt from the previous step. The newly extracted subgraph is added to the most discriminative set S to enhance the learning in the next step.

2. *Risk Minimization and Fast Training:* A linear program is solved to achieve minimum risk based on current selected subgraphs. To enable fast training on large scale graphs, a novel cutting plane algorithm is employed.

3. *Updating Graph Weights for New Iteration:* After the linear program is solved, the weight values for training graphs are updated and the algorithm continues in a new iteration until the whole algorithm converges.

Next we will present our Cogboost algorithm for cost-sensitive graph classification and then propose a cutting plane algorithm to handle large scale graph datasets.

7.3 Cost-Sensitive Learning for Graph Data

For graph classification, boosting [120] has been previously used to identify subgraphs from the training graphs as features. After that, each subgraph is regarded as a decision stump (weak classifier) to build a boosting process:

$$\hbar_{g_k}(G_i; \pi_k) = \pi_k(2I(g_k \subseteq G_i) - 1); \tag{7.2}$$

where $\pi_k \in \mathcal{Y} = \{-1, +1\}$ is a parameter controlling the label of the classifier. In this chapter, the weak classifier is written as $\hbar_{g_k}(G_i)$ for short.

Let $\mathcal{F} = \{g_1, \cdots, g_m\}$ be the full set of subgraphs in T. We can use $\mathcal{F}$ as features to represent each graph G_i into a vector space as $\boldsymbol{x}_i = \{\hbar_{g_1}(G_i), \cdots, \hbar_{g_m}(G_i)\}$, with $\boldsymbol{x}_i^k = \hbar_{g_k}(G_i)$. In the following subsection, we use G_i and $\boldsymbol{x}_i$ interchangeably as they both refer to the same graph.

The prediction rule for a graph G_i is a linear combination of the weak classifiers:

$$f(\boldsymbol{x}_i) = \boldsymbol{w}^{\boldsymbol{T}}\boldsymbol{x}_{\boldsymbol{i}} = \sum_{(g_k,\pi_k)\in\mathcal{F}\times\mathcal{Y}} w_k \hbar_{g_k}(G_i) \tag{7.3}$$

where $\boldsymbol{w} = \{w_k\}_{k=1,\cdots,m}$ is the weight vector for all weak classifiers. The predicted class label of $\boldsymbol{x}_i$ is +1 (positive) if $f(\boldsymbol{x}_i) > 0$ or -1 otherwise.

Similar to SVM, gBoost [120] aims to achieve minimum loss *w.r.t.* a standard hinge loss function $L(f, y) = \lfloor 1 - yf \rfloor_+$, where $\lfloor x \rfloor_+ = \max(x, 0)$. igBoost [109] extends gBoost by assigning larger weight values to graphs in different classes. Both gBoost and igBoost are not optimal when dealing with cost-sensitive cases, because their loss functions do not follow the Bayes decision rules to minimize the expected risk/loss. In this section, we will first present an optimal hinge loss function, and then formalize our algorithm into a boosting paradigm.

7.3.1 Optimal Cost-sensitive Loss Function

A graph classifier $f(\cdot)$ maps a graph G_i to a class label $y_i \in \{-1, 1\}$. Assume graphs and class labels are drawn from probability distribution $P_{\mathcal{G}}(G_i)$ and $P_{\mathcal{Y}}(y_i)$, respectively. Given a non-negative loss function $L(f(G_i), y_i)$, the classifier $f(G_i)$ is optimal if it minimizes the loss/risk $R = E_{G_i,y_i}[L(f(G_i), y_i)]$. Let $\eta = P_{\mathcal{Y}|\mathcal{G}}(1|G_i)$ be the probability for G_i being 1, from a Bayes decision rule point of view, this is equivalent to minimize the conditional risk.

$$E_{\mathcal{Y}|\mathcal{G}}(L(f(G_i), y_i)|G = G_i) \quad = \eta L(f(G_i), 1) + (1 - \eta)L(f(G_i), -1) \tag{7.4}$$

The loss function in Eq. (7.1) is a Bayes consistent loss function [93], *i.e.*, it implements the Bayes decision rule to achieve minimum conditional risk (Eq. (7.4)). This suggests that ideally Eq. (7.1) can be used to design some cost-sensitive al-

gorithms for minimizing conditional risk. However, Eq. (7.1) is extended by a 0-1 loss function. Minimizing the 0-1 loss is computationally expensive because it is not convex. State of the art algorithms usually use surrogate loss functions to approximate the 0-1 loss (*e.g.*, SVM and gBoost [120] employ hinge loss). The hinge loss induced SVM algorithms enforce maximum margins between the support vectors and the hyper-planes, which can achieve good classification performance.

A recent work on SVM [91] theoretically suggests that the standard hinge loss can be extended to be cost-sensitive, by setting the loss function $L(f(G_i), y_i)$ as:

$$L(f(G_i), y_i) = \begin{cases} \lfloor C_1 - C_1 \cdot f(G_i) \rfloor_+ & : \quad y_i = 1 \\ \lfloor 1 + (2C_{-1} - 1) \cdot f(G_i) \rfloor_+ & : \quad y_i = -1 \end{cases} \tag{7.5}$$

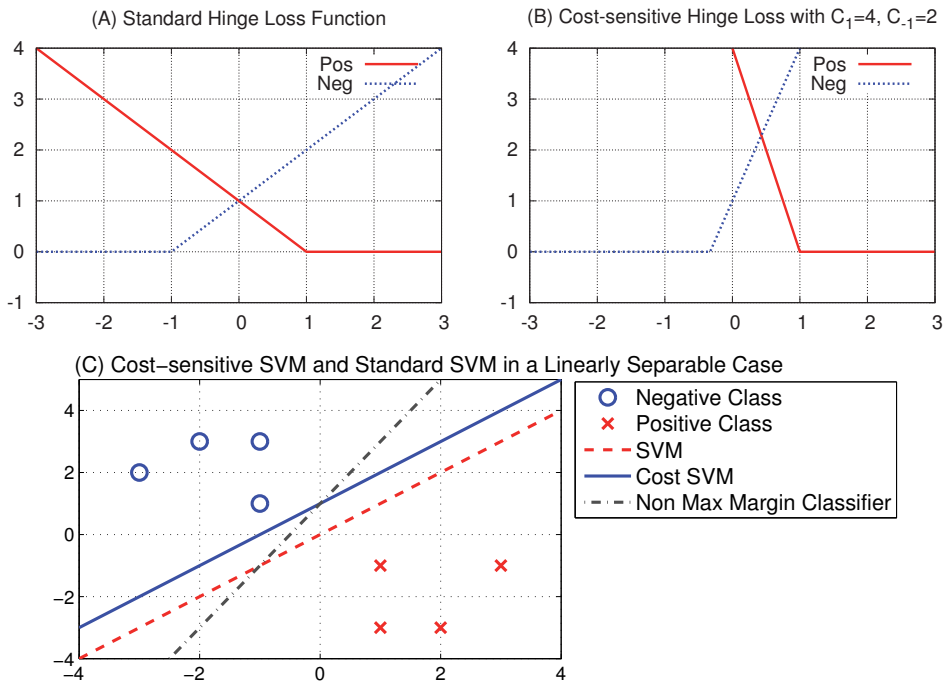


Figure 7.3: Different loss functions and formulations with respect to support vector machines (SVMs): (A) Standard Hinge Loss, (B) Cost-sensitive Hinge Loss with $C_1 = 4$ and $C_{-1} = 2$, and (C) Different SVM formulations with Standard Hinge Loss and Cost-sensitive Hinge Loss (**cf.**[91]).

It is proved in [91] that the new hinge loss function Eq. (7.5) also implements the Bayes decision rule. Additionally, employing Eq. (7.5) also enjoys the merit of maximum margin principle for classification. The standard hinge loss and its cost-sensitive hinge loss is illustrated in Fig. 7.3. They have different explanations with respect to the loss and the margins (distance to the hyperplane from support vectors). Specifically, for standard hinge loss (Fig. 7.3. (A)), the positive and negative class both have equal margins (unit margins); and for cost-sensitive hinge loss (Fig. 7.3. (B)), the negative class has a much smaller margin when the positive class still have a unit margin. As shown in Fig. 7.3. (C), the margins for positive and negative classes are uneven when cost-sensitive hinge loss function Eq. (7.5) is utilized in a SVM formulation.

Note that the loss function employed in igBoost [109] is heuristically adapted from standard hinge loss, which does not necessarily follow the Bayes decision rule. In other words, it is a sub-optimal loss function for cost-sensitive learning. In the following subsection, we will use the cost-sensitive hinge loss function in Eq. (7.5), and re-formulate it into a linear program boosting framework.

7.3.2 Cost-Sensitive Formulation for Graphs

Motivated by the optimal loss function in Eq. (7.5), we formalize our learning task as the following regularized risk minimization problem:

$$\begin{aligned} \min_{\boldsymbol{w}} \quad & \|\boldsymbol{w}\| + \tfrac{C}{l}\{\sum_{\{i|y_i=1\}} L(f(\boldsymbol{x}_i),1) + \sum_{\{j|y_j=-1\}} L(f(\boldsymbol{x}_j),-1) \\ s.t. \quad & \boldsymbol{w} \succeq 0 \end{aligned} \tag{7.6}$$

In Eq.(7.6), we enforce the weight for each subgraph to be positive, *i.e.*, $\boldsymbol{w} \succeq 0$. We also impose 1-norm regularization on $\boldsymbol{w}$ (*i.e.*, $\|\boldsymbol{w}\|$), which will favor sparse solutions with many variables being exactly 0. This strategy is similar to the problem of LASSO for variable shrinking [130]. And we use i and j to index the positive and negative training graphs, respectively. C is a parameter to trade-off the regularization term and loss term. The objective function in Eq.(7.6) can be

reformulated as follows:

$$\begin{aligned} \min_{\boldsymbol{w},\boldsymbol{\xi}} \quad & \|\boldsymbol{w}\| + \tfrac{C}{l}\{C_1 \sum_{\{i|y_i=1\}} \xi_i + \gamma \sum_{\{j|y_j=-1\}} \xi_j\} \\ s.\ t. \quad & f(\boldsymbol{x}_i) \geq 1 - \xi_i,\ y_i = 1 \\ & f(\boldsymbol{x}_j) \leq -\tfrac{1}{\gamma} + \xi_j,\ y_j = -1 \\ & f(\boldsymbol{x}_i) = \sum_{k=1}^{m} w_k \cdot \hbar_{g_k}(G_i), \boldsymbol{w} \succeq 0, \boldsymbol{\xi} \succeq 0, \gamma = 2C_{-1} - 1 \end{aligned} \tag{7.7}$$

In Eq.(7.7), ξ_i and ξ_j are slack variables concerning the loss of misclassifying a positive and a negative graph, respectively. In this case, cost-sensitivity is controlled by C_1 and γ, which impose a smaller margin on negative examples than positive examples (In Fig. 7.3. (B) and (C), for an example with $C_1 = 4$, and $\gamma = 2C_{-1} - 1 = 3$, the margin for negative example is $\frac{1}{\gamma} = \frac{1}{3}$). As suggested in [93], we can set γ as a parameter subject to $1 \leq \gamma \leq C_1$ instead of a fixed value $(2C_{-1} - 1)$ to achieve better classification.

Solving objective function in Eq. (7.7) requires a complete set of subgraph features (*i.e.*, represent G_i as $\boldsymbol{x}_i = \{\hbar_{g_1}(G_i), \cdots, \hbar_{g_m}(G_i)\}$), which are unavailable unless we enumerate the whole subgraph space in advance. In practice, this is likely impossible because the whole subgraph set is very large and possibly infinite. In the following subsection, we will transform this formulation to its Lagrange dual problem and use a boosting algorithm to solve it in an iterative way.

7.3.3 Boosting for Cost-sensitive Learning on Graphs

The Lagrange dual of a problem usually provides additional insights to the original (primal) problem. The dual problem of Eq.(7.7) is [1]

$$\begin{aligned} \max_{\boldsymbol{\mu}} \quad & \textstyle\sum_{i=1}^{l_+} \mu_i + \frac{1}{\gamma} \sum_{j=1}^{l_-} \mu_j \\ s.t. \quad & \sum_{i=1}^{l_+} \mu_i \hbar_{g_k}(G_i) - \sum_{j=1}^{l_-} \mu_j \hbar_{g_k}(G_j) \leq 1, \forall g_k \in \mathcal{F} \\ & 0 \leq \mu_i \leq \tfrac{CC_{-1}}{l}, i = 1, \cdots, l^+ \\ & 0 \leq \mu_j \leq \tfrac{\gamma C}{l}, \quad j = 1, \cdots, l_- \end{aligned} \tag{7.8}$$

[1] The derivation from the primal problem Eq.(7.7) to dual problem Eq.(7.8) is shown in Appendix A.2.

where l_+ and l_- indicate the number of graphs in positive and negative sets ($l = l_+ + l_-$), respectively. While solving the primal problem in Eq.(7.7) returns a vector $\boldsymbol{w}$ indicating the weights of each subgraph, the dual problem in Eq. (7.8) will produce a vector $\boldsymbol{\mu} = \{\mu_i\}_{i=1,\cdots,l}$. Nevertheless, Eq.(7.7) and Eq.(7.8) will generate the same objective values.

Insights of Dual Problem (1) The solution $\{\mu_i\}_{i=1,\cdots,l}$ can be interpreted as the weight values of graphs in order to achieve minimum loss. (2) Each constraint $\sum_{i=1}^{l_+} \mu_i h_{g_k}(G_i) - \sum_{j=1}^{l_-} \mu_j h_{g_k}(G_j) \leq 1$ in Eq. (7.8) indicates a subgraph pattern g_k over the whole training graphs. It provides a natural metric to assess the cost-sensitive discriminative power of a subgraph.

Definition 12. ***Cost-sensitive Discriminative Score:*** *For a subgraph decision stump $\hbar_{g_k}(G_i)$, its cost-sensitive discriminative score over the whole training graphs is:*

$$\Theta(g_k, \pi_k) = \sum_{i=1}^{l_+} \mu_i \hbar_{g_k}(G_i) - \sum_{j=1}^{l_-} \mu_j \hbar_{g_k}(G_j) \tag{7.9}$$

Eq.(7.8) requires discriminative scores for all subgraphs ≤ 1, which latter will serve as an termination condition of our iterative algorithm.

Linear Program Boosting Framework Because we do not have a predefined feature set $\mathcal{F}$ in advance, we cannot solve Eq.(7.7) or Eq.(7.8). Therefore, we propose to use *column generation (CG)* techniques [97] to solve the objective function (Eq. (7.7)). The idea of CG is to begin with an empty feature set S, and iteratively select and add one feature/column to S which violates the constraint in the dual problem (Eq. (7.8)) mostly. After S is updated, CG re-solve the primal problem Eq.(7.7). This procedure continues until no more subgraph violating the constraint in (Eq.(7.8)).

Our cost-sensitive graph boosting framework is illustrated in Algorithm 9. CogBoost iteratively selects the most discriminative subgraph $(g^\star, \pi^\star)$ at each round (step 4). If the current optimal pattern no longer violates the constraint or it has reached the maximum number of iterations T_{max}, the iteration process stops (steps 5-6). Because in the last few iterations, the optimal value only changes subtlety, we add a small value Δ to relax the stopping condition (typically, we use

Algorithm 9 CogBoost Algorithm for Graph Classification

Require:
$\mathcal{T}_l = \{(G_1, y_1), \cdots, (G_l, y_l)\}$: Training Graphs;
T_{max}: Maximum number of iteration;
Ensure:
$f(\boldsymbol{x}_i) = \sum_{(g_k,\pi_k)\in S} w_k^{(t-1)} \hbar_{g_k}(G_i)$: Classifier;
1: $S \leftarrow \emptyset$;
2: $t \leftarrow 0$;
3: **while** true **do**
4: Obtain the most discriminative decision stump $(g^\star, \pi^\star)$; //Algorithm 10;
5: **if** $\Theta(g^\star, \pi^\star) \leq 1 + \Delta$ **or** $t = T_{max}$ **then**
6: **break;**
7: $S \leftarrow S \bigcup (g^\star, \pi^\star)$;
8: Solve Eq. (7.7) based on S to get $\boldsymbol{w}^{(t)}$, and Lagrange multipliers of Eq. (7.8) $\boldsymbol{\mu}^{(\boldsymbol{t})}$;
9: $t \leftarrow t + 1$;
10: **return** $f(\boldsymbol{x}_i) = \sum_{(g_k,\pi_k)\in S} w_k^{(t-1)} \hbar_{g_k}(G_i)$;

$\Delta = 0.01$ in our experiments). In step 8, we solve the linear programming problem based on the selected subgraphs to recalculate two set of weights: (1) $\boldsymbol{w}^{(\boldsymbol{t})}$, the weights for subgraph decision stumps in S; and (2) $\boldsymbol{\mu}^{(\boldsymbol{t})}$, the weights of training graph for optimal subgraph mining in the next round, which can be obtained from the Lagrange multipliers of the primal problem. Once the algorithm converges or the number of maximum iteration is reached, CogBoost returns the final classifier model $f(\boldsymbol{x}_i)$ in step 10.

7.3.4 Cost-sensitive Subgraph Exploration

To learn the classification model, we need to find the most discriminative subgraph which considers each training graph's weight in each step (step 4 in Algorithm 9). The subgraph exploration is completely model driven, *i.e.*, we select a subgraph which violates the constraint in Eq.(7.8) mostly. Based on the definition of discriminative score in Eq.(7.9), we need to perform a weighted subgraph mining over training graphs.

In CogBoost, we employ a Depth-First-Search (DFS) based algorithm gSpan

Algorithm 10 Cost-sensitive Subgraph Exploration

Require:
$\mathcal{T}_l = \{(G_1, y_1), \cdots, (G_l, y_l)\}$: Labeled Graphs;
$\boldsymbol{\mu} = \{\mu_1, \cdots, \mu_l\}$: Weights for labeled graph examples;
min_sup: mininum support for subgraph mining;
Ensure:
$(g^\star, \pi^\star)$: The most discriminative subgraph;
1: $\tau = 0$, $(g^\star, \pi^\star) \leftarrow \emptyset$;
2: **while** Recursively visit the DFS Code Tree in gSpan **do**
3: $g_p \leftarrow$ current visited subgraph in DFS Code Tree;
4: **if** g_p has been examined **or** $sup(g_p) < min\_sup$ **then**
5: **continue**;
6: Compute score $\Theta(g_p, \pi_p)$ for subgraph g_p according Eq.(7.9);
7: **if** $\Theta(g_p, \pi_p) > \tau$ **then**
8: $(g^\star, \pi^\star) \leftarrow (g_p, \pi_p)$; $\tau \leftarrow \Theta(g_p, \pi_p)$;
9: **if** The upperbound of score $\widehat{\Theta}(g_p) > \tau$ **then**
10: Depth-first search the subtree rooted from node g_p;
11: **return** $(g^\star, \pi^\star)$;

[151] to enumerate subgraphs. The key idea of gSpan is that each subgraph has a unique DFS Code, which is defined by its lexicographic order of the discovery time during the search process. Two subgraphs are isomorphism iff they have the same minimum DFS Code. By employing a depth first search strategy on the DFS Code tree (where each node is a subgraph), gSpan can enumerate all frequent subgraphs efficiently. To speed up the enumeration, we further employ a branch-and-bound scheme to prune the search space of DFS Code tree by utilizing an upper bound of discriminative score [120] for each subgraph pattern. Similar upper bound is also used in previous Algorithm 7, and has been described in Theorem 3.

Our subgraph mining algorithm is listed in Algorithm 10. The minimum value τ and optimal subgraph $(g^\star, \pi^\star)$ are initialized in step 1. We prune out duplicated subgraph features or subgraph with low support ($sup(\cdot)$ returns the support of a subgraph) in step 4-5, and compute the discriminative score $\Theta(g_p, \pi_p)$ for g_p in step 6. If $\Theta(g_p, \pi_p)$ is larger than τ, we update the optimal subgraph in step 8. We use an branch-and-bound pruning rule in [120] to prune the search space in steps 9-10. Finally, the optimal subgraph pattern $(g^\star, \pi^\star)$ is returned in step 11.

7.4 Fast Training for Large Scale Graphs

For CogBoost algorithm, it needs to iteratively mine an optimal subgraph (step 4 of Algorithm 9) and solve a linear problem (step 8 of Algorithm 9). To enable fast training for large scale graph datasets, for step 4 we can set a proper support and use some heuristic techniques, such as reusing the search space during the enumeration of subgraphs rather than re-mining subgraph from scratch, just as [120] does. For step 8, in this section, we derive a *cutting plane* algorithm to speed up the training process.

7.4.1 From l-Slacks to 1-Slack Formulation

Eq.(7.7) in step 8 of Algorithm 9 has $l = l_+ + l_-$ slack variables ξ_i and ξ_j, inspired by the techniques used in the SVM formulation [66], we propose to solve it efficiently by reducing the number of slack variables as follows,

$$\begin{aligned} \min_{\boldsymbol{w},\xi} \quad & \|\boldsymbol{w}\| + C\xi \\ s.\,t. \quad \forall \boldsymbol{c} \in \{0,1\}^l, \quad & \tfrac{1}{l}\boldsymbol{w}^T\{C_1 \sum_{y_i=1} c_i \boldsymbol{x}_i - \gamma \sum_{y_j=-1} c_j \boldsymbol{x}_j\} \\ & \geq \tfrac{1}{l}\{C_1 \sum_{y_i=1} c_i + \sum_{y_j=1} c_j\} - \xi, \\ & \boldsymbol{w} \succeq 0, \xi \geq 0. \end{aligned} \tag{7.10}$$

The above formulation can be proved to be equal to Eq.(7.7) [1], with $\xi = \{C_1 \sum_{\{i|y_i=1\}} \xi_i + \gamma \sum_{\{j|y_j=-1\}} \xi_j\}/l$. Note that although Eq.(7.10) has 2^l constraints in total, such a formulation can be solved by cutting plane algorithm in linear time by iteratively selecting a small number of most violated constraints (*Cutting Planes*). This leads to an efficient solution to the optimization, so our algorithm can effectively scale to large datasets.

