# Graph Ensemble Boosting for Imbalanced Noisy Graph Stream Classification

Shirui Pan, Jia Wu, Xingquan Zhu, *Senior Member, IEEE*, and Chengqi Zhang, *Senior Member, IEEE*

***Abstract*—Many applications involve stream data with structural dependency, graph representations, and continuously increasing volumes. For these applications, it is very common that their class distributions are imbalanced with minority (or positive) samples being only a small portion of the population, which imposes significant challenges for learning models to accurately identify minority samples. This problem is further complicated with the presence of noise, because they are similar to minority samples and any treatment for the class imbalance may incorrectly focus on the noise and result in deterioration of accuracy. In this paper, we propose a classification model to tackle imbalanced graph streams with noise. Our method, graph ensemble boosting (gEBoost), employs an ensemble based framework to partition graph stream into chunks each containing a number of noisy graphs with imbalanced class distributions. For each individual chunk, we propose a boosting algorithm to combine discriminative subgraph pattern selection and model learning as a unified framework for graph classification. To tackle the concept drifting in graph streams, an instance level weighting mechanism is used to dynamically adjust the instance weight, through which the boosting framework can emphasize on difficult graph samples. The classifiers built from different graph chunks form an ensemble for graph stream classification. Experiments on real-life imbalanced graph streams demonstrate clear benefits of our boosting design for handling imbalanced noisy graph stream.**

***Index Terms*—Data streams, graphs, imbalanced class distributions, graph ensemble boosting, noise.**

## I. Introduction

GRAPH classification has drawn increasing interests due to the large number of applications involving complex structured data with dependency relationships. Examples include identifying bugs in computer program flows [1], categorizing scientific publications using co-authorships [2] or citation-ships [3], and predicting chemical compound activities in bioassay tests [4], [5], [6].

The main challenge of graph classification is that graphs only contain node-edge structure information, and no feature is readily available for training classifiers. This challenge motivates many works for graph classification [7], [8], [9], [4], [10], [11], [12], which either try to learn global distance/similarity between two graphs [10], or selecting some local discriminative subgraphs [4], [11] and transfer the graph into vector format (by examining the appearance of subgraph features in each graph, as shown in Fig. 1).

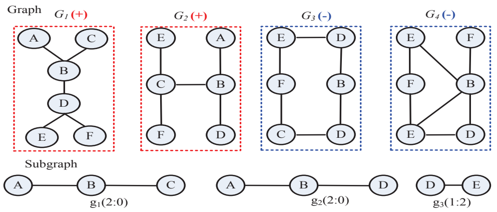


Fig. 1. An example demonstrating graphs and subgraphs: The toy graph database consists of four graphs $G_1, \cdots, G_4$. Given three subgraph features $g_1$, $g_2$, and $g_3$ , each graph can be represented as a vector by examining the presence of the subgraph features. For instance, $G_1$ and $G_2$ can be represented as vectors [1,1,1] and [1,1,0], respectively. The numbers $(a : b)$ followed each subgraph denote the number of times the subgraph appears in positive and negative classes, respectively.

Existing algorithms [8], [11] have demonstrated good classification performance for graph data 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 (http://pubchem.ncbi.nlm.nih.gov). Learning from datasets with imbalanced class distributions has been widely studied in past years. Popular techniques include sampling [13], ensemble learning [14], [15], and SVM adapting [16], [17], and a recent monograph has discussed many methods for imbalanced data classification [18]. 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 [13] to under-sample graphs in the majority class to obtain a relatively balanced graph dataset, and then apply graph classification methods [11], [19]. 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,

S. Pan, J. Wu, and C. Zhang are with the Centre for Quantum Computation & Intelligent Systems, FEIT, University of Technology Sydney, NSW 2007, Australia; E-mail: shirui.pan@student.uts.edu.au; chengqi.zhang@uts.edu.au

J. Wu is also with the Dept. of Computer Science, China University of Geosciences, Wuhan 430074, China; Email: jia.wu@student.uts.edu.au;

X. Zhu is with the Dept. of Computer and Electrical Engineering & Computer Science, Florida