The dual of l-slack formulation in Eq.(7.8) provides solution $\boldsymbol{\mu}$ which can interpret the graph weights for subgraph mining in the next iteration. To establish the same relationship between the new objective function Eq.(7.10) and the graph

[1] Appendix A.3 proves the equality of Eq.(7.7) and Eq.(7.10).

weights $\boldsymbol{\mu}$, we also refer to its dual problem, which is given as follows[1]:

$$\begin{aligned} \max_{\boldsymbol{\lambda_c}} \quad & \tfrac{c_1}{l}\sum_{\boldsymbol{c}}\lambda_{\boldsymbol{c}}\sum_{i}c_i + \tfrac{1}{l}\sum_{\boldsymbol{c}}\lambda_{\boldsymbol{c}}\sum_{i}c_j \\ s.t. \quad & \tfrac{C_1}{l}\sum_{\boldsymbol{c}}\lambda_{\boldsymbol{c}}\sum_{i}c_i\boldsymbol{x}_i^k - \tfrac{\gamma}{l}\sum_{\boldsymbol{c}}\lambda_{\boldsymbol{c}}\sum_{j}c_j\boldsymbol{x}_j^k \leq 1, \forall k \\ & 0 \leq \sum_{\boldsymbol{c}}\lambda_{\boldsymbol{c}} \leq C. \end{aligned} \tag{7.11}$$

Comparing the the dual problems of Eq.(7.8) and Eq.(7.11), they are identical if:

$$\mu_{i|y_i=1} = \frac{C_1}{l}\sum_{\boldsymbol{c}}\lambda_{\boldsymbol{c}}c_i;\ \ \mu_{j|y_j=-1} = \frac{\gamma}{l}\sum_{\boldsymbol{c}}\lambda_{\boldsymbol{c}}c_j; \tag{7.12}$$

7.4.2 Cutting-plane Algorithm for Fast Training

The basic idea of cutting-plane algorithm is similar to the column generation algorithm, or it can be regarded as a row generation algorithm (each constrain in Eq.(7.11) is a row). Instead of considering all constraints (rows) as a whole, our cutting-plane algorithm considers only the most violated constraint (row) each time. The selected violated constraints form a working set $\mathcal{W}$. It utilizes an iterative procedure to solve the problem. By doing this, the linear program can be solved efficiently.

Our detailed cutting plane algorithm is shown in Algorithm 11. Initially, the working set $\mathcal{W}$ is an empty set in step 1. In each iteration, we solve the optimization problem based on current working set $\mathcal{W}$ in step 3 ($\boldsymbol{w}$=0 and ξ = 0 for the first iteration). Steps 4-6 find the most violated constraint, which is determined by the cost-sensitive loss function in Eq.(7.5). Step 9 adds the most constraint to the working set. The iteration continues until it reaches the convergence (steps 7-8). Also, we add a small constant ϵ (In our experiments, we set $\epsilon = 0.01$ as default value) to enable early termination of iterations.

Our cutting plane algorithm can always return an ϵ-tolerance accurate solution (approximate the solution of Eq. (7.7) very well). It is efficient because each time we solve a linear program in a small working set, the cutting plan algorithm is independent of the sample size. This essentially ensures that our solutions can

[1]The derivation from Eq.(7.10) to Eq.(7.11) is given in Appendix A.4.

Algorithm 11 Cutting plane algorithm for linear problem Eq.(7.7)

Require:

$\{\boldsymbol{x}_1, \cdots, \boldsymbol{x}_l\}$: Training graphs with subgraph representation.

C, C_1, C_{-1}: Parameters for classifier learning.

ϵ: Cutting-plane termination threshold.

Ensure:

$\boldsymbol{w}$: Classifier weights; $\boldsymbol{\mu}$: Graph weights;

Initialize $\mathcal{W} \leftarrow \emptyset$;

while true **do**

Obtain primal and dual solutions $\boldsymbol{w}, \xi, \boldsymbol{\lambda}$ by solving

$$\begin{aligned} \min_{\boldsymbol{w},\boldsymbol{\xi}} \quad & \|\boldsymbol{w}\| + C\xi \\ s.\ t. \quad & \forall \boldsymbol{c} \in \mathcal{W},\ \boldsymbol{w}^T \tfrac{1}{l}\{C_1 \sum_{y_i=+1} c_i \boldsymbol{x}_i - \gamma \sum_{y_j=-1} c_j \boldsymbol{x}_i\} \geq \\ & \tfrac{1}{l}(C_1 \sum_{y_j=1} c_i + \sum_{y_i=-1} c_j) + \xi, \qquad \boldsymbol{w} \succeq 0, \xi > 0. \end{aligned}$$

for $i = 1 \cdots l$ **do**

applying the following rule the find the most constraint variables on positive graphs ($y_i = 1$)

$$c_i = \begin{cases} 1 & : \quad y_i \cdot f(\boldsymbol{x}_i) < 1 \\ 0 & : \quad else \end{cases}$$

applying the following rule to negative graphs($y_j = -1$)

$$c_j = \begin{cases} 1 & : \quad \gamma \cdot y_i \cdot f(\boldsymbol{x}_i) < 1 \\ 0 & : \quad else \end{cases}$$

if $\frac{1}{l}(C_1 \sum_{y_i=1} c_i + \sum_{y_j=-1} c_j) - \boldsymbol{w}^T \frac{1}{l}(C_1 \sum_{y_i=+1} c_i \boldsymbol{x}_i - \gamma \sum_{y_j=-1} c_j \boldsymbol{x}_i) \leq \xi + \epsilon$ **then**

break;

$\mathcal{W} \leftarrow \mathcal{W} \bigcup \boldsymbol{c}$;

update μ_i and μ_j according to Eq. (7.12);

return $\boldsymbol{w}$ and $\boldsymbol{\mu}$;

scale to very large scale graph datasets.

7.5 Time Complexity Analysis: Theoretical Aspect and Practice

The time complexity of CogBoost includes two major components: (1) mining a cost-sensitive discriminative subgraphs $\mathcal{O}(P(l))$ (step 4 of Algorithm 9), and (2) solving a linear program problem $\mathcal{O}(Q(l))$ (step 8 of Algorithm 9), where P and Q are functions for mining subgraph and solving LP problem of size l. For subgraph mining, CogBoost employs a gSpan based algorithm (Algorithm 10) for subgraph enumeration in the first iteration ($\mathcal{O}(P(l))$), and re-uses the search space [120] of the first iteration ($\mathcal{O}(\overline{P}(l))$). Because re-using search space can significantly reduce the mining time, we have $\mathcal{O}(\overline{P}(l)) \ll \mathcal{O}(P(l))$. Suppose the number of iterations of CogBoost (Algorithm 9) is T_{max}, the total time complexity of CogBoost is:

$$\mathcal{O} = \mathcal{O}(P(l)) + (T_{max} - 1)\mathcal{O}(\overline{P}(l)) + T_{max}\mathcal{O}(Q(l)) \tag{7.13}$$

7.5.1 Time complexity of Subgraph Mining

Theoretical Aspect: Intuitively, because the subgraph space is infinitely large, the time complexity for subgraph mining is NP-hard, and $O(P(l))$ for subgraph mining is inevitable for graph classification. Thus all existing subgraph feature selection algorithms for graph classification [75, 120, 129] derive some upper-bounds to prune the search space. In CogBoost, we incorporate the upper-bound in [120] and the support threshold *min_sup* to reduce the subgraph space. It is worth noting that CogBoost can still function properly even though users do not specify the *min_sup* value for subgraph mining. If *min_sup* were not specified, CogBoost can only rely on the upper-bound in [120] to prune the search space.

Practice: In practice, we observe that when the data set is considerable large (*e.g.*, 40,000 chemical compounds or more), setting a support threshold *min_sup* =

5% can significantly speed up the mining progress. However, setting a threshold may incur missing of discriminative subgraphs because some infrequent subgraphs are not checked. Accordingly, we suggest removing the *min_sup* threshold for small graph dataset while setting a proper support for large datasets. The proper support value depends on the domains of applications. For instance, when the average number of nodes and edges of the graph dataset are large, a larger support (5-10%) is preferred. On the other hand, if the average number of nodes and edges are small, a small support (about 1%-3%) is a good choice. For real-world applications, it is useful to first check the statistics of the graph samples before carrying out the graph classification tasks.

7.5.2 Time complexity of LP Solving

Theoretical Aspect: For LP problem, Eq. (7.7) is solvable in polynomial time $O(Q(l)) = \mathcal{O}(l^k)$ with some constant k [94]. In other words, gBoost [120] and igBoost [109] needs polynomial time for this step. By using cutting plane algorithm, the time complexity would be $O(Q(l)) = \mathcal{O}(sl)$, where s is the number of non-zero features in the original problem (please refer to [66] for detailed analysis). Therefore, CogBoost can significantly reduce the runtime when the graph sample size l is large. In Section 7.6, our experiments will demonstrate that the improvement of LP problem solving without using cutting plan algorithm is marginal.

Practice: The cutting plane algorithm uses a working set $\mathcal{W}$ (in Algorithm 11), $\mathcal{W}$ overlaps significantly during two consecutive iterations of Algorithm 9. Therefore, in our implementations, we re-use top 200 most violated constraints in $\mathcal{W}$ in the previous iteration, which can significantly improve the algorithm efficiency. In practice, the classifier weights $\boldsymbol{w}$ in two consecutive iterations may be very close to each other, one can also use the warm-start technique (using $\boldsymbol{w}$ in previous iteration as initial value for linear problem solving) to speed up the learning process.

7.6 Experiments

In this section, we evaluate CogBoost in terms of its average misclassification cost (or average cost) and runtime performance. The *average cost* is calculated by using the total misclassification costs divided by the number of test instances. The lower the average costs, the better the algorithm performance is. The runtime performance is evaluated based on the actual runtime of the algorithm.

7.6.1 Experimental Settings

Two types of real-life datasets, NCI chemical compounds and Twitter graphs, are used in our experiments. Table 3.2 summaries the statistics of the two benchmark datasets.

- **NCI Graph Datasets:** In our experiment, we used the original NCI datasets, which are naturally imbalanced and are ideal benchmarks for testing imbalanced or cost-sensitive classification tasks.
- **Twitter Graphs:** This dataset has been used for simulated graph stream classification in Chapter 6. In this chapter, we aggregate all graphs as one dataset without considering their temporal order.

Baselines We compare our proposed CogBoost algorithm with the following baseline algorithms.

- **gBoost** [120] is a state-of-the-art boosting method, which has demonstrated good performance for graph classification.
- **igBoost** [109] extends gBoost to handle imbalanced graph datasets. The weight of a minority (positive) graph is assigned with a β times higher weight value than a majority (negative) graph.
- **Fre+CSVM** first mines a set of frequent subgraphs (with minimum support 3%) from the entire graph dataset, and selects the top-K most frequent subgraphs as features. Afterwards, each graph dataset is transferred into vector format by checking the existence of selected subgraphs in the

original graph datasets. Finally, the cost-sensitive support vector machine algorithm [9, 135] is applied to the transferred vectors.

- **gSemi+CSVM**[1] employs a gSemi [75] algorithm to mine top-K discriminative subgraphs from the entire graph dataset, and then transfers the original graph database into vectors. Similar to Fre+CSVM, the cost-sensitive SVM algorithm [9] is used to learn a model from the transferred vectors.

To validate the effectiveness of the cutting plane solver in our CogBoost algorithm for large scale graphs, we implement two variants of CogBoost,

- **CogBoost-a:** This variant discards the cutting plane module and solves the linear program of Eq.(7.7) directly. In other words, it uses all l slack variables in total.

- **CogBoost-1:** The CogBoost-1 utilizes the cutting plane module to solve the linear program (Eq.(7.10)) for large scale graphs, *i.e.*, it has only one (*i.e.* 1) slack variable each time.

For each graph dataset, we randomly split it into two subsets. The training set consists of 70% of the graph dataset, and the rest is used as the test set. The results reported in the chapter are based on the average number of repetitions. Note that for gBoost [120] and igBoost [109], the previous studies only validate their performance using a rather small number of graphs (from several hundreds to several thousands graphs), whereas in our experiments, our training data is much larger.

Parameter Settings For fair comparisons, the default misclassification cost for positive graphs is set as $C_1 = 20$ for NCI graphs, which is actually the approximated imbalanced ratio ($\frac{|Neg|}{|Pos|}$) of these graph datasets. For Twitter graphs, a large C_1 will result in that all graphs are classified in one class for all algorithm. To avoid this case, we set the default value $C_1 = 3$. For all experiments, the cost for negative graphs is always set as $C_{-1} = 1$ for all datasets. As suggested in [93],

[1]We encounter an out-of-memory error for gSemi+CSVM algorithm on Twitter graph dataset, because gSemi algorithm [75] needs to do matrix calculation to select subgraphs. Java fails to create such a large "double" matrix (about 100,000*100,000).

we selected the best parameter γ instead of fixing it to $2C_{-1}-1$ for CogBoost algorithm. For igBoost, we set $\beta = C_1$, so positive class graphs have a weight value β times higher than negative graphs. The regularization parameter in our algorithm is C, and $D = 1/v$ for gBoost and igBoost. To make them comparable, we vary C from $\{0.1, 1, 10, 100, 1000, 10000\}$, and v from $\{0.01, 0.2, 0.4, 0.6, 0.8, 1.0\}$. These candidate values are set according to the property of each algorithm. *min_sup* is set to 5% for NCI graphs and 0.5% for Twitter graphs.

Because both igBoost and gBoost require over 10 hours to complete a classification task, it is impractical to select the best parameters for each algorithm on the whole training graphs on each dataset. Therefore, we select the parameters for each algorithm which achieves the minimum misclassification cost over a sample of 5000 training graphs on each dataset. Then we train the classifiers with these selected parameters on the whole training graphs. For Fre+CSVM and gSemi+CSVM, the number of most informative subgrahps K is always equal to T_{max} employed into another boosting algorithm, *i.e.*, we ensure that all algorithms use the same number of features for graph classification.

Unless specified otherwise, other parameters for our algorithm are set as follows: $T_{max} = 50$ and $\epsilon = 0.01$.

All our algorithms are implemented using a Java package MoSS [16, 17] and Matlab toolbox CVX [50, 51]. MoSS [1] provides a framework for frequent subgraph mining, and CVX [2] serves as a module for solving linear programs. JavaBuilder provided by Matlab bridges MoSS and CVX into a united framework. All our experiments are conducted on a cluster node of Red Hat OS with 12 processors (X5690 @3.47GHz) and 48GB memory.

7.6.2 Experimental Results

In this subsection, we evaluate the effectiveness of CogBoost for cost-sensitive learning and fast cutting-plane training in terms of average cost and runtime performance. The experimental results for NCI and Twitter graphs under default parameter settings are illustrated in Fig. 7.4.

[1] http://www.borgelt.net/moss.html
[2] http://cvxr.com/cvx/

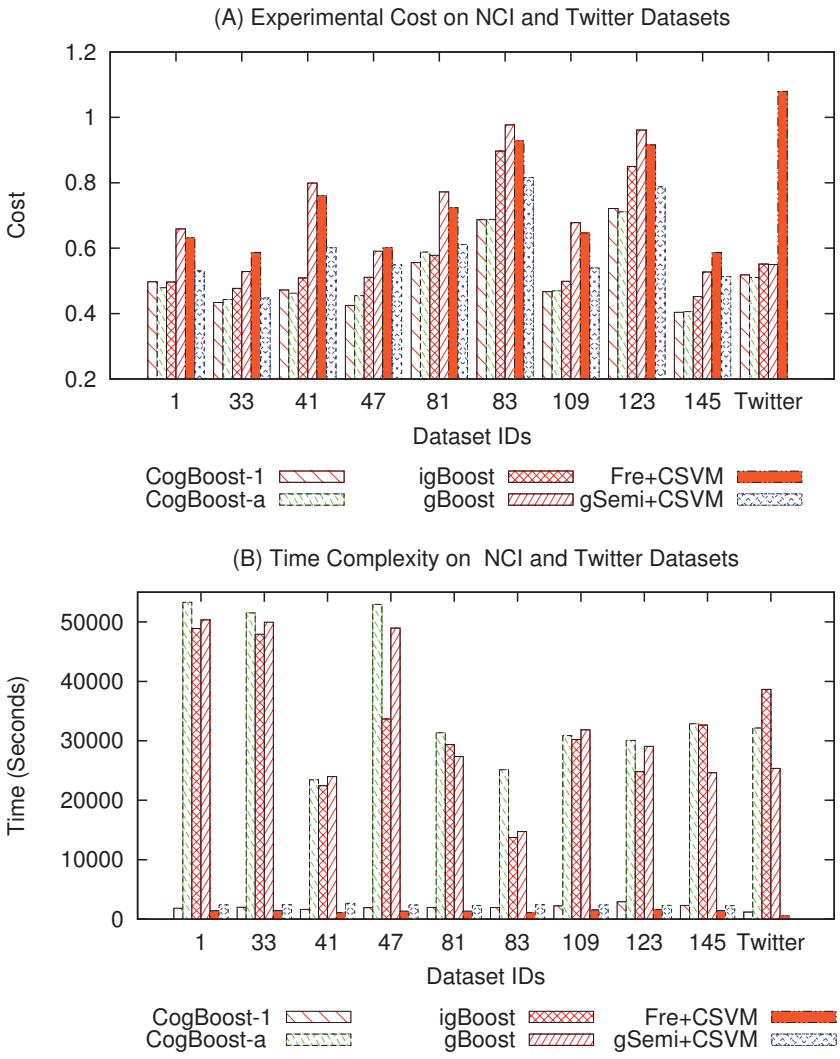


Figure 7.4: Experimental Results. (A) Average cost, (B) Time Complexity.

.

Average Cost For average cost, Fig. 7.4. (A) demonstrates that gBoost has the worst performance on 5 out of 10 datasets (*i.e.* the largest average cost). This is mainly because gBoost is a cost-insensitive algorithm, which considers that all training graphs are equally important in terms of their costs. As a result, gBoost fails to leverage the costs of graph samples to discover subgraph features mostly discriminative for differentiating graphs in the positive class, leading to deteriorated classification performance.

For igBoost, Fre+CSVM, and gSemi+CSVM, all of them have a mechanism to assign weight values to different classes. Fig. 7.4. (A) shows that igBoost outperforms Fre+CSVM and gSemi+CSVM, which is mainly attributed to igBoost's integration of discriminative subgraph selection and classifier learning for graph classification. For Fre+CSVM and gSemi+CSVM, they decompose subgraph selection and classifier learning into two separated steps, without integrating them to gain mutual benefits, *i.e.*, the subgraphs selected by frequency and gSemi score [75] may not be a good feature set for SVM learning. As a result, Fre+CSVM and gSemi+CSVM are inferior to igBoost. This is, in fact, consistent with previous studies [120], which confirmed that gBoost outperforms a frequent subgraph based algorithm (mine frequent subgraphs as features and then apply SVMs).

The experimental results in Fig. 7.4 (A) show that CogBoost outperforms igBoost. This is because the loss function in igBoost is not a cost-sensitive loss function, but heuristically adapted from the hinge loss function (*i.e.*, simply assigning different weights to different classes). Therefore, it does not necessarily implement the Bayes decision rule and cannot guarantee minimum conditional risk.

In contrast, CogBoost-1 and CogBoost-a adopt an optimal cost-sensitive loss function which implements the Bayes decision rule to achieve minimum cost. Evidently, both CogBoost-1 and CogBoost-a outperform gBoost over all graph datasets with significant performance gain, and outperforms igBoost for most graph datasets.

Runtime Performance The algorithm runtime in Fig. 7.4 (B) shows that gBoost, igBoost, and CogBoost-a all require an order of magnitude more time over CogBoost-1, Fre+CSVM, and gSemi+CSVM. For instance, CogBoost-1 only

needs about 1,846 seconds on NCI-1 dataset whereas all other boosting algorithms take about over 50,000 seconds to complete the task. Overall, CogBoost-1 is 25 times faster than all other boosting algorithms. This result validates that reformulating our problem from Eq.(7.7) to a new problem (Eq. (7.10)) and using cutting plane algorithm to solve it can efficiently speed up the problem solving.

Note that Fre+CSVM and gSemi+CSVM have a little less runtime than CogBoost-1, this is because they only solve the SVM formulation (quadratic program) once, while our algorithm iteratively solves a linear program in each iterations.

Comparing the runtime of NCI and Twitter datasets, we found that although twitter dataset is significant larger than NCI, the time consumption for NCI and Twitter does not differ much. This is because the average number of nodes and edges for twitter dataset is much smaller than NCI, making it much efficient for subgraph mining for all algorithms.

Runtime Consumption Details for Boosting algorithms To better understand why CogBoost is more efficient than its peers, we investigate detailed runtime consumption in each step for boosting algorithms. These boosting methods all consist of two key steps in each iteration: *i.e.*, optimal subgraph mining and linear problem solving. Accordingly, we report the algorithm runtime in each iteration in Fig. 7.5, and report average time consumption in Table 7.1.

Table 7.1 and Fig. 7.5 show that, on average, subgraph mining can be done in less than 20 seconds for all algorithms. At the first iteration, the subgraph mining step requires a significant amount of runtime. This is because gSpan needs to generate the search tree until the pruning condition is satisfied. Creating a new node is time consuming, because the list of embeddings is updated, and the minimality of the DFS code has to be checked (See [120] for more details). In the latter iterations, the time consumption for this process can be reduced greatly because the searching space can be reused. The node creation is necessary only if it were not created in previous iterations. As a result, we can observe that the algorithm is more efficient in the latter iterations.

As for the LP optimization steps, gBoost, igBoost, and CogBoost-a all consume much more time than CogBoost-1. This is because they all need to solve a

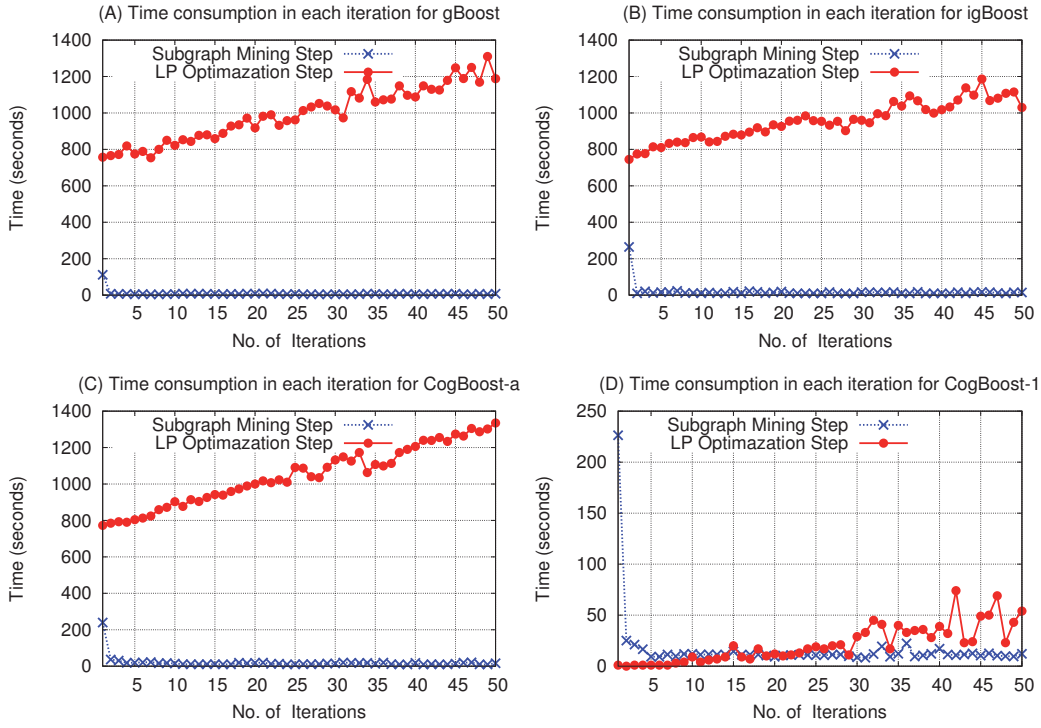


Figure 7.5: Runtime performance in each iterations. Runtime consumption for (A) gBoost, (B) igBoost, (C) CogBoost-a, and (D) CogBoost-1.

linear problem (similar to Eq. (7.7) for gBoost and igBoost) with l slack variables $\xi_{i|i=1,\cdots,l}$ in each iteration (l is the total number of graph examples). When l is large, it will require a very large amount of time to solve the linear problem. In contrast, CogBoost-1 solves the linear problem (Eq. (7.10)) with only one single slack variable ξ by using cutting plane algorithms (Algorithm 11). This new formulation can greatly reduce the time required for linear problem solving.

The results in Table 7.1 and Fig. 7.5 show that CogBoost-1 only needs about 21.58 seconds for one iteration while all other algorithms require about 1,000 seconds to complete this step. Because LP optimization step is the most computationally intensive step for boosting algorithms, CogBoost-1 is much faster than all existing boosting algorithms for graph classification.

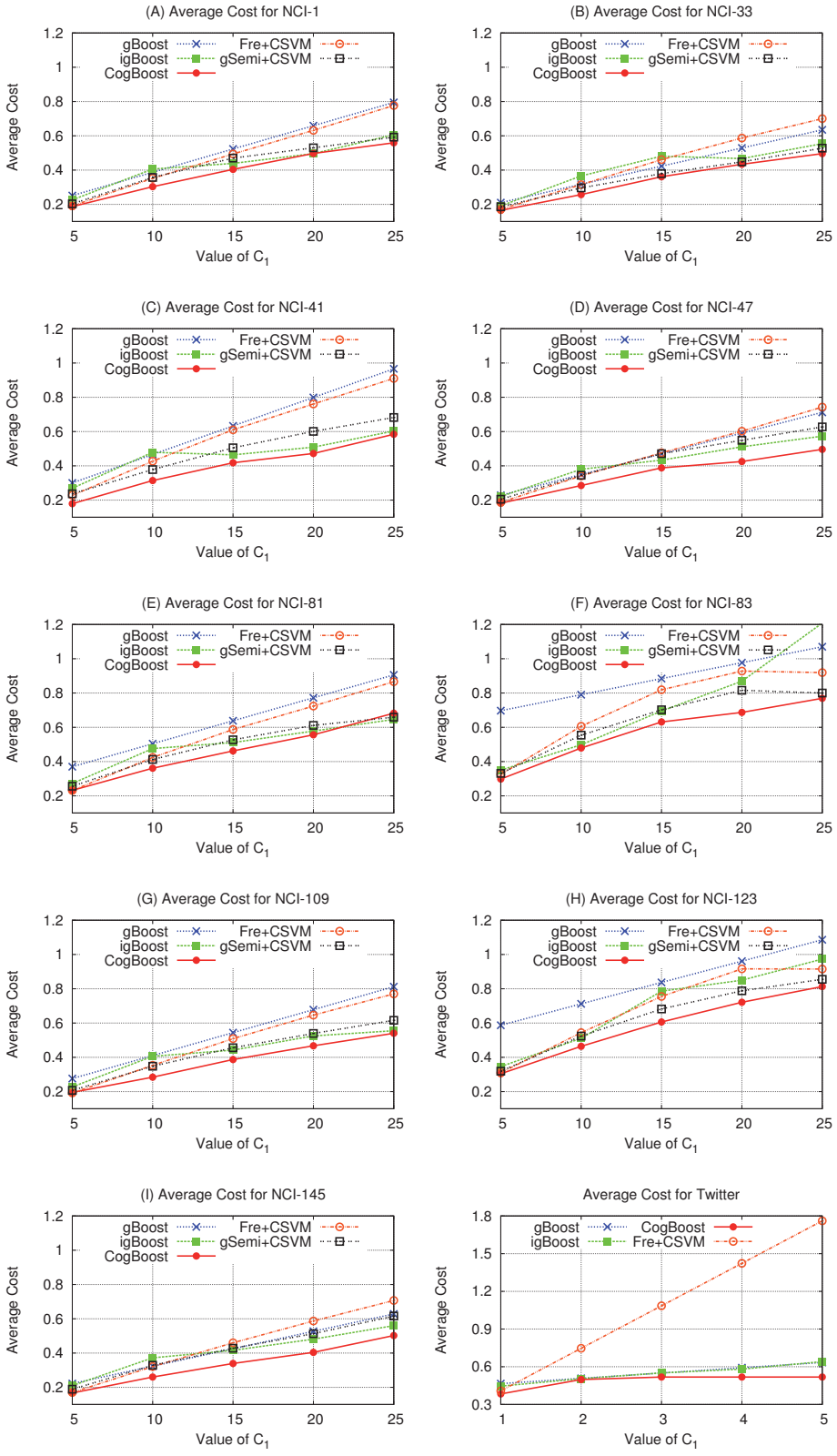


Figure 7.6: Average Cost with respect to different C_1 value

Table 7.1: Average Time Consumption in Each Iteration (Seconds)

	gBoost	igBoost	CogBoost-a	CogBoost-l
Subgraph Mining	8.24	18.24	19.39	16.33
LP Optimazation	993.42	954.66	1046.12	21.58

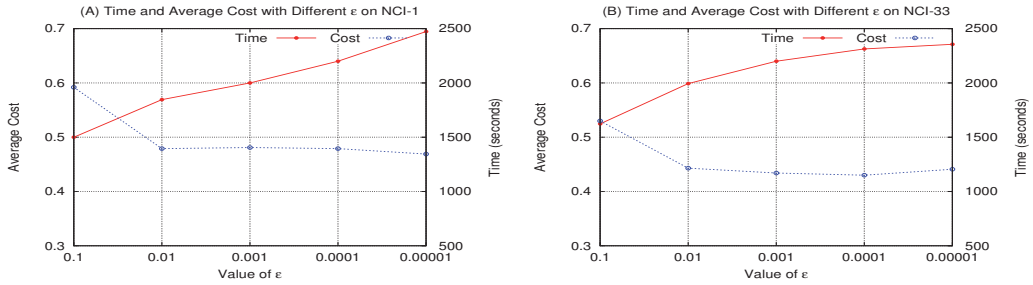


Figure 7.7: Average cost (left y-axis) and algorithm runtime (right y-axis) with respect to different ϵ values ($x-$axis). (A) NCI-1, and (B) NCI-33

7.6.2.1 Performance *w.r.t.* different ϵ values

Comparison of CogBoost-1 and CogBoost-a The experiment results in Fig. 7.4 show that CogBoost-1 can always achieve similar (or very close) classification performance as CogBoost-a. This is because CogBoost-1 can always return an ϵ-tolerance solution to CogBoost-a in each iteration. This, in fact, empirically proves the correctness of our CogBoost-1 formulation. Because CogBoost-1 can achieve accurate solutions to CogBoost-a but with much less runtime consumption than CogBoost-a, in the following experiments, we will report CogBoost-1 (termed as **CogBoost**) for comparison with other algorithms.

Meanwhile, we will mainly focus on the classification performance for cost-sensitive learning because the time complexity is relatively stable for each graph dataset with the same number of training graphs.

7.6.2.2 Performance *w.r.t.* different cost values

In order to study the algorithm performance *w.r.t.* different cost values, we vary the C_1 values from 5 to 25 for NCI graphs and 1 to 5 for Twitter graphs and report the algorithm performance in Fig. 7.6 where the x-axis in each subfigure

shows the C_1 values and the y-axis show the average costs of different methods.

Fig. 7.6 shows that with the increasing of C_1 value, the average costs of all all algorithms will increase. This is because the increasing of C_1 value results in a higher misclassification cost of positive graphs. Comparing to gBoost, igBoost achieves less average cost on most graph datasets. This is mainly attributed to the uneven weight assignment scheme for different classes adopted in igBoost, which allows igBoost to deal with the cost-sensitive problem to some extend.

For all datasets, CogBoost achieves minimum average cost with respect to different C_1 values. This is attributed to the optimal hinge loss function employed in CogBoost, which implements the Bayes decision rules and forces CogBoost to favor high cost samples in order to minimize the misclassification costs. This result is actually consistent with results from a previous study [91], which addresses cost-sensitive support vector machine algorithm for vector data, while CogBoost is a boosting algorithm for graph classification.

In CogBoost, the parameter ϵ controls CogBoost's solutions in solving the cutting plane algorithm (Algorithm 11). In order to validate ϵ's impact on the algorithm performance, we vary ϵ values and report CogBoost's performance in Fig. 7.7.

Fig. 7.7 shows that for large ϵ values (*e.g.* $\epsilon = 0.1$ on NCI-1 dataset), the corresponding average cost is also large. This is because a large ϵ value returns a solution far away from the optimal solution and results in poor performance for CogBoost. As ϵ continuously decreases (from 0.1 to 0.00001), the average cost on both NCI-1 and NCI-33 datasets decrease. This is because with a small ϵ value, CogBoost can return accurate solution for classification. However, the runtime consumption for smaller ϵ values will also increase because more iterations are required in the cutting algorithm. Our empirical results suggest that a moderate value (such as ϵ=0.01) has a good tradeoff between time complexity and average cost. So we set ϵ=0.01 as a default value in our experiments.

In summary, our experiments suggest that cost-sensitive graph classification is a much more complicated problem than traditional cost-sensitive learning, mainly because graph classification heavily relies on subgraph feature exploration. Simply converting a graph dataset into a vector representation, by using frequent subgraph features, and then applying cost-sensitive learning (like Fre+CSVM does)

is far from optimal. Indeed, subgraph features play vital role for graph classification. By using a cost-sensitive subgraph exploration process and a cost-sensitive loss function, CogBoost demonstrates its superb performance for cost-sensitive graph classification.

7.7 Conclusion

In this chapter, we formulated a cost-sensitive graph classification problem for large scale graph datasets. We argued that many real-world applications involve data with dependency structures and the cost of misclassifying samples in different classes is inherently different. This problem motivates us to consider effective graph classification algorithms with cost-sensitive capability and being suitable for large scale graph datasets. To solve the problem, we proposed a fast boosting algorithm, CogBoost, which embeds the costs into the subgraph exploration and the learning process. The boosting procedure utilizes an optimal loss function to minimize the misclassification costs by implementing the Bayes decision rule. To enable fast training on large scale graphs, a cutting plane formulation is derived so that the linear problem can be solved efficiently in each iteration. Experimental results on large real-life graph datasets validate our designs.

Chapter 8

Joint Structure Feature Exploration and Regularization for Multi-Task Graph Classification

8.1 Introduction

For complex task graph classification, there is usually the case that each graph classification task only has a limited number of labeled data, yet multiple similar graph classification tasks co-exist.

In practice, if the sets of tasks are collected from similar or very close domains, then multi-task learning can be an effective scheme to improve the classification performance.

Two motivated multi-task graph classification examples are given as follows:

Functional Brain Analysis aims to map human brain as a network (or a graph) to model relationships between diseases and functions of brain regions [26]. In

order to carry out a specific learning task, such as diagnosing Attention Deficit Hyperactivity Disorder (ADHD) [128], each object has to go through functional magnetic resonance imaging (fMRI) and intensive data preprocessing to collect training data. This severely limits each task to maximum of only a couple of hundred objects. On the other hand, institutions may have data collected for different but relevant learning tasks, such as Gender [136] or Alzheimer's disease study. The limited samples for each individual task and the commonality between tasks raise an interesting question as to whether multiple brain function classification tasks can be combined to learn a multi-task model for maximum performance gain.

Chemical Compound Categorization is important in biomedical research for testing whether a chemical compound is active to a specific cancer, such as melanoma. For melanoma cancer, determining activities of a molecule is expensive as it requires time, efforts, and expensive resources [7] to conduct biological assay. In reality, some similar bioassay tasks [1], such as an anti-cancer test for prostate, may be available. As the graph data for different types of cancer may share common substructures, learning multiple related tasks together may potentially help improve the generalization performance of each single task.

Instead of treating each task as a single-task graph classification (STG) problem, we formulate a multi-task graph classification (MTG) problem which intends to simultaneously handle multiple relevant graph classification tasks.

When solving MTG problems, one simple approach is to treat each task independently and train an STG algorithm (*e.g.*, gBoost [120]) for each task. The result from this approach is, however, far from optimal. This is because (1) the insufficient number of labeled graphs for each task makes learning algorithm difficult to comprehend graph structures for finding effective subgraphs to train classification models. From a machine learning perspective, the limited labeled graphs are biased samples obtained from a sampling process of a large collection of graph examples. A subgraph feature discovered from these graph samples may be also biased, and has limited capability to differentiate test graphs; and (2) a learning model trained from a small number of labeled graphs tends to overfit

[1] https://pubchem.ncbi.nlm.nih.gov/

the training samples and results in poor performance on test data.

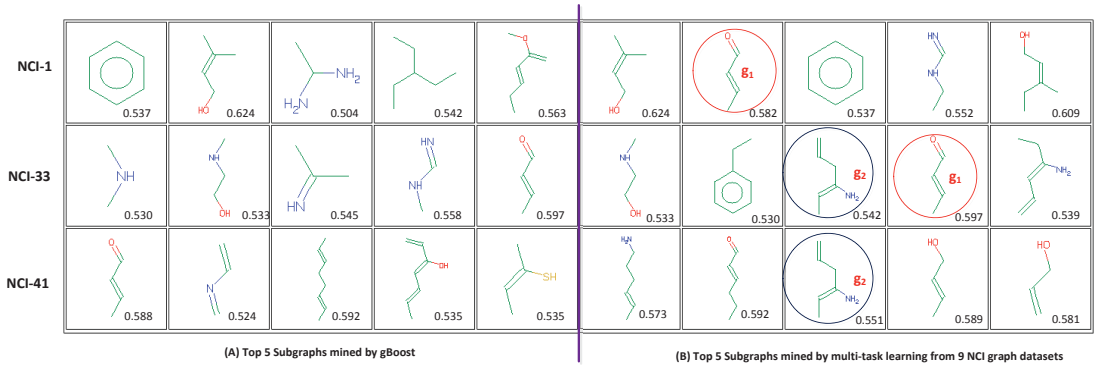


Figure 8.1: The comparisons of the Top 5 most discriminative subgraphs for each graph classification task, mined by (A) gBoost [120], or by (B) multi-task learning (using 50 training graphs for each task). The numeric value next to each subgraph indicates the classification accuracy on test graphs using this single subgraph as a feature (*i.e.,* an indicator of the classification quality of this subgraph). Multi-task learning in (B) favors subgraphs which also have high discriminative powers across all tasks. For instance, the circled g_1 is ranked at the second place for NCI-1 on the training data because it also has a high score on NCI-33. g_1's score 0.582 in NCI-1 outperforms 4 out of top 5 features selected by gBoost, but it was not discovered by gBoost as the Top 5 discriminative subgraphs (so its importance is under-evaluated when learning from NCI-1 task alone).

A second approach to solve MTG problems is to first mine frequent subgraphs [151] as features to transfer graphs into vector format, and then employ state-of-the-art multi-task learning (MTL) algorithms [8, 35] to the vectors. This method is still suboptimal, mainly because subgraph feature exploration process is not tied to the learning tasks (because there are multiple learning tasks). With suboptimal features, it can hardly achieve good classification performance.

Instead of treating MTG as multiple independent learning tasks, we advocate a multi-task driven subgraph (MTDS) mining to explore low dimensional discriminative subgraph features for training all classification models simultaneously. By integrating MTDS based feature selection into our multi-task graph classification objective function, we are expected to allow knowledge to be shared across all tasks for better subgraph validation and model regularization. The niche of our multi-task subgraph feature exploration and multi-task graph classification stems from the following key observations:

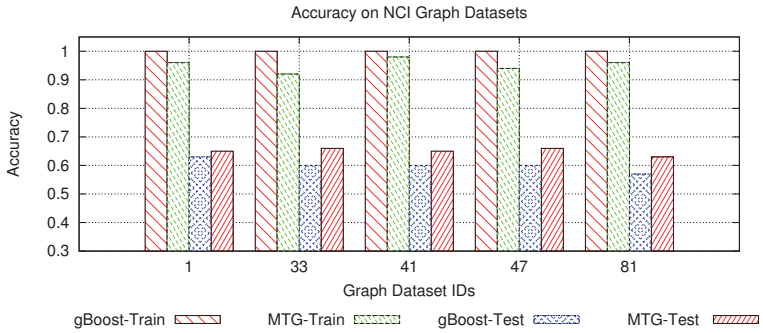


Figure 8.2: Accuracy comparisons on training and test graphs with 50 training graphs for each task. STG algorithms (*i.e.*, gBoost) can easily fit the training graphs perfectly (with 100% accuracy) but its performance on test graphs is much worse (about 0.6 or less). MTG algorithms can avoid overfitting because graphs from relevant tasks are used for regularization.

Multi-Task Shared Subgraphs: Because multiple graph classification tasks are relevant to each other, some common discriminative subgraph features may exist across different tasks. A significant subgraph on one task may also have a high discriminative score on the other task. For instance, in Fig. 8.1, g_1 is a common subgraph of tasks NCI-1 and NCI-33. However, when performing subgraph selection on NCI-1 task only, g_1 will be missed by a STG algorithm (*e.g.*, gBoost [120]). In this context, combining NCI-1 and NCI-33 as a multi-task problem clearly helps NCI-1 task find a better discriminative subgraph for classification.

Implicit Evaluation Set and Better Regularization: To avoid overfitting incurred by insufficient training samples, machine learning algorithms usually validate their models on some evaluation sets before testing or incorporating regularization terms for model learning. With a limited number of training graphs, a good subgraph explored from a single task has a very high risk of overfitting the training data (as shown in Fig. 8.2). By unifying multiple tasks as one objective function, one can use other tasks as implicit evaluation sets for each task. So an MTG objective function can help prune subgraph features which are only good in biased training data of an individual task but are not promising for other tasks. This is, in fact, particularly useful for graph classification mainly because relevant graph samples from other tasks can be considered as implicit evaluation set which

can help validate the sub-graph mining process to achieve a better regularization (detailed in Fig. 8.2).

Motivated by the above observations, we propose a multi-task graph classification algorithm, which iteratively selects the most discriminative subgraphs to achieve minimum regularized loss for all tasks. The multi-task graph classification is achieved by combining subgraph selection and model learning into an iterative process, which mutually benefits subgraph exploration and multi-task learning: for subgraph selection, we can select some low dimensional subgraphs shared among all tasks by employing the MTDS selection scheme; and for multi-task learning, all tasks are jointly regularized to achieve better classification performance.

It is worth noting that our approach is not a simple extension of existing multi-task learning algorithms [8, 35] to the graph classification domain. This is mainly because subgraph feature exploration is the inherent challenge for graph classification, whereas existing multi-task learning methods fall short in exploring structure features across multiple tasks for learning. Secondly, our method closes the loop from multi-task driven subgraph (MTDS) mining to joint regularization across multiple graph classification tasks, which has not been addressed by either graph classification or multi-task learning communities. Last, but not the least, our research proposes effective pruning strategy to reduce the search space for graph feature exploration, and our solution is evaluated by using different loss functions and regularization norms. So our design can serve as a reference for future research in the area.

The main contributions of this chapter can be summarized as follows:

- To the best of our knowledge, this is the first work to handle multi-task learning for graph data. We propose an algorithm with theoretically proved convergence to jointly regularize multiple tasks to exploit discriminative subgraphs for multi-task graph classification.
- We generalize the *column generation technique* [120] to multi-task graph classification setting. Any differentiable loss function such as least squares, exponential, and logistic loss functions can be used in our algorithm.
- We propose to integrate two sparsity-inducing regularization norms, ℓ_1-

norm and $\ell_{2,1}$-norm, for multi-task learning for graph data.

- We derive two branch-and-bound rules to prune search space for multi-task driven subgraph mining.

The remainder of the chapter is structured as follows. Problem definitions and preliminary for graph classification are described in Section 8.2. Section 8.3 reports the proposed algorithm for multi-task graph classification. Experimental results are presented in Section 8.4, and the relation to other algorithm is discussed in Section 8.5. We conclude the chapter in Section 8.6.

8.2 Problem Definition & Preliminaries

Multi-task Graph Classification: Given a set of graph classification tasks, where each task $t \in \{1, 2, \cdots, T\}$ has a set of labeled graphs $\{(G_{t,1}, y_{t,1}), \cdots, (G_{t,n_t}, y_{t,n_t})\}$, we use $G_{t,i} \in \mathcal{G}$ ($\mathcal{G}$ is the graph space) to denote the i^{th} graph in task t, and $G_{t,i}$'s class label is $y_{t,i} \in \mathcal{Y} = \{+1, -1\}$. Multi-task graph classification **aims** to learn T functions (classification models) $f_t : \mathcal{G} \to \mathcal{Y}, t \in [1, T]$, which have best classification performance on the unseen graphs over *all* tasks.

8.2.1 Preliminaries

Single Task Graph Classification. To support graph classification, state-of-the-art algorithms [40, 120] use a set of subgraphs from the training graphs as features. After that, each subgraph g_k can map a given graph $G_{t,i}$ into the class label space $\mathcal{Y} = \{+1, -1\}$:

$$\hbar_{g_k}(G_{t,i}) = 2I(g_k \subseteq G_{t,i}) - 1; \tag{8.1}$$

Here $I(a) = 1$ if a holds, and 0 otherwise.

Let $\mathcal{F} = \{g_1, \cdots, g_m\}$ be the full set of subgraphs in $\mathcal{G}$. We can use $\mathcal{F}$ as features to represent each graph $G_{t,i}$ into a vector space as $\boldsymbol{x}_{t,i} = \{\hbar_{g_1}(G_{t,i}), \cdots, \hbar_{g_m}(G_{t,i})\}$, with $\boldsymbol{x}^k_{t,i} = \hbar_{g_k}(G_{t,i})$. In the following subsection, $G_{t,i}$ and $\boldsymbol{x}_{t,i}$ are

used interchangeably as they both refer to the same graph (*i.e.*, the i-th graph in task t). Given the full subgraphs $\mathcal{F}$, the prediction function for the task t is a linear classifier:

$$f_t(\boldsymbol{x}_{t,i}) = \boldsymbol{x_{t,i}} \cdot \boldsymbol{w_t} + b_t = \sum_{g_k \in \mathcal{F}} w_{t,k} \hbar_{g_k}(G_{t,i}) + b_t \tag{8.2}$$

where $\boldsymbol{w}_t = [\boldsymbol{w}_{t,1}, \cdots, \boldsymbol{w}_{t,m}]'$ is the weight vector for all features for the task t, and b_t is the bias of the model. The predicted class of $\boldsymbol{x}_{t,i}$ is +1 if $f_t(\boldsymbol{x}_{t,i}) > 0$ or -1 otherwise.

For single task graph classification, state-of-the-art algorithm gBoost [120] formulate its objective function as a linear programming problem, then integrates the discriminative subgraph mining into the model learning process via column generation techniques.

8.3 Multi-task Graph Classification

In this section, we describe our proposed algorithm for multi-task graph classification.

8.3.1 Regularized Multi-task Graph Classification Formulation

To achieve multi-task graph classification, our theme is to use multi-task to guide an iterative subgraph exploration process which leads to the lowest regularized empirical risks (for all tasks). This can be formulated as the following objective function:

$$\mathcal{J} = \min_{\boldsymbol{W},\boldsymbol{b}} \underbrace{\sum_{t=1}^{T} \frac{1}{n_t} \sum_{i=1}^{n_t} \mathcal{L}(y_{t,i}, f_t(\boldsymbol{x}_{t,i}))}_{\mathcal{C}} + \gamma R(\boldsymbol{W}) \tag{8.3}$$

Here $\boldsymbol{W} = [\boldsymbol{w_1}, \cdots, \boldsymbol{w_T}]$ is a weight matrix indicating the weights of each subgraph on different tasks, $\boldsymbol{b} = [b_1, \cdots, b_T]$ are the bias parameters for each function f_t. The first term $\mathcal{C}$ measures the loss on the training graphs for all tasks, where $\mathcal{L}(y_{t,i}, f(\boldsymbol{x}_{t,i}))$ is a loss function measuring the misclassification penalty of a graph $G_{t,i}$. The second part is a regularization term to enforce sparse solutions, and a parameter γ is used to control the magnitude of the regularization part. We mainly consider the logistic loss function

$$\mathcal{L}(y_{t,i}, f_t(x_{t,i})) = \log(1 + \exp\{-y_{t,i} f_t(\boldsymbol{x}_{t,i})\}) \tag{8.4}$$

Note that any other differentiable loss function, such as least square loss $\mathcal{L}(y, f_t) = \frac{1}{2}(y - f_t)^2$ or exponential loss $\mathcal{L}(y, f_t) = \exp\{-y f_t\}$, can be used in our algorithm. As for the second term $R(\boldsymbol{W})$, our main objective is to obtain a sparse solution on $\boldsymbol{W}$, *i.e.*, a finite set of subgraph features shared by all tasks. We consider the following regularizers:

- ℓ_1**-norm Lasso Regularization**

$$R(\boldsymbol{W}) = \sum_{k,t} |\boldsymbol{W}_{k,t}|$$

.

The rational is that ℓ_1-norm regularizer can produce solutions with many coefficients being 0, which is known as Lasso [130] and has been widely applied for variable selections. A simplification of Lasso in MTG is to use a parameter γ to control the regularization of all tasks, assuming that different tasks share the same sparsity parameter.

- $\ell_{2,1}$**-norm Regularization.** Because the total subgraph space is infinitely large, and we want to select only a few subgraphs among all possible ones, we propose to use a mixed-norm regularizers $\ell_{2,1}$ norm

$$\|\boldsymbol{W}\|_{2,1} = \sum_{k=1}^{m} \sqrt{\sum_{t=1}^{T} |\boldsymbol{W}_{k,t}|^2} = \sum_{k=1}^{m} \|\boldsymbol{W}_{k,\cdot}\|_2$$

where $\boldsymbol{W}_{k,\cdot}$ is the k-th row of $\boldsymbol{W}$. The $\ell_{2,1}$ regularizer first computes the ℓ_2-norm (across the tasks) of each row in $\boldsymbol{W}$ and then calculates the ℓ_1-norm of the vector $d(\boldsymbol{W}) = (\|\boldsymbol{W}_{1,\cdot}\|_2, \cdots, \|\boldsymbol{W}_{m,\cdot}\|_2)$. This is a special case of group Lasso [155] for group variable selection and was previously applied in [8] for multi-task learning on vector data. This norm ensures that common features will be selected across all tasks. Using this regularizer can produce some rows of $\boldsymbol{W}$ be $\mathbf{0}$. If a row $\boldsymbol{W}_{k,\cdot} = \mathbf{0}$ [1], the subgraph (feature) g_k will not be used in any tasks.

8.3.2 Multi-task Graph Classification: Challenges and Solution Sketch

Challenges: When the whole feature set $\mathcal{F} = \{g_1, \cdots, g_m\}$ is small and available for learning, the objective function in Eq. (8.3) can be effectively solved by using some existing toolbox [164] for either ℓ_1 or $\ell_{2,1}$ norm regularization. For graph data, however, the challenges are twofold: (1) the whole feature set $\mathcal{F}$ is implicit and unavailable, until we fully enumerate all subgraphs for all training graphs, which is NP-complete. (2) the number of subgraphs is huge and possibly infinite ($m \to +\infty$).

Solution Sketch: To solve the aforementioned challenges, we propose to iteratively include features/subgraphs into our objective function. In other words, the multi-task subgraph selection and model learning are integrated into one objective function for mutual benefits. More specifically, we perform subgraph selection based on subgradient of the objective function $\mathcal{J}$, so the empirical loss can always be reduced when selecting and adding the most discriminative subgraph to the existing subgraph feature set. After a new subgraph is incorporated, we re-solve the new *restricted master problem*[2] of Eq. (8.3), which is defined as

[1] $\mathbf{0}$ or $\mathbf{1}$ indicates T dimensional vectors with all 0 or 1 values.
[2] A reduced problem based on the selected features only.

follows:

$$\mathcal{J}_1 = \min_{\boldsymbol{W}^{(s)},\boldsymbol{b}^{(s)}} \underbrace{\sum_{t=1}^{T} \frac{1}{n_t} \sum_{i=1}^{n_t} \mathcal{L}(y_{t,i}, f_t(\boldsymbol{x}_{t,i}^{(s)}))}_{\mathcal{C}} + \gamma R(\boldsymbol{W}^{(s)}) \tag{8.5}$$

where $\boldsymbol{W}^{(s)}$ and $\boldsymbol{b}^{(s)}$ are the solutions based on the selected features in the s-th iteration, and $\boldsymbol{x}_{t,i}^{(s)}$ is feature representation of $\boldsymbol{x}_{t,i}$ w.r.t the selected features.

The aforementioned feature selection and model learning procedure continue until the algorithm converges. To handle the huge subgraph space, we derive two branch-and-bound pruning rules to reduce the searching space. The above algorithm design enjoys two unique advantages: (1) The discriminative subgraph selection is driven by the well defined multi-task learning objective function for model learning, and (2) the model learning will be further enhanced by the including new selected discriminative subgraph features.

Our method is based on the gradient/subgradient on the functional space of the objective function Eq. (8.3). Let us define the gradient of the loss term $\mathcal{C}$ in Eq. (8.3) on the subgraph feature g_k with respect to the t-th task as $\nabla \mathcal{C}_{k,\boldsymbol{w}_t}$.

$$\begin{aligned} \nabla \mathcal{C}_{k,\boldsymbol{w}_t} &= \frac{\partial \mathcal{C}}{\partial \boldsymbol{w}_{k,t}} = \frac{1}{n_t} \sum_{i=1}^{n_t} \frac{\partial \mathcal{L}(y_{t,i}, f_t(\boldsymbol{x}_{t,i}))}{\partial f_t(\boldsymbol{x}_{t,i})} \frac{\partial f_t(\boldsymbol{x}_{t,i})}{\partial \boldsymbol{w}_{k,t}} \\ &= -\frac{1}{n_t} \sum_{i=1}^{n_t} \frac{y_{t,i} \boldsymbol{x}_{t,i}^k}{1+e^{y_{t,i} f_t(\boldsymbol{x}_{t,i})}} = \sum_{i=1}^{n_t} y_{t,i} \alpha_{t,i} \boldsymbol{x}_{t,i}^k \end{aligned} \tag{8.6}$$

Here, $\alpha_{t,i} = -\frac{1}{n_t(1+e^{y_{t,i} f_t(\boldsymbol{x}_{t,i})})}$ is the gradient for the graph sample $\boldsymbol{x}_{t,i}$, in the latter, we can regard it as a weight associated to graph $G_{t,i}$ for subgraph mining process.

Then the gradient vector on feature g_k over all T tasks is defined as:

$$\nabla C_{k,\cdot} = [\nabla \mathcal{C}_{k,\boldsymbol{w}_1}, \cdots, \nabla \mathcal{C}_{k,\boldsymbol{w}_T}] \tag{8.7}$$

8.3.3 Optimal Subgraph Candidate Exploration

Because we assume that some subgraph/features g_k will not be used for learning the classification models, *i.e.*, $\boldsymbol{W}_{k,\cdot} = \mathbf{0}$, it makes sense to partition the whole subgraph features $\mathcal{F}$ into two disjoint subsets $\mathcal{F}_1$ and $\mathcal{F}_2$. $\mathcal{F}_1$ stores active features which are used to learn classification model and this set is frequently updated as desired, and $\mathcal{F}_2$ includes unselected graphs with $\mathbf{0}$ weights (*i.e.*, for $g_k \in \mathcal{F}_2, \boldsymbol{W}_{k,\cdot} = \mathbf{0}$). Then we can iteratively select the best features from $\mathcal{F}_2$ to $\mathcal{F}_1$.

Stopping Conditions and Conditional Score. According to the optimal conditions, when reaching the optimum, the first derivative of Eq. (8.3) should be 0:

$$\nabla C_{k,\boldsymbol{w}_t} + \gamma \boldsymbol{o}_{k,t} = 0; \tag{8.8}$$

Where $\boldsymbol{o}_{k,t}$ is the subgradient of the ℓ_1 or $\ell_{2,1}$ norm of $\boldsymbol{W}_{k,t}$. Let $\boldsymbol{o}_k = [\boldsymbol{o}_{k,1}, \cdots, \boldsymbol{o}_{k,T}]$ be the subgradient vector over all tasks. For the ℓ_1-norm of $\boldsymbol{W}_{k,\cdot}$ (*i.e.*, $|\boldsymbol{W}_{k,\cdot}|$), each dimension of $\boldsymbol{o}_k$ is as follows:

$$\boldsymbol{o}_{k,t} \in \begin{cases} [-1, 1] & : \quad \boldsymbol{W}_{k,t} = 0 \\ \mathbf{sign}(\boldsymbol{W}_{k,t}) & : \quad \boldsymbol{W}_{k,t} \neq 0 \end{cases} \tag{8.9}$$

Now we can state the optimal condition for ℓ_1 norm regularization. According to Eq. (8.8) and Eq. (8.9), a vector $\hat{\boldsymbol{W}} = [\hat{\boldsymbol{w}}_1, \cdots, \hat{\boldsymbol{w}}_t]$ is the optimal solution of our objective function Eq. (8.3) if and only if:

$$\|\nabla \mathcal{C}_{k,\cdot}\|_\infty \leq \gamma \quad \text{if} \quad \hat{\boldsymbol{W}}_{k,\cdot} = \mathbf{0} \tag{8.10}$$

$$\nabla \mathcal{C}_{k,\cdot} + \gamma \mathbf{sign}(\hat{\boldsymbol{W}}_{k,\cdot}) = \mathbf{0} \quad \text{if} \quad \hat{\boldsymbol{W}}_{k,\cdot} \neq \mathbf{0} \tag{8.11}$$

where $\|\nabla \mathcal{C}_{k,\cdot}\|_\infty = \max_{t=1}^{T} |\nabla \mathcal{C}_{k,\boldsymbol{w}_t}|$. Eq. (8.10) ensures that $\forall t, |\nabla \mathcal{C}_{k,\boldsymbol{w}_t}| \leq \gamma$. Here we have the "if and only if" condition because our objective function Eq. (8.3) is convex, so any local optimum is a global optimum. As a result, Eq. (8.8) is a necessary and sufficient condition of reaching the optimum.

Similarly, for the $\ell_{2,1}$-norm, $\boldsymbol{o}_k$ for $\|\boldsymbol{W}_{k,\cdot}\|_2$ are:

$$\boldsymbol{o}_k \in \begin{cases} \boldsymbol{z} \in \mathbb{R}^T \ , \ \|\boldsymbol{z}\|_2 \leq 1 & : \ \boldsymbol{W}_{k,\cdot} = \boldsymbol{0} \\ \frac{\boldsymbol{W}_{k,\cdot}}{\|\boldsymbol{W}_{k,\cdot}\|_2} & : \ \boldsymbol{W}_{k,\cdot} \neq \boldsymbol{0} \end{cases} \tag{8.12}$$

Therefore, according to Eq. (8.8) and Eq. (8.12), a vector $\hat{\boldsymbol{W}} = [\hat{\boldsymbol{w}}_1, \cdots, \hat{\boldsymbol{w}}_t]$ is the optimal solution to our objective function Eq. (8.3) if and only if:

$$\|\nabla \mathcal{C}_{k,\cdot}\|_2 \leq \gamma \quad \text{if} \quad \hat{\boldsymbol{W}}_{k,\cdot} = \boldsymbol{0} \tag{8.13}$$

$$\nabla \mathcal{C}_{k,\cdot} + \gamma \|\hat{\boldsymbol{W}}_{k,\cdot}\|_2^{-1} \hat{\boldsymbol{W}}_{k,\cdot} = \boldsymbol{0} \quad \text{if} \quad \hat{\boldsymbol{W}}_{k,\cdot} \neq \boldsymbol{0} \tag{8.14}$$

In order to reduce the objective value of $\mathcal{J}$ in Eq. (8.3), we propose to select subgraphs in $\mathcal{F}_2$ whose weight violates Eq. (8.10) for ℓ_1-norm regularizer or Eq. (8.13) for $\ell_{2,1}$-norm regularizer, and update the selected active set $\mathcal{F}_1$ with the newly selected features and re-optimize the Eq. (8.3) with current features. This process will repeat until no candidate violates either Eq. (8.10) or Eq. (8.13). In other words, these two equations can naturally induce the stopping criterion for our process. Let us define the conditional score of a subgraph as follows:

Definition 13. ***Conditional Score:*** *For a subgraph pattern g_k, its conditional score over all T tasks is defined as:*

$$\Upsilon(g_k) = \|\nabla \mathcal{C}_{k,\cdot}\|_q, q \in \{\infty, 2\} \tag{8.15}$$

where $q = \infty$ for ℓ_1 regularization and $q = 2$ for $\ell_{2,1}$ regularization; $\nabla \mathcal{C}_{k,\cdot}$ is defined in Eq. (8.7).

As a result, all candidate subgraphs which violate Eq. (8.10) or Eq. (8.13) can be defined as

$$\mathcal{F}_3 = \{g_k | g_k \in \mathcal{F}_2, \Upsilon(g_k) > \gamma\} \tag{8.16}$$

$\mathcal{F}_3$ defines all candidate subgraphs which can be selected and added to $\mathcal{F}_1$.

Optimal Multi-task Subgraph Selection: Intuitively, any subgraph in $\mathcal{F}_3$ can be selected and added to $\mathcal{F}_1$ in each iteration. To ensure quick convergence, we will select the one with the most effect in reducing the function value of $\mathcal{J}$ in Eq. (8.3). From Eq. (8.3) and (8.8), the gradients for subgraph g_k over T tasks are defined as:

$$\Gamma = \sum_{t=1}^{T} \nabla C_{k,\boldsymbol{w}_t} + \gamma \sum_{t=1}^{T} \boldsymbol{o}_{k,t} = \nabla C_{k,\cdot} \cdot \mathbf{1} + \gamma \boldsymbol{o}_k \cdot \mathbf{1} \tag{8.17}$$

From Eq. (8.9) and (8.12), we know that $\mathbf{0}$ is a feasible subgradient for both ℓ_1 and $\ell_{2,1}$ norm regularizers. Therefore, we can set $\boldsymbol{o}_k = \mathbf{0}$, in such case, $\Gamma = \nabla \mathcal{C}_{k,\cdot} \cdot \mathbf{1}$. Then we can compute the absolute value $|\nabla \mathcal{C}_{k,\cdot} \cdot \mathbf{1}|$, and choose the largest one each time (because it will possibly have the most impact in reducing $\mathcal{J}$ in Eq. (8.3)).

8.3.4 Multi-task Graph Classification Algorithm

Before we proceed to explain our multi-task graph classification algorithm using the gradients/subgradients for subgraph mining, we formally define the multi-task score for a subgraph to quantify its utility value for MTG as follows:

Definition 14. ***MTG Discriminative Score:*** *For a subgraph pattern g_k, its discriminative score over all T tasks is defined as follows:*

$$\Theta(g_k) = |\nabla \mathcal{C}_{k,\cdot} \cdot \mathbf{1}| = |\sum_{t=1}^{T} \nabla \mathcal{C}_{k,\boldsymbol{w}_t}| \tag{8.18}$$

where $\nabla \mathcal{C}_{k,\cdot}$ and $\nabla \mathcal{C}_{k,\boldsymbol{w}_t}$ are defined in Eq. (8.7) and Eq. (8.6), respectively.

Algorithm 12 illustrates the detailed steps of our iterative subgraph feature learning for multi-task graph classification. Initially, the weights for all training graphs in each task are equally set as $1/n_t$ (n_t is the number of labeled subgraph in task t), and the active set $\mathcal{F}_1$ is initialized to be empty.

In the next step, the algorithm mines a set of subgraphs $\mathcal{P}$ from $\mathcal{F}_3$ which have the highest MTG discriminative scores defined by Eq. (8.18). This step involves a multi-task driven subgraph mining procedure, which will be addressed in the next subsection. In order to reduce the number of iterations for subgraph mining, we mine top K subgraphs in each iteration (instead of the best one).

On steps 4-5, if the current graph set $\mathcal{P}$ is empty, it means that there is no more subgraph violating the optimal condition of Eq. (8.10) or Eq. (8.13), so the algorithm will stop. On step 6, we add the newly selected subgraphs $\mathcal{P}$ to existing subgraph set $\mathcal{F}_1$, and re-solve the restricted objective function Eq. (8.5) on step 7. To solve the restricted objective function, we use the MALSAR toolbox [1] in our experiments.

In the last step, the algorithm updates the weight $\alpha_{t,i}$ for each graph $G_{t,i}$. This will help compute the gradient vector of $\nabla \mathcal{C}_{k,\cdot}$ (for the purpose of computing MTG discriminative score of each subgraph) for subgraph mining in next round.

Theoretical Study: A nice property of our algorithm is that its theoretical convergence is assured as shown in Theorem 4.

Theorem 4. ***(Convergence Properties:)*** *Algorithm 12 guarantees that the restricted objective function Eq. (8.5) will monotonically decrease.*

Proof. Without loss of generality, we assume in each iteration, a subgraph is selected and added to $\mathcal{F}_1$, i.e., we set $K = 1$ in Algorithm 12. Let the optimal objective value based on current s features (i.e., $|\mathcal{F}_1| = s$) with respect to Eq. (8.5) is obtained at $(\hat{\boldsymbol{W}}^{(s)}, \hat{\boldsymbol{b}}^{(s)})$, i.e.,

$$\mathcal{J}_1(\hat{\boldsymbol{W}}^{(s)}, \hat{\boldsymbol{b}}^{(s)}) = \underbrace{\sum_{t=1}^{T} \frac{1}{n_t} \sum_{i=1}^{n_t} \mathcal{L}(y_{t,i}, f_t(\boldsymbol{x}_{t,i}^{(s)}))}_{\mathcal{C}} + \gamma R(\hat{\boldsymbol{W}}^{(s)})$$

[1] http://www.MALSAR.org

Then in the $t+1$-th iteration, the optimal objective value of Eq. (8.5) is:

$$\min \mathcal{J}_1(\boldsymbol{W}^{(s+1)}, \boldsymbol{b}^{(s+1)}) = \min(\mathcal{C} + \gamma R)|\forall(\boldsymbol{W}^{(s+1)}, \boldsymbol{b}^{(s+1)})$$
$$\leq (\mathcal{C} + \gamma R)|([\hat{\boldsymbol{W}}^{(s)}; \boldsymbol{0}], \hat{\boldsymbol{b}}^{(s)})$$

Thus the objective value of the restricted problem Eq.(8.5) based on the currently selected features $\mathcal{F}_1$ will always monotonously decrease in two successive iterations. Because the objective function value is non-negative (bounded), we can ensure that it will finally converge as iteration continues. □

Algorithm 12 Multi-task Graph Classification Algorithm

Require:
$\{(G_{t,1}, y_{t,1}), \cdots, (G_{t,n}, y_{t,n})\}, t \in \{1, 2, \cdots, T\}$: Graph Datasets from T tasks;
S_{max}: Maximum number of iterations;
K: Number of optimal subgraph used in each iteration;
Ensure:
$\boldsymbol{W}^{(s)}, \boldsymbol{b}^{(s)}$: Parameters for multi-task models
1: $\alpha_{ti} = 1/n_t$; $\mathcal{F}_1 \leftarrow \emptyset$; $s \leftarrow 1$;
2: **while** $s \leq S_{max}$ **do**
3: Mine top-K subgraph features $\mathcal{P} = \{g_i\}_{i=1,\cdots,k}$ from $\mathcal{F}_3$ with maximum discriminative score defined by Eq. 8.18 ; //Algorithm 13;
4: **if** $\mathcal{P} = \emptyset$ **then**
5: **break;**
6: $\mathcal{F}_1 \leftarrow \mathcal{F}_1 \bigcup \mathcal{P}$;
7: Solve Eq. (8.5) based on $\mathcal{F}_1$ to get new weights matrix $\boldsymbol{W}^{(s)}, \boldsymbol{b}^{(s)}$;
8: Update the graph weights on each training graph

$$\alpha_{t,i} = -\frac{1}{n_t(1+e^{y_{t,i} f_t(\boldsymbol{x}_{t,i})})}$$

9: $s \leftarrow s + 1$;
10: **return** $\boldsymbol{W}^{(s)}, \boldsymbol{b}^{(s)}$;

According to optimal conditions (Eq. (8.10) or Eq. (8.13)), if the algorithm has reached the optimal solution, $\forall g_k, g_k \notin \mathcal{F}_1$, we will have $\boldsymbol{W}_{k,\cdot} = \boldsymbol{0}$, thus its conditional score $\Upsilon(g_k) = \|\nabla \mathcal{C}_{k,\cdot}\|_q < \gamma, q \in \{\infty, 2\}$. In such case, no more

subgraphs will be obtained in $\mathcal{P}$ from $\mathcal{F}_3$, *i.e.*, $\mathcal{P} = \emptyset$. Thus our stopping condition (steps 4-6) guarantees the optimal solution of our algorithm.

8.3.5 Multi-Task Driven Subgraph Mining

To obtain a set of discriminative features $\mathcal{P}$ in $\mathcal{F}_3$ from the T tasks of training graphs, we need to perform the subgraph enumeration procedure. In order to mine the top-K subgraphs on step 3 of Algorithm 12, we need to enumerate the entire set of subgraph patterns from the training graphs of all tasks. In our MTG algorithm, we employ a frequent subgraph mining based algorithm gSpan [151]. By employing a depth first search strategy on the DFS Code tree (where each node is a subgraph), gSpan can enumerate all frequent subgraphs efficiently.

During the subgraph mining process, because the search space is exponentially large/infinite, an effective pruning scheme is essential. In this subsection, we will first derive the upper-bound of MTG discriminative score, and then provide a conditional score upper-bound. Both of them will help prune the searching space and speedup the subgraph mining.

Theorem 5. ***(MTG Discriminative Score Upper-bound:)*** *Let g and g' are two subgraph patterns, and $g \subseteq g'$, for the subgraph g, we define*

$$\begin{aligned} A_1(g) &= 2\sum_{t=1}^{T}\sum_{\{i|y_{t,i}=+1, g\in G_{t,i}\}}\alpha_{t,i} \\ A_2(g) &= 2\sum_{t=1}^{T}\sum_{\{i|y_{t,i}=-1, g\in G_{t,i}\}}\alpha_{t,i} \\ A_3 &= \sum_{t=1}^{T}\sum_{i=1}^{n_t}\alpha_{t,i}y_{t,i} \end{aligned}$$

$$\hat{\Theta}(g) = \begin{cases} \max\{|A_1(g) - A_3|, |A_2(g)|\} & : \quad A_3 \geq 0 \\ \max\{|A_2(g) + A_3|, |A_1(g)|\} & : \quad A_3 < 0 \end{cases}$$

then $\Theta(g') \leq \hat{\Theta}(g)$, where $\Theta(g')$ is defined in Eq. (8.18).

Proof. We start with the definition of $\Theta(g')$ in Eq. (8.18):

$$
\begin{aligned}
\Theta(g') &= |\sum_{t=1}^{T}\sum_{i=1}^{n_t} y_{t,i}\alpha_{t,i}f_t(\boldsymbol{x}_{t,i})| \\
&= |\sum_{t=1}^{T}\sum_{i=1}^{n_t} y_{t,i}\alpha_{t,i}\cdot[2I(g'\subseteq G_{t,i})-1]| \\
&= |2\sum_{t=1}^{T}\sum_{g'\subseteq G_t} y_{t,i}\alpha_{t,i} - \sum_{t=1}^{T}\sum_{i=1}^{n_t}\alpha_{t,i}y_{t,i}| \\
&= |A_1(g') - A_2(g') - A_3| \\
&\leq \begin{cases} \max\{|A_1(g')-A_3|, |A_2(g')|\} & : \quad A_3 \geq 0 \\ \max\{|A_2(g')+A_3|, |A_1(g')|\} & : \quad A_3 < 0 \end{cases} \\
&\leq \begin{cases} \max\{|A_1(g)-A_3|, |A_2(g)|\} & : \quad A_3 \geq 0 \\ \max\{|A_2(g)+A_3|, |A_1(g)|\} & : \quad A_3 < 0 \end{cases} \\
&= \hat{\Theta}(g)
\end{aligned}
$$

The first inequality holds because for $\alpha_{t,i} < 0$, $A_1(g') \leq 0$ and $A_2(g') \leq 0$, so the upper-bound depends on A_3. If $A_3 \geq 0$, $A_1(g')$ and A_3 will have different signs, then the upper-bound is the maximum one of $\{|A_1(g') - A_3|, |A_2(g')|\}$. The case is similar for $A_3 < 0$. The second inequality holds because $|A_1(g')| \leq |A_1(g)|$ and $|A_2(g')| \leq |A_2(g)|$ for $g \subseteq g'$. □

Theorem 5 states that for any super graph of a subgraph g, its MTG discriminative score, over T tasks, is upper-bounded by $\hat{\Theta}(g)$.

Single Discriminativeness Bound: The above discriminative score upper-bound can also be applied to each single task separately. If only the task t is considered, the single discrimativeness upper-bound is defined as $\hat{\Theta}(g,t)$, which requires that $A_1(g), A_2(g)$ and A_3 in Theorem 5 are computed over the task t only.

Theorem 6. ***(Conditional Score Upper-bound:)*** *Given two subgraph features g and g_k ($g \subseteq g_k$), and a set of upper-bounds of single discriminativeness*

bound:

$$\hat{\Upsilon}_{(g,\cdot)} = [\hat{\Theta}(g,1), \cdots, \hat{\Theta}(g,T)]$$

let

$$\hat{\Upsilon}(g) = \|\hat{\Upsilon}_{(g,\cdot)}\|_q, q \in \{\infty, 2\}$$

then $\Upsilon(g_k) \leq \hat{\Upsilon}(g)$, *where* $\Upsilon(g_k)$ *is defined in Eq. (8.15).*

Proof. The conditional score for g_k on the task t is $|\nabla \mathcal{C}_{k,\boldsymbol{w}_t}|$, which is upper-bounded by $\hat{\Theta}(g,t)$, *i.e.*, $\hat{\Theta}(g,t) \geq |\nabla \mathcal{C}_{k,\boldsymbol{w}_t}|$. Because every entry in $\Upsilon(g_k)$ is smaller than that in $\hat{\Upsilon}_{(g,\cdot)}$, the ℓ_∞ or ℓ_2 norm on the vector also holds, *i.e.*, $\hat{\Upsilon}(g) \geq \Upsilon(g_k)$. □

According to Theorem 6, once a subgraph g is generated, the conditional scores for all its super-graphs are upper-bounded by $\hat{\Upsilon}(g)$. Therefore, we use this rule to prune unpromising candidates effectively.

Multi-task Driven Subgraph Mining Algorithm: Our multi-task driven subgraph mining algorithm is listed in Algorithm 13. The minimum value η in optimal set $\mathcal{P}$ are initialized on step 1. Duplicated subgraph features are pruned on steps 4-5, and the discriminative score $\Theta(g_p)$ and conditional score $\Upsilon(g_p)$ for g_p are calculated on step 6. If g_p is included in the current candidate set $\mathcal{F}_3' = \{g_k | g_k \in \mathcal{F}_2, \Upsilon(g_k) > \gamma + \varepsilon\}$ and $\Theta(g_p)$ is larger than η, we add g_p to the feature set $\mathcal{P}$ (steps 7-8). Here, we have relaxed $\mathcal{F}_3$ from Eq. (8.16) to a ε-tolerance set, *i.e.*, $\mathcal{F}_3'$, because $\Upsilon(g_k)$ only changes subtly in the last few iterations (ε=0.005 in our experiments).

When the size of $\mathcal{P}$ exceeds the predefined size K, the subgraph with the minimum discriminative score is removed (steps 9-11). After that, the algorithm updates the minimum optimal value η on step 12, and uses two branch-and-bound pruning rules, Theorems 5 and 6, to prune the search space on steps 13. These two rules will reduce unpromising candidates by using discriminative scores and conditional scores perspectives, respectively. Finally, the optimal set $\mathcal{P}$ is returned on step 15.

Algorithm 13 Multi-Task Driven Subgraph Mining

Require:

$\{(G_{t,1}, y_{t,1}), \cdots, (G_{t,n}, y_{t,n})\}, t \in \{1, 2, \cdots, T\}$: Graph Datasets from T tasks;

γ : Predefined regularization parameter;

α_{ti} : Weight for each graph example;

K: Number of optimal subgraph patterns;

$\mathcal{F}_1$: Already selected subgraph set;

Ensure:

$\mathcal{P} = \{g_k\}_{k=1,\cdots,K}$: The top-$K$ subgraphs;

$\eta = 0$, $P \leftarrow \emptyset$;
while Recursively visit the DFS Code Tree in gSpan **do**
  $g_p \leftarrow$ current visited subgraph in DFS Code Tree;
  if g_p has been examined **then**
    continue;
  Compute scores $\Theta(g_p)$ and $\Upsilon(g_k)$ for subgraph g_p according Eq. (8.18) and Eq. (8.15);
  if $g_p \in \mathcal{F}_3'$ & $\Theta(g_p) > \eta$ **then**
    $\mathcal{P} \leftarrow \mathcal{P} \bigcup g_p$;
  if $|\mathcal{P}| > K$ **then**
    $g^\star \leftarrow \arg\min_{g_k \in \mathcal{P}} \Theta(g_k)$;
    $\mathcal{P} \leftarrow \mathcal{P}/\{g^\star\}$;
  $\eta \leftarrow \min_{g_k \in \mathcal{P}} \Theta(g_k)$;
  if $\hat{\Theta}(g_p) > \eta$ & $\hat{\Upsilon}(g_p) > \gamma$ **then**
    Depth-first search the subtree rooted from node g_p;
return $\mathcal{P} = \{g_k\}_{k=1,\cdots,K}$;

The above pruning process is a key feature of our algorithm, because we do not require any support threshold for subgraph mining (whereas all other subgraph mining methods will require users to predefine a threshold value).

8.4 Experiment

8.4.1 Experimental Settings

Benchmark Data: We validate the performance of the proposed algorithm on two multi-task graph classification datasets, *i.e.*, NCI and PTC collections, as

shown in table 3.2.

- **Anti-cancer activity prediction (NCI):** We select the NCI-balanced dataset, *i.e.*, the collection with balanced class distribution on each task.

- **Predictive Toxicology Challenge Dataset (PTC):** The PTC datasets are divided into four subsets. For each subset, we only consider one type of carcinogenicity test as its learning task.

Comparing Methods: In our experiments, we consider two baseline methods, from graph classification and multi-task learning perspectives, as follows:

- **gBoost** simply applies gBoost algorithm [120] to each graph classification task separately, without considering graph samples from other tasks.

- **MTL-ℓ_1 and MTL-ℓ_{21}** firstly mine a set of frequent subgraphs from the whole training graphs (we set minimum support as 0.1, which results in over 2500 subgraph features on NCI datasets), and then use those features to transfer each graph dataset into vector format, and then apply traditional Multi-task Learning algorithms to the transferred vector datasets. For **MTL-ℓ_1**, it uses ℓ_1 regularization. And for **MTL-ℓ_{21}**, it employs $\ell_{2,1}$ regularization, as [8] does. Both methods are implemented with logistic loss function and available in MALSAR toolbox [164].

- **MTG-ℓ_1 and MTG-ℓ_{21}** are our proposed methods, with **MTG-ℓ_1** regularized by ℓ_1 norm, and **MTG-ℓ_{21}** regularized by $\ell_{2,1}$ norm.

Unless otherwise specified, the parameters for MTG are set as follows: K=15, and $S_{max} = 15$. $\gamma = 0.01$ is set for both MTL-ℓ_1 and MTG-ℓ_1, $\gamma = 0.02$ is set for both MTL-ℓ_{21} and MTG-ℓ_{21}. Detailed studies of parameters K and γ are reported in Section 5.2.3. For gBoost algorithm, the parameter v is set to 0.2, as it usually achieves good results on both NCI and PTC datasets.

8.4.2 Experimental Results

8.4.2.1 Results on NCI Tasks

For NCI multi-task collection, we randomly label a small set of graphs as training graphs for each task, the rest are used for test. The number of training graphs in each task is vary from 50 to 400. We conduct each group of experiment 10 times and report the average accuracies and AUC values for each single task under 10 trials of experiment in Fig. 8.3 and Fig. 8.4, respectively.

The results in Figs. 8.3 and 8.4 show that with the increase of training data for each task, all algorithms obtain continuous improvement gains for both accuracy and AUC values. Over all graph classification tasks, MTG algorithms, including MTG-ℓ_1 and MTG-ℓ_{21}, outperform both gBoost (STG algorithm) and MTL algorithms for vector data significantly. Meanwhile, gBoost algorithm and MTL algorithms are comparable to each other. This is mainly because gBoost and MTL each has its own strength and weakness in handling multi-task graph classification problems. More specifically, gBoost is designed for graph classification, so it can select the most discriminative subgraphs for each single task. The previous study has shown that gBoost outperforms traditional frequent subgraph based algorithm for STG problems [120]. However, the key weakness of gBoost for MTG is that it ignores relevant graphs from similar tasks. When the number of labeled graphs in each task is very limited, the selected subgraphs may overfit the training graphs, leading to deteriorated classification results. For MTL algorithms, regardless of using ℓ_1 or $\ell_{2,1}$ regularization, they will first mine a set of frequent subgraph as features and then employ multi-task learning techniques for classification. Although these methods can enjoy some benefits of MTL by jointly optimizing related learning tasks, their subgraph mining process is not driven by the multi-task learning objective, and will therefore miss some genuine discriminative subgraphs at the first step. As a result, the classification performance of MTL methods is no better than gBoost.

In contrast, the proposed MTG-ℓ_1 and MTG-ℓ_{21} not only take the advantages of using graph samples from multiple relevant tasks, but also unify both multi-task subgraph feature selection and model learning into one objective function. This design helps both methods outperform gBoost and MTL algorithms with

Table 8.1: Accuracies on 9 NCI graph classification tasks *w.r.t* different numbers of training graphs in each task

#Train	gBoost	MTL-ℓ_1	MTL-ℓ_{21}	MTG-ℓ_1	MTG-ℓ_{21}
50	0.590	0.600	0.605	0.609	**0.622**
100	0.617	0.632	0.636	0.656	**0.673**
150	0.638	0.653	0.653	0.684	**0.697**
200	0.658	0.661	0.669	0.701	**0.719**
250	0.665	0.666	0.676	0.709	**0.727**
300	0.674	0.671	0.690	0.715	**0.735**
350	0.675	0.675	0.693	0.715	**0.738**
400	0.676	0.680	0.701	0.727	**0.750**

Table 8.2: AUC values on 9 NCI graph classification tasks *w.r.t* different numbers of training graphs in each task

#Train	gBoost	MTL-ℓ_1	MTL-ℓ_{21}	MTG-ℓ_1	MTG-ℓ_{21}
50	0.619	0.630	0.645	0.651	**0.667**
100	0.656	0.679	0.683	0.713	**0.731**
150	0.682	0.707	0.711	0.745	**0.761**
200	0.713	0.719	0.730	0.763	**0.785**
250	0.716	0.727	0.738	0.773	**0.792**
300	0.727	0.734	0.752	0.781	**0.804**
350	0.730	0.737	0.758	0.784	**0.812**
400	0.727	0.743	0.770	0.795	**0.820**

significant performance gains.

Another interested fact reflected by Fig. 8.3 and Fig. 8.4 is that MTG-ℓ_{21} outperforms MTG-ℓ_1 on most tasks. This is because $\ell_{2,1}$ regularization considers group effect, which is a special group lasso [155] and usually has better performance for group variable selection.

The average results, in terms of accuracy and AUC values, with respect to various training graphs over all task are described in Table 8.1 and Table 8.2. The results demonstrated that MTG-$\ell_{2,1}$ can achieve significant improvements over gBoost and MTL methods. For instance, it outperforms gBoost and MTL-ℓ_1 at 9.3% and 7.7% in term of AUC value(400 samples each task), respectively.

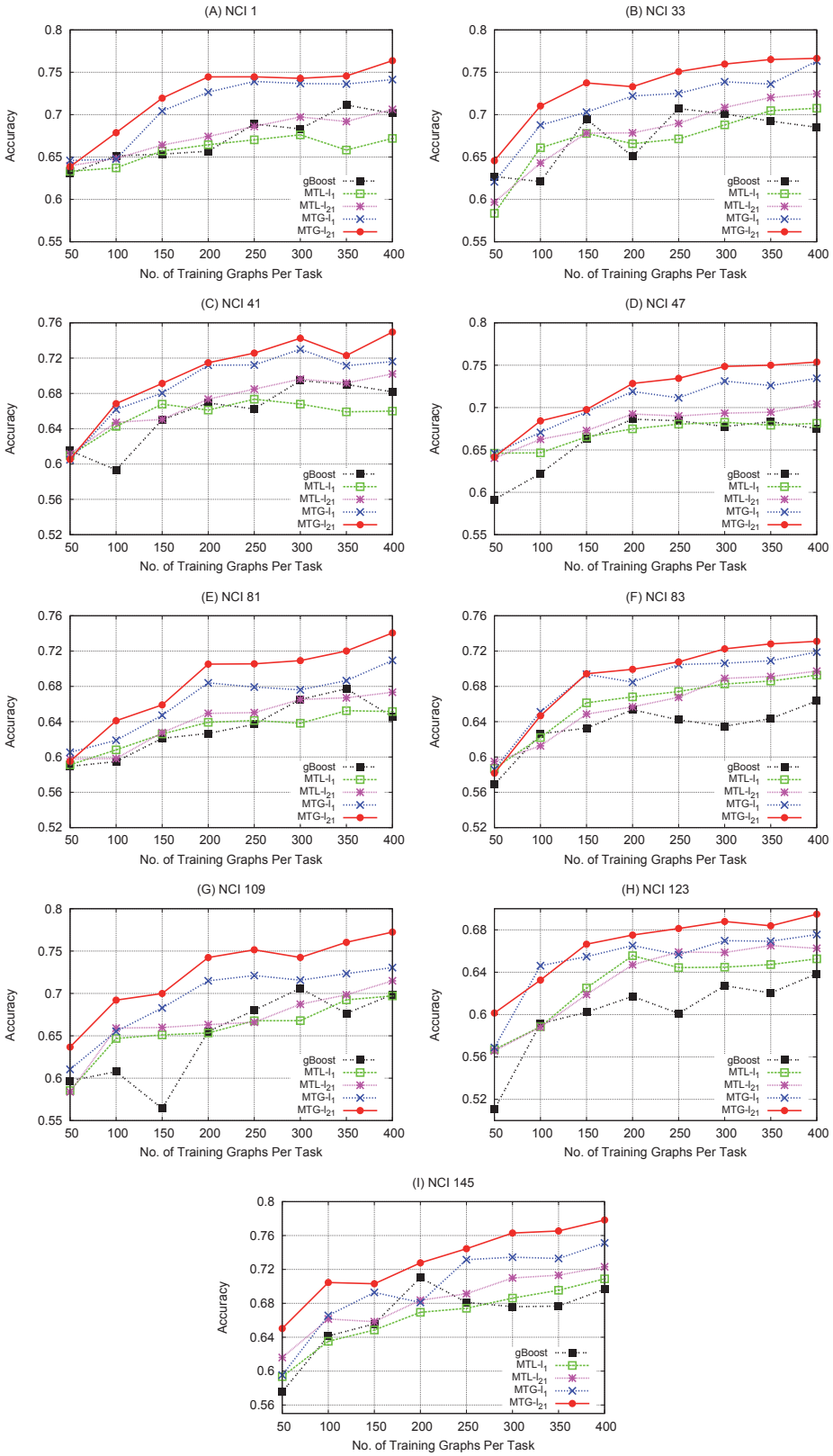


Figure 8.3: The classification accuracy of each single task *w.r.t.* the number of training graphs in each task.

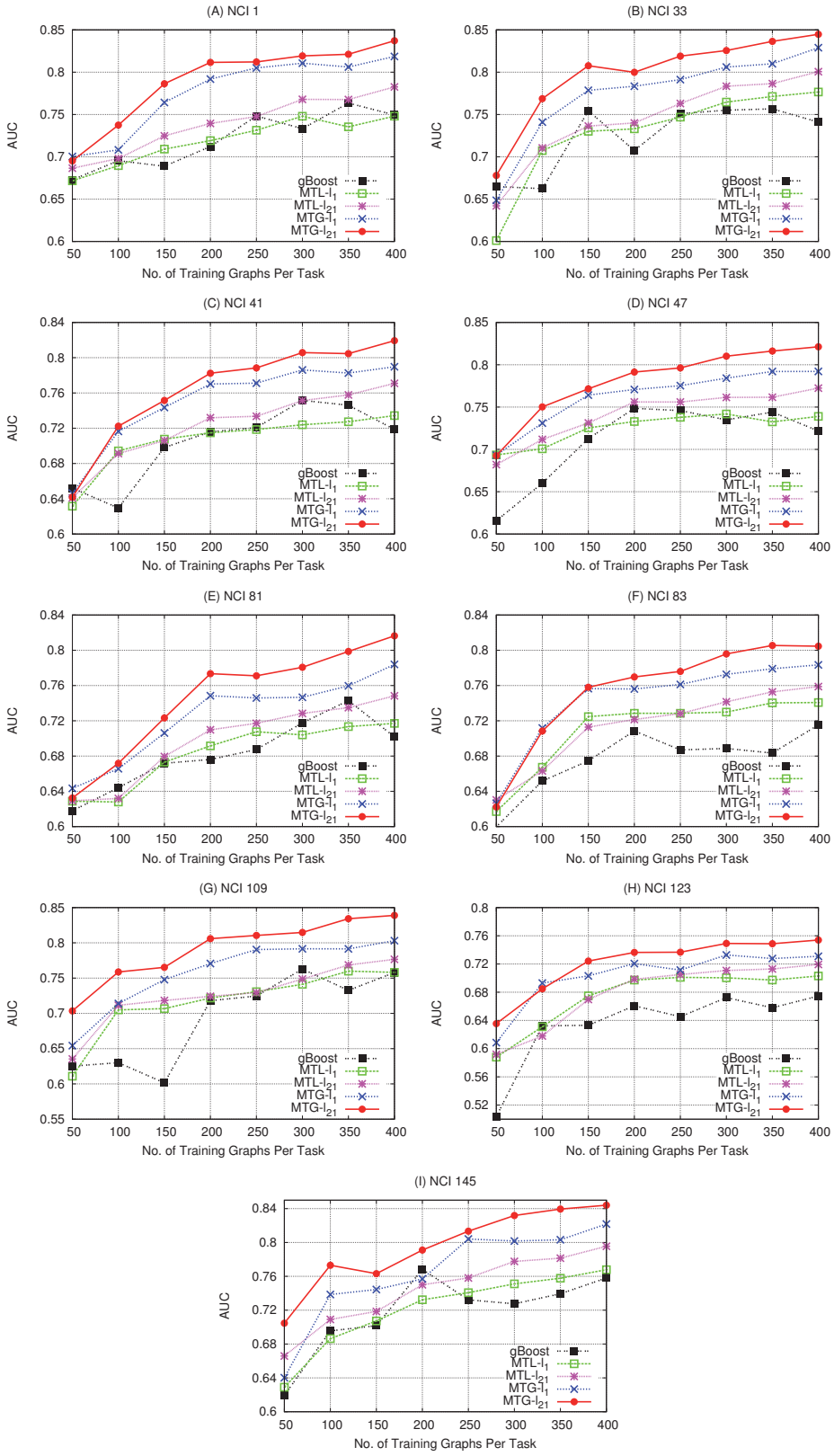


Figure 8.4: The AUC values of each single task *w.r.t.* the number of training graphs in each task.

Table 8.3: Accuracies on PTC tasks

Tasks	gBoost	MTL-ℓ_1	MTL-ℓ_{21}	MTG-ℓ_1	MTG-ℓ_{21}
MR	0.573	**0.664**	0.643	0.594	0.655
FR	0.561	0.522	0.547	0.541	**0.607**
MM	0.640	0.648	0.623	0.658	**0.677**
FM	0.680	0.602	0.626	0.671	**0.682**
Avg.	0.613	0.609	0.610	0.616	**0.655**

Table 8.4: AUC values on PTC tasks

Tasks	gBoost	MTL-ℓ_1	MTL-ℓ_{21}	MTG-ℓ_1	MTG-ℓ_{21}
MR	0.574	0.574	0.586	0.631	**0.656**
FR	0.522	0.516	0.517	0.505	**0.591**
MM	0.600	0.563	0.597	0.624	**0.671**
FM	0.686	0.601	0.623	0.696	**0.702**
Avg.	0.596	0.563	0.581	0.614	**0.655**

8.4.2.2 Results on PTC Tasks

For PTC graph classification tasks, the number of training graphs for each task is very limited. So instead of varying the training samples for each task (such as for NCI tasks), we conduct 10-fold cross-validation on PTC tasks. In this way, we can reduce the bias of each method caused by limited training samples. The accuracies and AUC values are reported in Tables 8.3 and 8.4.

The results in Tables 8.3 and 8.4 show that MTG methods achieve considerable performance gains over gBoost and MTL methods for almost all task. MTG-ℓ_{21} outperforms others methods on 3 out of 4 tasks in terms of accuracy, and beats all its peers over all tasks in terms of AUC. Note that for PTC tasks, AUC values are more important because they are all imbalanced tasks.

8.4.2.3 Convergence Study and Parameter Analysis

In this subsection, we study the impact of parameters K and γ on the algorithm performance.

Table 8.5: Running statistics *w.r.t* different K values for MTG-ℓ_{21} (50 training graphs for each task, $S_{max} = 150$)

K	Obj($\mathcal{J}_1$)	#Iter	$\lvert\mathcal{F}_1\rvert$	Accuracy	AUC	Time(s)
1	2.0754	122	122	0.621	0.670	1032
5	2.0814	29	138	0.620	0.667	283
10	2.0861	18	164	0.620	0.669	203
15	2.0779	15	172	0.622	0.667	186
20	2.0771	14	209	0.622	0.669	181

Impact of K values: In order to study the role of the K value, which denotes the number of subgraph selected in each iteration, on the algorithm performance, we inspect the convergence and runtime performance of our algorithm with different K values. The results in Table 8.5 shows that small K values (*e.g.* K=1) requires a large number of iterations and more system runtime. When K values continuously increase, the number of iterations and runtime will drop dramatically because more subgraphs are discovered and included in the feature set each time. For large K values, there is not much difference in terms of algorithm runtime.

Interestingly, Table 8.5 shows that although different K values will result in different number of subgraphs to be finally selected in $\mathcal{F}_1$, the algorithm will always converge to a ε-tolerance optimal solution (ε is used in Algorithm 13) via solving objective function Eq. (8.5), as shown in column 2 of Table 8.5. As a result, their accuracy and AUC values are very close to each other, regardless of different K values selected in the experiments. This result actually demonstrates the convergence of our algorithms, and assures that K will mainly impact on the algorithm runtime performance.

The number of iterations in Eq. (8.3) show that our algorithm has fast convergence speed. When $K = 15$, it will take 15 iterations for algorithms to reach convergence. In practice, we found that there is no need to wait until the algorithm reach convergence for optimal results, so we set the maximum number of iteration S_{max}=15 in our experiments.

Impact of γ values: We vary the regularization parameter γ from 0.005 to 0.5, and report the results in Table 8.6, where the sparsity denotes the percentages of

Table 8.6: Results *w.r.t.* different γ values for MTG-ℓ_1 (50 training graphs for each task, $S_{max} = 15$)

γ	$\lvert\mathcal{F}_1\rvert$	Sparsity	Accuracy	AUC
0.005	178	0.720	0.613	0.654
0.01	225	0.785	0.609	0.651
0.05	219	0.905	0.606	0.641
0.1	115	0.922	0.588	0.619
0.5	8	1	0.5	0

zero elements in the final weight matrix $\boldsymbol{W}$. The results show that increasing γ values will result in increased sparsity, because ℓ_1 norm regularizes more elements to be 0. For small γ values (from 0.005 to 0.05), the accuracy and AUC values have minor differences. But for very large γ values ($\gamma = 0.5$), the regularization term dominates the objective function Eq. (8.3), with no subgraph being used for classification, which results in poor AUC values. Similar results are also observed for MTG-ℓ_{21} algorithm.

8.4.2.4 Runtime Efficiency Study

In this subsection, we investigate the pruning efficiency of MTG in reducing the search space [Theorem 5 and 6] for subgraph feature exploration. Because the search space is infinitely large, it is challenging to assess the pruning effectiveness of MTG. Accordingly, we introduce a threshold value *min_sup*, which denotes the minimum frequency of each qualified subgraph feature in the training graph datasets, to bound the number of subgraphs in the search space. By doing so, we know the total number of subgraph candidates, and then can assess the pruning efficiency by checking the percentage of candidates pruned by the pruning process.

In our experiments, the *min_sup* threshold value, together with Theorems 5 and 6, are used for pruning the search space in step 13 of Algorithm 13. Then our MTG is compared with the following baselines:

- **Fre-MTG**: this method only uses the support threshold *min_sup* to prune the search space [on step 13 of Algorithm 13], with Theorems 5 and 6 being discarded. In other words, the multi-task driven subgraph mining procedure

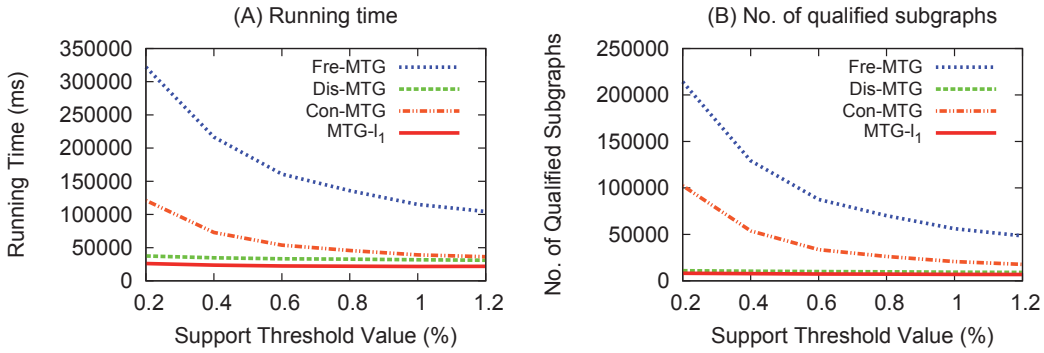


Figure 8.5: Pruning effectiveness with different pruning modules on NCI tasks for subgraph mining. A) Running time; B) Number of enumerated subgraphs.

is reduced to a classical frequent subgraph mining problem.

- **Dis-MTG**: this method uses the support threshold *min_sup* and the bound of discriminative score (Theorem 5) to prune the search space.

- **Con-MTG**: this method uses support threshold *min_sup* and the bound of conditional score (Theorem 6) to prune the search space.

The experimental results in Fig. 8.5.(A) show that with the increase of support threshold value *min_sup*, all methods experience reduced running time. This is because that a large support value will result in a small number of subgraph features (Fig. 8.5.(B)). Among the comparing methods, Fre-MTG consumes much more time than others because there is no pruning process to help reduce the search space.

By sequentially including the upper-bounds of discriminative score (Dis-MTG) and conditional score (Con-MTG) into the pruning process, the running time of the algorithm is reduced significantly. For instance, when using a small threshold 0.2 for NCI tasks, our MTG algorithm only takes about 28,000 ms to mine the optimal subgraphs whereas Fre-MTG requires about 330,000 ms. MTG algorithm is an order of magnitude faster than Fre-MTG, which demonstrates the significant pruning efficiency of our MTG algorithm.

It is worth noting that using a support threshold value *min_sup* in the subgraph pattern mining process may result in missing of discriminative subgraph

features, because some subgraph features may be very informative for classification but are not frequent to meet the support threshold value. However, discarding the support threshold value (*i.e.*, $min\_sup = 0$) will make most algorithms unable to find subgraph patterns. For example, in our experiments, we have tried to further reduce the support threshold min_sup for Fre-MTG, but it caused an out-of-memory error on a 16 GB memory machine for NCI tasks. In comparison, our MTG algorithm is able to mine discriminative subgraphs very quickly (in less than 40 seconds for NCI tasks), even if the support threshold is removed (*i.e.*, $min\_sup = 0$).

Our runtime efficiency study suggests that MTG is not only efficient in pruning the subgraph feature space to find high quality subgraph features, it can also carry out subgraph feature exploration without requiring the minimum support threshold value min_sup. As a result, it will result in a better opportunity and better efficiency to find discriminative subgraph features for multi-task graph classification.

8.5 Discussion

Relations to gBoost Algorithm. Our incremental subgraph selection algorithm advances the existing *column generation* style techniques [120] for graph classification. For gBoost, its learning objective function is

$$\begin{array}{ll} \max\limits_{\rho,\boldsymbol{w},\boldsymbol{\xi}} & \rho - \frac{1}{vn}\sum_{i=1}^{n}\xi_i \\ s.t. & y_i\sum_{k=1}^{m}\hbar_{g_k}(G_i)w_k + \xi_i \geq \rho; \\ & \sum_{k=1}^{m} w_k = 1; \\ & w_k \geq 0, \xi_i \geq 0; \end{array} \tag{8.19}$$

From [28], we know that this formula is equivalent to the following linear programming:

$$\begin{array}{ll} \min\limits_{\boldsymbol{w},\boldsymbol{\xi}} & \sum_{k=1}^{m} w_k + C\sum_{i=1}^{n}\xi_i \\ s.t. & y_i\sum_{k=1}^{m}\hbar_{g_k}(G_i)w_k + \xi_i \geq 1; \\ & w_k \geq 0, \xi_i \geq 0; \end{array} \tag{8.20}$$

Eq. (8.20) is actually a ℓ_1 svm formulation, and can be also formulated as regularized loss minimization formulation:

$$\min \|\boldsymbol{w}\|_1 + C \sum_{i=1}^{n} \mathcal{L}_h(y_i, f(\boldsymbol{x}_i)) \tag{8.21}$$

Here $\mathcal{L}_h(y_i, f(\boldsymbol{x}_i)) = \max(1 - y_i f(\boldsymbol{x}_i), 0)$, which is known as hinge loss in machine learning.

Compared to our objective function in Eq. (8.3), we find that gBoost (Eq. 8.21) is a special case of Eq. (8.3), with only one single task and ℓ_1-norm regularization being used and $\boldsymbol{w} \geq 0$. Although the hinge loss function is non-differentiable, our subgradient method still applies, as long as $\nabla \mathcal{C}_{k,\boldsymbol{w}_t}$ in Eq. (8.6) is defined properly. This observation shows the following advantages of our algorithm: (1) the gBoost algorithm is only designed for STG problem, whereas our algorithm can jointly learn multiple tasks simultaneously and can achieve better classification results; (2) gBoost employs a hinge loss function which is similar to SVM and requires the problem to be formulated as a linear programming. Our algorithm has removed the linear programming constraint and can employ any differentiable loss function, in addition to the logistic loss function considered in our paper. This generalization has great attractiveness in many applications, especially when the probability estimation for classification is required (the logistic function can provide some probabilistic information compared to the hinge loss function); (3) while gBoost employs ℓ_1 norm regularization to obtain sparse solution, our algorithm considers an additional mixed norm $\ell_{2,1}$ and provides a solution to incrementally select discriminative subgraphs for regularized loss minimization problems; (4) although this chapter mainly focuses on classification problems, our algorithm can be easily generalized to multi-task regression scenarios if a proper loss function, such as a least square loss function, is used.

8.6 Conclusion

In this chapter, we formulated a unique multi-task graph (MTG) classification problem. Our goal is to combine multiple graph classification tasks into one learning objective for all tasks to achieve optimal classification accuracies. We argued that due to the inherent complexity of the graph data and the costs involved in the labeling process, many graph classification tasks have very limited number of training samples. By unifying multiple tasks to guide the subgraph feature exploration and the succeeding learning process, multi-task graph classification has clear advantages of finding better subgraph features and avoiding overfitting, compared to models learned from each single task. In the chapter, an MTG algorithm is proposed to combine all tasks as a jointly regularized function, which ensures that the inclusion of subgraph features can only result in minimized regularization loss, which in turn leads to optimal learning models. Two Branch-and-bound pruning rules are also proposed to prune the search space effectively. Experiments and comparisons on real-world data confirmed the superb performance of our algorithms.

8. JOINT STRUCTURE FEATURE EXPLORATION AND REGULARIZATION FOR MULTI-TASK GRAPH CLASSIFICATION

Chapter 9

Conclusions and Future Work

This chapter summarize the whole thesis and provides some further research directions.

9.1 Summary of This Thesis

Due to the constant development of modern technologies in electronic devices, data collecting, and social networks, recent years has witnessed rapid increasing of *big data*. The "3V" properties of *big data* (volume, velocity, and variety) make the traditional data mining tasks unprecedentedly challenging. From the volume and velocity perspectives, data is increasing dramatically and rapidly, and a learning system should return the results in a timely fashion. To handle this problem, we develop streaming models for effective and efficient data mining in the thesis. From the view of variety, data is more and more interconnected and exhibits structural information (*i.e*, graphs). Considering volume, velocity, and variety as a whole, in the thesis, we studied complex graph stream mining, and proposed effective and efficient learning algorithms for three sub-tasks: (1) correlated graph stream search, (2) graph stream classification, and (3) complex task graph classification.

Specifically, in Part I, we proposed a CGStream algorithm for searching correlated subgraph patterns from10namic graph streams. The proposed method is an order of magnitude faster than the straightforward approach.

In Part II, we studied graph stream classification using labeled and unla-

beled graphs and proposed a novel algorithm to select discriminative subgraph features with minimum redundancy for graph stream classification. We also investigated imbalanced and noisy graph stream classification and proposed gEBoost algorithm to handle the imbalance, noise, and concept drift of dynamic graph streams.

Finally, we looked into complex task graph classification in Part III. We proposed an effective boosting algorithm, CogBoost, for cost-sensitive learning of large scale graphs. We also proposed to jointly learn multiple graph classification task for better performance gain and generalization ability for graph classification. Although we have focused on algorithms of complex task graph classification in Part III, these models can be easily extended to graph stream scenarios, by using the frameworks of graph stream classification in Part II.

9.2 Future Work

All the proposed algorithms in this thesis are based on the frequent subgraph mining framework gSpan [151]. As a result, the proposed algorithms inherit some disadvantages from gSpan. Specifically, gSpan algorithm is not scalable to millions or billions of graph samples. One possible solution is to replace the gSpan module with a distributed subgraph mining algorithm. We will consider using distributed subgraph mining algorithm for effective graph classification.

Other problems that remain unexplored in the research community, from the perspective of graph classification, include:

- **Graph classification with structured prediction.** Structured prediction [67, 132], where the output is complex in that the output might be a string, a tree, a graph, multiple labels, multiple classes, is a well established research direction in the machine learning community in the past decade. However, for graph data, structured prediction has not been studied yet though it is widely seen in real-life applications.

- **Multi-task graph classification from multiple sources.** The proposed multi-task graph classification can be considered as learning from a single

source (view). For graph data, there are usually multiple views/sources available for describing the same objects [146, 147]. How to perform multi-task multi-view graph classification remains unknown.

- **Instance-based graph stream classification.** Our framework for graph stream classification employs a chunk/batch-based strategy for handling concept drifts of graph streams. How to perform graph stream classification in a instance-based manner is a challenge in this field.

- **Active learning from graph streams.** Labeling graph objects is expensive, especially in the streaming scenarios with increasing data. One direction might be employing active learning techniques [37, 38, 43, 44] to handling graph streams in the future.

Appendix A

A.1 Duality of Eq.(6.4)

The Lagrangian function of Eq.(6.4) can be written as:

$$\begin{aligned} L \quad & (\boldsymbol{\xi}, \boldsymbol{w}, \rho) = \rho - C(\beta \textstyle\sum_{\{i|y_i=+1\}}^{n^+} \delta_i \varphi_i \xi_i + \sum_{\{i|y_i=-1\}}^{n^-} \delta_i \varphi_i \xi_i) \\ & + \sum_{i=1}^{n} \mu_i \{ y_i \textstyle\sum_{j=1}^{m} w_j \cdot \hbar(G_i; g_j, \pi_j) + \xi_i - \rho \} \\ & - \gamma(\textstyle\sum_{j=1}^{m} w_j - 1) + \sum_{j=1}^{m} q_j \cdot w_j + \sum_{i=1}^{n} p_i \cdot \xi_i \end{aligned} \tag{A.1}$$

Where, we have $\mu_i \geq 0, p_i \geq 0, q_i \geq 0$, and γ can be either positive (> 0) or negative (< 0).

At optimum, the first derivative of the Lagrangian w.r.t. the primal variables ($\boldsymbol{\xi}, \boldsymbol{w}$,and ρ) must vanish,

$$\begin{array}{lcl} \frac{\partial L}{\partial \xi_{i|y_i=1}} & = & -C\beta\delta_i\varphi_i + \mu_i + p_i = 0 \Rightarrow 0 \leq \mu_i \leq C\beta\delta_i\varphi_i \\ \frac{\partial L}{\partial \xi_{i|y_i=-1}} & = & -C\delta_i\varphi_i + \mu_i + p_i = 0 \Rightarrow 0 \leq \mu_i \leq C\delta_i\varphi_i \\ \frac{\partial L}{\partial \rho} & = & 1 - \sum_{i=1}^{n} \mu_i = 0 \Rightarrow \sum_{i=1}^{n} \mu_i = 1 \\ \frac{\partial L}{\partial w_j} & \Rightarrow & \sum_{i=1}^{n} y_i \mu_i \cdot \hbar(G_i; g_j, \pi_j) - \gamma + q_j = 0 \\ & \Rightarrow & \sum_{i=1}^{n} y_i \mu_i \cdot \hbar(G_i; g_j, \pi_j) \leq \gamma \end{array}$$

Substituting these variables in Eq. (A.1), we obtain the its dual problem as Eq. (6.6).

A.2 Duality of Eq. (7.7)

The Lagrangian function of Eq.(7.7) can be written as:

$$\begin{aligned} L \quad & (\boldsymbol{\xi}, \boldsymbol{w}) \\ & = \|\boldsymbol{w}\| + \tfrac{C}{l}\{C_1 \sum_{\{i|y_i=1\}} \xi_i + \gamma \sum_{\{j|y_j=-1\}} \xi_j\} \\ & + \sum_{i|y_i=1} \mu_i\{1 - \xi_i - \sum_{k=1}^{m} w_k \cdot \hbar_{g_k}(G_i)\} \\ & + \sum_{j|y_j=-1} \mu_j\{\sum_{k=1}^{m} w_k \cdot \hbar_{g_k}(G_j) + \tfrac{1}{\gamma} - \xi_j\} \\ & - \sum_{i=1}^{m} q_k \cdot w_k - \sum_{i=1}^{l-} p_i \cdot \xi_i - \sum_{j=1}^{l+} p_j \cdot \xi_j \end{aligned} \tag{A.2}$$

Where, we have $\mu_i \geq 0, \mu_j \geq 0, p_i \geq 0, q_k \geq 0$.

At optimum, the first derivative of the Lagrangian *w.r.t.* the primal variables $(\boldsymbol{\xi}, \boldsymbol{w})$ must vanish,

$$\begin{aligned} \tfrac{\partial L}{\partial \xi_{i|y_i=1}} \quad &= \quad \tfrac{CC_1}{l} - \mu_i - p_i = 0 \Rightarrow 0 \leq \mu_i \leq \tfrac{CC_1}{l} \\ \tfrac{\partial L}{\partial \xi_{j|y_j=-1}} \quad &= \quad \tfrac{C\gamma}{l} - \mu_j - p_j = 0 \Rightarrow 0 \leq \mu_j \leq \tfrac{C\gamma}{l} \\ \tfrac{\partial L}{\partial w_k} \quad &\Rightarrow \quad 1 - \textstyle\sum_i \mu_i \hbar_{g_k}(G_i) + \sum_j \mu_j \hbar_{g_k}(G_j) - q_k = 0 \\ &\Rightarrow \quad \textstyle\sum_i \mu_i \hbar_{g_k}(G_i) - \sum_j \mu_j \hbar_{g_k}(G_j) < 1 \end{aligned}$$

Note that $\frac{\partial L}{\partial w_k}$ with respect to w_k equals to 1 because $w_k \geq 0$. Substituting these variables in Eq. (A.2), we obtain its dual problem as Eq. (7.8).

A.3 Equality of Eq. (7.7) and Eq. (7.10)

Here we will prove that given any solution $\boldsymbol{w}$ of Eq. (7.10), it will be also the solution of Eq. (7.7).

Given a $\boldsymbol{w}$, the ξ_i and ξ_j in Eq. (7.7) can be optimized independently, i.e., $\xi_i = \max(0, 1 - \boldsymbol{w}^T \boldsymbol{x}_i)$ and $\xi_j = \max(0, 1/\gamma + \boldsymbol{w}^T \boldsymbol{x}_j)$. For Eq. (7.10), the optimal

ξ for a given $\boldsymbol{w}$ is:

$$\begin{aligned}
\max_{\boldsymbol{c}\in\{0,1\}^l} \quad & \tfrac{1}{l}\{C_1 \sum_{y_i=1} c_i + \sum_{y_j=1} c_j\} \\
& -\tfrac{1}{l}\boldsymbol{w}^T\{C_1 \sum_{y_i=1} c_i\boldsymbol{x}_i - \gamma \sum_{y_j=-1} c_j\boldsymbol{x}_j\} \\
= \quad & 1/l \sum_{y_i=1} \max_{\boldsymbol{c}\in\{0,1\}^l} (C_1 c_i - C_1 c_i \boldsymbol{w}^T\boldsymbol{x}_i) \\
& +1/l \sum_{y_j=1} \max_{\boldsymbol{c}\in\{0,1\}^l} (c_j + c_j\gamma\boldsymbol{w}^T\boldsymbol{x}_j) \\
= \quad & C_1/l \sum_{y_i=1} \max(0, 1-\boldsymbol{w}^T\boldsymbol{x}_i) \\
& +\gamma/l \sum_{y_j=1} \max(0, 1/\gamma + \boldsymbol{w}^T\boldsymbol{x}_j) \\
= \quad & \{C_1 \sum_{\{i|y_i=1\}} \xi_i + \gamma \sum_{\{j|y_j=-1\}} \xi_j\}/l = \xi
\end{aligned}$$

Therefore, the objective functions of Eq. (7.7) and Eq. (7.10) are equal for any $\boldsymbol{w}$ given the optimal $\boldsymbol{\xi}$ and ξ, i.e., they are equivalent.

A.4 Duality of Eq.(7.10)

Here we derive the duality of Eq.(7.10). The Lagrangian function of Eq.(7.10) can be written as:

$$\begin{aligned}
L(\xi, \boldsymbol{w}) \quad & = \|\boldsymbol{w}\| + C\xi \\
& -\sum_{\boldsymbol{c}} \lambda_{\boldsymbol{c}}\{\tfrac{1}{l}\boldsymbol{w}^T(C_1 \sum_{y_i=1} c_i\boldsymbol{x}_i - \gamma \sum_{y_j=-1} c_j\boldsymbol{x}_j) \\
& -\tfrac{1}{l}(C_1 \sum_{y_i=1} c_i + \sum_{y_j=1} c_j) + \xi\} - \sum_{i=1}^{m} q_k \cdot w_k - p\xi
\end{aligned} \tag{A.3}$$

Where, we have $\lambda_{\boldsymbol{c}} \geq 0, p \geq 0, q_k \geq 0$.

Similarly, we take the first derivative of the Lagrangian *w.r.t.* the primal variables $(\xi, \boldsymbol{w})$,

$$\begin{aligned}
\tfrac{\partial L}{\partial \xi} \quad & = \quad C + \sum_{\boldsymbol{c}} \lambda_c - p = 0 \Rightarrow 0 \leq \sum_{\boldsymbol{c}} \lambda_c \leq C \\
\tfrac{\partial L}{\partial w_k} \quad & \Rightarrow \quad 1 - \tfrac{C_1}{l} \sum_{\boldsymbol{c}} \lambda_{\boldsymbol{c}} \sum_i c_i \boldsymbol{x}_i^k + \tfrac{\gamma}{l} \sum_{\boldsymbol{c}} \lambda_{\boldsymbol{c}} \sum_j c_j \boldsymbol{x}_j^k - q_k = 0 \\
& \Rightarrow \quad \tfrac{C_1}{l} \sum_{\boldsymbol{c}} \lambda_{\boldsymbol{c}} \sum_i c_i \boldsymbol{x}_i^k - \tfrac{\gamma}{l} \sum_{\boldsymbol{c}} \lambda_{\boldsymbol{c}} \sum_i c_j \boldsymbol{x}_j^k < 1
\end{aligned}$$

Substituting these variables in Eq. (A.3), we obtain the its dual problem as Eq. (7.11).

References

[1] Naoki Abe, Bianca Zadrozny, and John Langford. An iterative method for multi-class cost-sensitive learning. In *Proc. ACM KDD*, pages 3–11, 2004. (Cited on page 138.)

[2] C Aggarwal. On Classification of Graph Streams. In *Proc. of SDM*, Arizona, USA, 2011. (Cited on pages 1, 19, 100, 118, and 119.)

[3] Charu C Aggarwal. *Data streams: models and algorithms*, **31**. Springer Science & Business Media, 2007. (Cited on page 1.)

[4] Charu C Aggarwal. The setwise stream classification problem. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 432–441. ACM, 2014. (Cited on page 18.)

[5] Charu C Aggarwal. A survey of stream classification algorithms. *Data Classification: Algorithms and Applications*, page 245, 2014. (Cited on pages 1 and 18.)

[6] R Akbani, S Kwek, and N Japkowicz. Applying support vector machines to imbalanced datasets. *Machine Learning: ECML 2004*, pages 39–50, 2004. (Cited on pages 17, 18, and 100.)

[7] P Anchuri, M Zaki, O Barkol, S Golan, and M Shamy. Approximate graph mining with label costs. In *ACM SIGKDD*, pages 518–526, 2013. (Cited on pages 2, 5, and 168.)

[8] Andreas Argyriou, Theodoros Evgeniou, and Massimo Pontil. Convex Multi-Task Feature Learning. *SSRN Electronic Journal*, 2007. (Cited on pages 20, 169, 171, 175, and 186.)

[9] Francis R Bach, David Heckerman, and Eric Horvitz. Considering cost asymmetry in learning classifiers. *The Journal of Machine Learning Research*, **7**:1713–1741, 2006. (Cited on pages 107, 138, and 157.)

[10] Manuel Baena-García, José del Campo-Ávila, Raúl Fidalgo, Albert Bifet, Ricard Gavaldà, and Rafael Morales-Bueno. Early drift detection method. In *Proc. of ECML/PKDD*, 2006. (Cited on page 18.)

[11] Ricardo A. Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999. (Cited on page 56.)

[12] Lu Bai, Luca Rossi, Andrea Torsello, and Edwin R Hancock. A quantum Jensen-Shannon graph kernel for unattributed graphs. *Pattern Recognition*, **48**[2]:344–355, 2015. (Cited on page 14.)

[13] Dimitris Bertsimas and John N Tsitsiklis. *Introduction to linear optimization*, chapter 4. Athena Scientific Belmont, 1973. (Cited on page 109.)

[14] M.A. Bhuiyan and M. Al Hasan. An iterative mapreduce based frequent subgraph mining algorithm. *Knowledge and Data Engineering, IEEE Transactions on*, **27**[3]:608–620, March 2015. (Cited on page 78.)

[15] Albert Bifet, Geoff Holmes, Bernhard Pfahringer, and Ricard Gavaldà. Mining frequent closed graphs on evolving data streams. In *KDD*, pages 591–599, 2011. (Cited on pages 14 and 38.)

[16] Christian Borgelt and Michael R Berthold. Mining molecular fragments: Finding relevant substructures of molecules. In *Data Mining, 2002. ICDM 2003. Proceedings. 2002 IEEE International Conference on*, pages 51–58. IEEE, 2002. (Cited on page 158.)

[17] Christian Borgelt, Thorsten Meinl, and Michael Berthold. Moss: a program for molecular substructure mining. In *Proceedings of the 1st international workshop on open source data mining: frequent pattern mining implementations*, pages 6–15. ACM, 2005. (Cited on page 158.)

[18] Sergey Brin, Rajeev Motwani, and Craig D. Silverstein. Beyond Market Baskets: Generalizing Association Rules to Correlations. In *ACM SIGMOD Record*, pages 265–276, 1997. (Cited on page 13.)

[19] C. Bron and J. Kerbosch. Algorithm 457: finding all cliques of an undirected graph. *C. of the ACM*, **16**[9]:575–577, 1973. (Cited on page 82.)

[20] Florian Cajori. *A history of mathematical notation*, **1-2**. New York: Dover, 1993. (Cited on page 48.)

[21] Rich Caruana. Multitask learning. *Mach. Learn.*, **28**[1]:41–75, July 1997. (Cited on page 20.)

[22] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. SMOTE: synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, **16**[1]:321–357, 2002. (Cited on page 18.)

[23] Sheng Chen and Haibo He. Towards incremental learning of nonstationary imbalanced data stream: a multiple selectively recursive approach. *Evolving Systems*, **2**[1]:35–50, 2011. (Cited on page 18.)

[24] H Cheng, X Yan, and J Han. Discriminative Frequent Pattern-Based Graph Classification. *Link Mining: Models, Algorithms, and Applications*, pages 237–262, 2010. (Cited on pages 14 and 15.)

[25] Lianhua Chi, Bin Li, and and Xingquan Zhu. Fast Graph Stream Classification Using Discriminative Clique Hashing. In *Proc of the 17th Pacific-Asia Conf on Knowledge Discovery and Data Mining (PAKDD)*, 2013. (Cited on pages 19, 118, and 119.)

[26] R Craddock, C James, P Holtzheimer, X Hu, and H Mayberg. A whole brain fMRI atlas generated via spatially constrained spectral clustering. *Human Brain Mapping*, **33**, 2012. (Cited on page 167.)

[27] M Culp and G Michailidis. Graph-Based semisupervised learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **30**[1]:174–179, 2008. (Cited on page 17.)

[28] A Demiriz, K P Bennett, and J Shawe-Taylor. Linear programming boosting via column generation. *Machine Learning*, pages 225–254, 2002. (Cited on page 195.)

[29] M Deshpande, M Kuramochi, N Wale, and G Karypis. Frequent Substructure-based Approaches for Classifying Chemical Compounds. *IEEE Trans. on Knowl. and Data Eng.*, **17**:1036–1050, 2005. (Cited on pages 1, 4, 17, 23, 99, and 137.)

[30] Chris Ding and Hanchuan Peng. Minimum redundancy feature selection from microarray gene expression data. *Journal of bioinformatics and computational biology*, **3**[02]:185–205, 2005. (Cited on page 75.)

[31] Gregory Ditzler and Robi Polikar. Incremental Learning of Concept Drift from Streaming Imbalanced Data. *IEEE Transactions on Knowledge and Data Engineering*, 2013. (Cited on pages 18, 119, and 120.)

[32] Pedro Domingos. MetaCost: a general method for making classifiers cost-sensitive. In *Proc. of ACM KDD*, pages 155–164, 1999. (Cited on page 138.)

[33] Pedro Domingos and Geoff Hulten. Mining high-speed data streams. In *Proc. of the 6th ACM KDD*, pages 71–80. ACM, 2000. (Cited on page 18.)

[34] Charles Elkan. The foundations of cost-sensitive learning. In *International joint conference on artificial intelligence*, pages 973–978. Citeseer, 2001. (Cited on page 138.)

[35] Theodoros Evgeniou and Massimiliano Pontil. Regularized multi–task learning. In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, pages 109–117. ACM, 2004. (Cited on pages 20, 169, and 171.)

[36] Wei Fan, Salvatore J Stolfo, Junxin Zhang, and Philip K Chan. AdaCost: misclassification cost-sensitive boosting. In *ICML*, pages 97–105, 1999. (Cited on page 19.)

[37] Meng Fang, Jie Yin, and Dacheng Tao. Active learning for crowdsourcing using knowledge transfer. In *Twenty-Eighth AAAI Conference on Artificial Intelligence*, 2014. (Cited on page 201.)

[38] Meng Fang, Jie Yin, Chengqi Zhang, and Xingquan Zhu. Active class discovery and learning for networked data. In *Proceedings of the 13th SIAM International Conference on Data Mining, May 2-4, 2013. Austin, Texas, USA.*, pages 315–323, 2013. (Cited on page 201.)

[39] Meng Fang, Jie Yin, and Xingquan Zhu. Transfer learning across networks for collective classification. In *Data Mining (ICDM), 2013 IEEE 13th International Conference on*, pages 161–170. IEEE, 2013. (Cited on page 20.)

[40] H Fei and J Huan. Boosting with Structure Information in the Functional Space: an Application to Graph Classification. In *Proc. of ACM SIGKDD*, Washington DC, USA, 2010. (Cited on pages 1, 14, 99, 137, and 172.)

[41] Hongliang Fei and Jun Huan. Structured feature selection and task relationship inference for multi-task learning. *Knowledge and information systems*, **35**[2]:345–364, 2013. (Cited on page 20.)

[42] Y Freund and R Schapire. A desicion-theoretic generalization of on-line learning and an application to boosting. In *Computational Learning Theory*, pages 23–37. Springer, 1995. (Cited on pages 84, 86, and 115.)

[43] Yifan Fu, Bin Li, Xingquan Zhu, and Chengqi Zhang. Active learning without knowing individual instance labels: A pairwise label homogene-

ity query approach. *Knowledge and Data Engineering, IEEE Transactions on*, **26**[4]:808–822, 2014. (Cited on page 201.)

[44] Yifan Fu, Xingquan Zhu, and Ahmed K. Elmagarmid. Active learning with optimal instance subset selection. *IEEE T. Cybernetics*, **43**[2]:464–475, 2013. (Cited on page 201.)

[45] Mikel Galar, Alberto Fernández, Edurne Barrenechea, Humberto Bustince, and Francisco Herrera. A review on ensembles for the class imbalance problem: bagging-, boosting-, and hybrid-based approaches. *IEEE Trans. SMC-Part C*, **42**[4]:463–484, 2012. (Cited on pages 17 and 100.)

[46] Jing Gao, Wei Fan, Jiawei Han, and S Yu Philip. A General Framework for Mining Concept-Drifting Data Streams with Skewed Distributions. In *Proc. SIAM Int'l Conf. Data Mining*, 2007. (Cited on page 18.)

[47] Alec Go, Richa Bhayani, and Lei Huang. Twitter sentiment classification using distant supervision. *CS224N Project Report, Stanford*, pages 1–12, 2009. (Cited on page 31.)

[48] Pinghua Gong, Jieping Ye, and Changshui Zhang. Robust multi-task feature learning. In *ACM SIGKDD*, pages 895–903. ACM, 2012. (Cited on page 20.)

[49] Pinghua Gong, Jiayu Zhou, Wei Fan, and Jieping Ye. Efficient multi-task feature learning with calibration. In *ACM SIGKDD*, pages 761–770. ACM, 2014. (Cited on page 20.)

[50] Michael Grant and Stephen Boyd. Graph implementations for non-smooth convex programs. In V. Blondel, S. Boyd, and H. Kimura, editors, *Recent Advances in Learning and Control*, Lecture Notes in Control and Information Sciences, pages 95–110. Springer-Verlag Limited, 2008. http://stanford.edu/~boyd/graph_dcp.html. (Cited on pages 112 and 158.)

[51] Michael Grant and Stephen Boyd. CVX: Matlab software for disciplined convex programming, version 2.1. http://cvxr.com/cvx, March 2014. (Cited on pages 112 and 158.)

[52] Ting Guo, Lianhua Chi, and Xingquan Zhu. Graph hashing and factorization for fast graph stream classification. In *Proceedings of the 22nd ACM international conference on Conference on information & knowledge management*, pages 1607–1612. ACM, 2013. (Cited on page 19.)

[53] Ting Guo, Xingquan Zhu, Jian Pei, and Chengqi Zhang. Snoc: Streaming network node classification. In *Data Mining (ICDM), 2014 IEEE International Conference on*, pages 150–159. IEEE, 2014. (Cited on page 19.)

[54] Isabelle Guyon and André Elisseeff. An introduction to variable and feature selection. *Journal of Machine Learning Research*, **3**:1157–1182, 2003. (Cited on page 16.)

[55] H He and E A Garcia. Learning from imbalanced data. *IEEE TKDE*, **21**[9]:1263–1284, 2009. (Cited on page 17.)

[56] Haibo He and Yunqian Ma. *Imbalanced Learning: Foundations, Algorithms, and Applications*. Wiley-IEEE Press, 2013. (Cited on pages 17 and 100.)

[57] Zhibin Hong, Xue Mei, Danil Prokhorov, and Dacheng Tao. Tracking via Robust Multi-task Multi-view Joint Sparse Representation. In *Proc. of ICCV*, pages 649–656. IEEE, 2013. (Cited on page 20.)

[58] Jun Huan, Wei Wang, and Jan Prins. Efficient mining of frequent subgraphs in the presence of isomorphism. In *Data Mining, 2003. ICDM 2003. Third IEEE International Conference on*, pages 549–552. IEEE, 2003. (Cited on page 78.)

[59] Lawrence J. Hubert. Matching models in the analysis of cross-classifications. *Psychometrika*, **44**:21–41, 1979. (Cited on pages 25, 37, 41, and 42.)

[60] Geoff Hulten, Laurie Spencer, and Pedro Domingos. Mining time-changing data streams. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 97–106. ACM, 2001. (Cited on page 1.)

[61] Laurent Jacob, Jean-philippe Vert, and Francis R Bach. Clustered multi-task learning: A convex formulation. In *Advances in neural information processing systems*, pages 745–752, 2009. (Cited on page 20.)

[62] Brijnesh J Jain and Klaus Obermayer. Structure spaces. *Journal of Machine Learning Research*, **10**:2667–2714, 2009. (Cited on page 14.)

[63] Brijnesh J Jain and Klaus Obermayer. Learning in Riemannian Orbifolds. *arXiv preprint arXiv:1204.4294 (2012)*, 2012. (Cited on page 14.)

[64] Ali Jalali, Sujay Sanghavi, Chao Ruan, and Pradeep K Ravikumar. A dirty model for multi-task learning. In *NIPS*, 2010. (Cited on page 20.)

[65] N Jin, C Young, and W Wang. In *Proc. of*, Hong Kong, China. (Cited on page 14.)

[66] Thorsten Joachims. Training linear SVMs in linear time. In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, pages 217–226, 2006. (Cited on pages 22, 151, and 155.)

[67] Thorsten Joachims, Thomas Finley, and Chun-Nam John Yu. Cutting-plane training of structural svms. *Machine Learning*, **77**[1]:27–59, 2009. (Cited on page 200.)

[68] Zhuoliang Kang, Kristen Grauman, and Fei Sha. Learning with whom to share in multi-task feature learning. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 521–528, 2011. (Cited on page 20.)

[69] M Karasuyama and H Mamitsuka. Multiple graph label propagation by sparse integration. *IEEE Transactions on Neural Networks and Learning Systems*, **24**[12]:1999–2012, 2013. (Cited on page 17.)

[70] H Kashima, K Tsuda, and A Inokuchi. *Kernels for Graphs*, chapter In: Schölk, pages 155–170. MIT Press, Cambridge (Massachusetts), 2004. (Cited on pages 14, 15, 69, and 99.)

[71] Yiping Ke, James Cheng, and Ng Wilfred. Correlation search in graph databases. In *KDD*, pages 390–399, 2007. (Cited on pages 13, 37, 38, 44, and 56.)

[72] Yiping Ke, James Cheng, and Ng Wilfred. Correlated pattern mining in quantitative databases. *ACM Trans. on Database Systems*, **33**:1–45, 2008. (Cited on page 37.)

[73] Yiping Ke, James Cheng, and Jeffrey Xu Yu. Efficient Discovery of Frequent Correlated Subgraph Pairs. In *ICDM*, pages 239–248, 2009. (Cited on page 13.)

[74] Yiping Ke, James Cheng, and Jeffrey Xu Yu. Top-k Correlative Graph Mining. In *SIAM International Conf. on Data Mining*, pages 1038–1049, 2009. (Cited on pages 3, 13, 37, and 38.)

[75] X Kong and P Yu. Semi-Supervised Feature Selection for Graph Classification. In *Proc. of ACM SIGKDD*, Washington, DC, USA, 2010. (Cited on pages 14, 16, 17, 78, 88, 99, 100, 137, 154, 157, and 160.)

[76] Xiangnan Kong and Philip S Yu. Multi-label feature selection for graph classification. In *IEEE International Conference on Data Mining*, pages 274–283. IEEE, 2010. (Cited on pages 6, 14, and 16.)

[77] Taku Kudo, Eisaku Maeda, and Yuji Matsumoto. An Application of Boosting to Graph Classification. In *Neural Information Processing Systems (NIPS)*, pages 729–736, 2004. (Cited on page 32.)

[78] Abhishek Kumar and Hal Daume. Learning Task Grouping and Overlap in Multi-task Learning. In *Proc. of ICML*, pages 1383–1390, 2012. (Cited on page 20.)

[79] Michihiro Kuramochi and George Karypis. Frequent subgraph discovery. In *Data Mining, 2001. ICDM 2001, Proceedings IEEE International Conference on*, pages 313–320. IEEE, 2001. (Cited on page 78.)

[80] J Leskovec and J Shawe-Taylor. Linear programming boosting for uneven datasets. In *Proc. of ICML*, page 456, 2003. (Cited on pages 17 and 100.)

[81] B Li, X Zhu, L Chi, and C Zhang. Nested Subtree Hash Kernels for Large-Scale Graph Classification over Streams. *Data Mining (ICDM), 2012 IEEE ...*, 2012. (Cited on page 19.)

[82] David Liben-Nowell and Jon Kleinberg. The link-prediction problem for social networks. *Journal of the American Society for Information Science and Technology*, **58**[7]:1019–1031, 2007. (Cited on page 17.)

[83] Jun Liu, Shuiwang Ji, and Jieping Ye. Multi-task feature learning via efficient l 2, 1-norm minimization. In *Proc. of UAI*, pages 339–348. AUAI Press, 2009. (Cited on page 20.)

[84] X Liu, L Wang, J Zhang, J Yin, and H Liu. Global and local structure preservation for feature selection. *IEEE Transactions on Neural Networks and Learning Systems*, **25**[6]:1083–1095, 2014. (Cited on page 17.)

[85] X Y Liu, J Wu, and Z H Zhou. Exploratory undersampling for class-imbalance learning. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, **39**[2]:539–550, 2009. (Cited on pages 17 and 100.)

[86] Susan Lomax and Sunil Vadera. A survey of cost-sensitive decision tree induction algorithms. *ACM Computing Surveys*, **45**[2], 2013. (Cited on pages 19 and 138.)

[87] Aurélie C Lozano and Naoki Abe. Multi-class cost-sensitive boosting with p-norm loss functions. In *Proc. of ACM KDD*, pages 506–514, 2008. (Cited on pages 19 and 138.)

[88] D G Luenberger. *Optimization by vector space methods*. Wiley-Interscience, 1997. (Cited on page 109.)

[89] Sheng Ma and Joseph L. Hellerstein. Mining Mutually Dependent Patterns. In *ICDM*, pages 409–416, 2001. (Cited on page 13.)

[90] Amin Mantrach, Nicolas van Zeebroeck, Pascal Francq, Masashi Shimbo, Hugues Bersini, and Marco Saerens. Semi-supervised classification and betweenness computation on large, sparse, directed graphs. *Pattern Recognition*, **44**[6]:1212–1224, 2011. (Cited on page 17.)

[91] H Masnadi-Shirazi and N Vasconcelos. Risk minimization, probability elicitation, and cost-sensitive SVMs. *ICML*, 2010. (Cited on pages x, 19, 20, 138, 145, 146, and 165.)

[92] Hamed Masnadi-Shirazi and Nuno Vasconcelos. Cost-sensitive boosting. *IEEE PAMI*, **33**[2]:294–309, 2011. (Cited on pages 19 and 138.)

[93] Hamed Masnadi-Shirazi, Nuno Vasconcelos, and Arya Iranmehr. Cost-Sensitive Support Vector Machines. *arXiv:1212.0975*, 2012. (Cited on pages 144, 147, and 157.)

[94] Nimrod Megiddo. On the complexity of linear programming. *Advances in economic theory*, pages 225–268, 1987. (Cited on page 155.)

[95] X. Mei, Z. Hong, D. Prokhorov, and D. Tao. Robust multitask multiview tracking in videos. *Neural Networks and Learning Systems, IEEE Transactions on*, **PP**[99]:1–1, 2015. (Cited on page 20.)

[96] Abdullah Mueen, Suman Nath, and Jie Liu. Fast approximate correlation for massive time-series data. In *SIGMOD*, pages 171–182, 2010. (Cited on page 37.)

[97] S G Nash and A Sofer. Linear and Nonlinear Programming. *Newyark McGraw-Hill*, 1996. (Cited on pages 21, 108, and 148.)

[98] Marion Neumann, Novi Patricia, Roman Garnett, and Kristian Kersting. Efficient graph kernels by randomization. In *Machine Learning and Knowledge Discovery in Databases*, pages 378–393. Springer, 2012. (Cited on page 14.)

[99] M.E.J. Newman and M. Girvan. Finding and evaluating community structure in networks. *Physical review E*, **69**[2]:026113, 2004. (Cited on page 81.)

[100] C H Nguyen and H Mamitsuka. Latent feature kernels for link prediction on sparse graphs. *IEEE Transactions on Neural Networks and Learning Systems*, **23**[11]:1793–1804, 2012. (Cited on page 17.)

[101] S Pan, X Zhu, C Zhang, and P S Yu. Graph Stream Classification using Labeled and Unlabeled Graphs. In *2013 IEEE 29th International Conference on Data Engineering (ICDE)*, pages 398–409. IEEE, 2013. (Cited on pages 8, 9, 19, 23, 29, 31, 100, 115, and 137.)

[102] Shirui Pan, Jia Wu, and Xingquan Zhu. CogBoost: Boosting for Fast Cost-sensitive Graph Classification. *IEEE TKDE*, 2015. (Cited on pages 8, 9, 16, 29, and 31.)

[103] Shirui Pan, Jia Wu, Xingquan Zhu, Guodong Long, and Chengqi Zhang. Finding the best not the most: Regularized loss minimization subgraph selection for graph classification. *Pattern Recognition*, **48**[11]:3783–3796, 2015. (Cited on pages 8 and 16.)

[104] Shirui Pan, Jia Wu, Xingquan Zhu, and Chengqi Zhang. Graph Ensemble Boosting for Imbalanced Noisy Graph Stream Classification. *IEEE Transactions on Cybernetics*, **45**[5]:940–954, 2015. (Cited on pages 8, 9, 16, 29, and 31.)

[105] Shirui Pan, Kuan Wu, Yang Zhang, and Xue Li. Classifier ensemble for uncertain data stream classification. In *Advances in knowledge discovery and data mining*, pages 488–495. Springer, 2010. (Cited on page 18.)

[106] Shirui Pan, Yang Zhang, and Xue Li. Dynamic classifier ensemble for positive unlabeled text stream classification. *Knowledge and information systems*, **33**[2]:267–287, 2012. (Cited on page 18.)

[107] Shirui Pan and Xingquan Zhu. CGStream: continuous correlated graph query for data streams. In *Proceedings of the 21st ACM international*

conference on Information and knowledge management, pages 1183–1192. ACM, 2012. (Cited on pages 7, 9, and 100.)

[108] Shirui Pan and Xingquan Zhu. Continuous top-k query for graph streams. In *Proceedings of the 21st ACM international conference on Information and knowledge management*, pages 2659–2662. ACM, 2012. (Cited on pages 7 and 100.)

[109] Shirui Pan and Xingquan Zhu. Graph Classification with Imbalanced Class Distributions and Noise. In *International Joint Conferences on Artificial Intelligence*, pages 1586–1592, 2013. (Cited on pages ix, 8, 9, 16, 138, 139, 140, 144, 146, 155, 156, and 157.)

[110] Shirui Pan, Xingquan Zhu, and Meng Fang. Top-k correlated subgraph query for data streams. In *Pattern Recognition (ICPR), 2012 21st International Conference on*, pages 2906–2909. IEEE, 2012. (Cited on pages 7 and 100.)

[111] Shirui Pan, Xingquan Zhu, and Chengqi Zhang. Imblanced noisy graph stream classification: Results and source code. Technical report, University of Technology Sydney, 2014. (Cited on page 117.)

[112] Sinno Jialin Pan and Qiang Yang. A Survey on Transfer Learning. *IEEE Trans. Knowl. Data Eng.*, pages 1345–1359, 2010. (Cited on page 20.)

[113] Hanchuan Peng, Fuhui Long, and Chris Ding. Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, **27**[8]:1226–1238, 2005. (Cited on page 75.)

[114] S. Petrovic, M. Osborne, and V. Lavrenko. Streaming first story detection with application to twitter. In *Proc. of Human Language Technologies*, Stroudsburg, USA, 2010. (Cited on page 1.)

[115] J Ross Quinlan. C4. 5: Programs for machine learning. 1993. (Cited on page 88.)

[116] S. Raghavan and H. Garcia-Molina. Representing web graphs. In *Proc. of ICDE*, Atlanta, USA, 2003. (Cited on page 1.)

[117] Sayan Ranu and Ambuj K Singh. Graphsig: A scalable approach to mining significant subgraphs in large graph databases. In *Proc. of ICDE*, pages 844–855. IEEE, 2009. (Cited on pages 14, 15, 16, 17, and 100.)

[118] K Riesen and H Bunke. Graph Classification by Means of Lipschitz Embedding. *IEEE Transactions on SMC, Part B: Cybernetics*, **39**:1472–1483, 2009. (Cited on page 14.)

[119] Bernardino Romera-Paredes, Andreas Argyriou, Nadia Berthouze, and Massimiliano Pontil. Exploiting unrelated tasks in multi-task learning. In *AI Statistics*, pages 951–959, 2012. (Cited on page 20.)

[120] H Saigo, S Nowozin, T Kadowaki, T Kudo, and K Tsuda. gBoost: a Mathematical Programming Approach to Graph Classification and Regression. *Machine Learning*, **75**:69–89, 2009. (Cited on pages ix, 16, 18, 99, 105, 106, 107, 112, 114, 119, 137, 138, 139, 140, 143, 144, 145, 150, 151, 154, 155, 156, 157, 160, 161, 168, 169, 170, 171, 172, 173, 186, 187, and 195.)

[121] Grigoris Karakoulas John Shawe-Taylor. Optimizing classifiers for imbalanced training sets. In *NIPS*, **11**, page 253. MIT Press, 1999. (Cited on pages 20 and 138.)

[122] Nino Shervashidze, Pascal Schweitzer, Erik Jan Van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. Weisfeiler-Lehman graph kernels. *Journal of Machine Learning Research*, **12**:2539–2561, 2011. (Cited on page 14.)

[123] JG Skellam. The frequency distribution of the difference between two poisson variates belonging to different populations. *Journal of the Royal Statistical Society. Series A (General)*, **109**[Pt 3]:296, 1946. (Cited on page 48.)

[124] W Nick Street and YongSeog Kim. A streaming ensemble algorithm (SEA) for large-scale classification. In *Proc. of 7th KDD*, pages 377–382, 2001. (Cited on page 18.)

[125] Y Sun, A K C Wong, and S K MOHAMED. Classification of imbalanced data: A review. *International Journal of Pattern Recognition and Artificial Intelligence*, **23**[04]:687–719, 2009. (Cited on page 17.)

[126] J Tang and H Liu. An Unsupervised Feature Selection Framework for Social Media Data. *IEEE Transactions on Knowledge and Data Engineering*, **26**[12]:2914–2927, 2014. (Cited on page 137.)

[127] J Tang, J Zhang, L Yao, J Li, L Zhang, and Z Su. Arnetminer: Extraction and mining of academic social networks. In *Proceeding of ACM SIGKDD*, pages 990–998, 2008. (Cited on page 29.)

[128] A Teneva, S Markovska-Simoskab, L Kocarevb, J Pop-Jordanovb, A Mullerc, and G Candrianc. Machine learning approach for classification of ADHD adults. *International Journal of Psychophysiology*, **93**, 2014. (Cited on page 168.)

[129] M Thoma, H Cheng, A Gretton, J Han, H Kriegel, A Smola, L Song, P Yu, X Yan, and K Borgwardt. Near-Optimal Supervised Feature Selection among Frequent Subgraphs. In *Proc. of SDM*, USA, 2009. (Cited on pages 14, 15, 16, and 154.)

[130] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288, 1996. (Cited on pages 146 and 174.)

[131] P Tiwari, J Kurhanewicz, and A Madabhushi. Multi-kernel graph embedding for detection, Gleason grading of prostate cancer via MRI/MRS. *Medical Image Analysis*, **17**[2]:219–235, 2013. (Cited on pages 4 and 137.)

[132] Ioannis Tsochantaridis, Thorsten Joachims, Thomas Hofmann, and Yasemin Altun. Large margin methods for structured and interde-

pendent output variables. In *Journal of Machine Learning Research*, pages 1453–1484, 2005. (Cited on page 200.)

[133] Alexey Tsymbal. The problem of concept drift: definitions and related work. *Computer Science Department, Trinity College Dublin*, **106**, 2004. (Cited on page 2.)

[134] Jason Van Hulse and Taghi Khoshgoftaar. Knowledge discovery from imbalanced and noisy data. *Data & Knowledge Engineering*, **68**[12]:1513–1542, 2009. (Cited on page 18.)

[135] K Veropoulos, C Campbell, N Cristianini, and Others. Controlling the sensitivity of support vector machines. In *Proc. of the IJCAI*, **1999**, pages 55–60. Citeseer, 1999. (Cited on pages 17, 19, 20, 100, 107, 138, and 157.)

[136] J Vogelstein, W Roncal, R Vogelstein, and C Priebe. Graph classification using signal-subgraphs: Applications in statistical connectomics. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **35**[7]:1539–1551, 2013. (Cited on pages 17 and 168.)

[137] K. Wagstaff, C. Cardie, S. Rogers, and S. Schroedl. Constrained k-means clustering with background knowledge. In *Proc. of ICML*, pages 577–584, 2001. (Cited on page 75.)

[138] Changliang Wang and Lei Chen. Continuous Subgraph Pattern Search over Graph Streams. In *ICDE*, pages 393–404, 2009. (Cited on pages 14 and 38.)

[139] Haixun Wang, Wei Fan, Philip S Yu, and Jiawei Han. Mining concept-drifting data streams using ensemble classifiers. In *Proc. of the 9th ACM KDD*, pages 226–235. ACM, 2003. (Cited on page 18.)

[140] Shuo Wang, Leandro L Minku, and Xin Yao. A learning framework for online class imbalance learning. In *IEEE Symposium Series on Computational Intelligence*, 2013. (Cited on page 18.)

[141] Gerhard Widmer and Miroslav Kubat. Learning in the presence of concept drift and hidden contexts. *Machine learning*, **23**[1]:69–101, 1996. (Cited on page 2.)

[142] J Wu, Zhibin Hong, Shirui Pan, Xingquan Zhu, Chengqi Zhang, and Zhihua Cai. Multi-Graph Learning with Positive and Unlabeled Bags. *Proc of SDM*, pages 217–225, 2014. (Cited on pages 8 and 16.)

[143] J Wu, S Pan, X Zhu, and Z Cai. Boosting for Multi-Graph Classification. *IEEE Transactions on Cybernetics*, **45**[3]:430–443, 2015. (Cited on pages 8 and 16.)

[144] J Wu, X Zhu, C Zhang, and P Yu. Bag Constrained Structure Pattern Mining for Multi-Graph Classification. *IEEE Transactions on Knowledge and Data Engineering*, **26**[10]:2382–2396, 2014. (Cited on pages 7, 99, and 137.)

[145] Jia Wu, Zhibin Hong, Shirui Pan, Xingquan Zhu, Zhihua Cai, and Chengqi Zhang. Exploring features for complicated objects: Cross-view feature selection for multi-instance learning. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*, pages 1699–1708. ACM, 2014. (Cited on page 8.)

[146] Jia Wu, Zhibin Hong, Shirui Pan, Xingquan Zhu, and Chengqi Zhang. Multi-Graph-View Learning for Graph Classification. In *International Conference on Data Mining*, pages 590–599, 2014. (Cited on pages 8, 16, and 201.)

[147] Jia Wu, Shirui Pan, Xingquan Zhu, and Chengqi Zhang. Multi-Graph-View Learning for Complicated Object Classification. In *International Joint Conferences on Artificial Intelligence*, 2015. (Cited on page 201.)

[148] X Wu, X Zhu, G Wu, and W Ding. Data Mining with Big Data. *IEEE TKDE*, **26**[1]:97–107, 2014. (Cited on pages 2, 135, and 140.)

[149] Hui Xiong, Pang ning Tan, and Vipin Kumar. Hyperclique pattern discovery. *Data Mining and Knowledge Discovery*, **13**:219–242, 2006. (Cited on page 13.)

[150] Hui Xiong, Shashi Shekhar, Pang-Ning Tan, and Vipin Kumar. Exploiting a support-based upper bound of Pearson's correlation coefficient for efficiently identifying strongly correlated pairs. In *KDD*, pages 334–343, 2004. (Cited on pages 13, 25, 37, and 42.)

[151] X F Yan and J W Han. gSpan: Graph-Based Substructure Pattern Mining. In V Kumar, S Tsumoto, N Zhong, P S Yu, and X D Wu, editors, *Proc. of ICDM*, pages 721–724, Maebashi City, Japan, 2002. UIUC. (Cited on pages 21, 55, 78, 112, 140, 150, 169, 182, and 200.)

[152] Xifeng Yan, Hong Cheng, Jiawei Han, and Philip S Yu. Mining significant graph patterns by leap search. In *ACM SIGMOD Conference*, pages 433–444. ACM, 2008. (Cited on pages 14 and 16.)

[153] Xifeng Yan and Jiawei Han. gSpan: Graph-Based Substructure Pattern Mining. In *ICDM*, pages 721–724, 2002. (Cited on page 43.)

[154] 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*, pages 286–295. ACM, 2003. (Cited on page 78.)

[155] Ming Yuan and Yi Lin. Model selection and estimation in regression with grouped variables. *J. of the Royal Statistical Society: Series B*, pages 49–67, 2006. (Cited on pages 175 and 188.)

[156] Bianca Zadrozny, John Langford, and Naoki Abe. Cost-sensitive learning by cost-proportionate example weighting. In *Proc. of IEEE ICDM*, 2003. (Cited on pages 19 and 138.)

[157] Peng Zhang, Jun Li, Peng Wang, Byron J Gao, Xingquan Zhu, and Li Guo. Enabling fast prediction for ensemble models on data

streams. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 177–185. ACM, 2011. (Cited on page 18.)

[158] Peng Zhang, Chuan Zhou, Peng Wang, Byron J Gao, Xingquan Zhu, and Li Guo. E-tree: An efficient indexing structure for ensemble models on data streams. *Knowledge and Data Engineering, IEEE Transactions on*, **27**[2]:461–474, 2015. (Cited on page 18.)

[159] Peng Zhang, Xingquan Zhu, Yong Shi, Li Guo, and Xindong Wu. Robust ensemble learning for mining noisy data streams. *Decision Support Systems*, **50**[2]:469–479, 2011. (Cited on page 18.)

[160] Shichao Zhang, Zhenxing Qin, Charles X Ling, and Shengli Sheng. Missing is useful": missing values in cost-sensitive decision trees. *IEEE TKDE*, **17**[12]:1689–1693, 2005. (Cited on pages 19 and 138.)

[161] Yu Zhang and Dit-Yan Yeung. A Convex Formulation for Learning Task Relationships in Multi-Task Learning. In *UAI'10*, pages 442–733, 2010. (Cited on page 20.)

[162] Yu Zhang and Dit-Yan Yeung. Multi-Task boosting by exploiting task relationships. In *ECML/PKDD*, pages 697–710. Springer-Verlag, 2012. (Cited on page 20.)

[163] Wenliang Zhong and James T Kwok. Convex Multitask Learning with Flexible Task Clusters. In *Proc. of ICML*, pages 49–56, 2012. (Cited on page 20.)

[164] Jiayu Zhou, Jianhui Chen, and Jieping Ye. MALSAR: Multi-tAsk Learning via StructurAl Regularization. *Arizona State Univ*, 2012. (Cited on pages 20, 175, and 186.)

[165] Wenjun Zhou and Hui Xiong. Volatile correlation computation: a checkpoint view. In *KDD*, pages 848–856, 2008. (Cited on pages 13, 37, and 42.)

[166] X Zhu, P Zhang, X Lin, and Y Shi. Active Learning From Stream Data Using Optimal Weight Classifier Ensemble. *IEEE Trans. on SMC - B*, **40**:1607–1621, 2010. (Cited on page 18.)

[167] Xingquan Zhu and Xindong Wu. Class noise handling for effective cost-sensitive learning by cost-guided iterative classification filtering. *IEEE TKDE*, **18**[10]:1435–1440, 2006. (Cited on page 138.)

[168] Y Zhu, J X Yu, H Cheng, and L Qin. Graph classification: A diversified discriminative feature selection approach. In *Conference on Information and Knowledge Management (CIKM)*, pages 205–214. ACM, 2012. (Cited on pages 14, 99, and 137.)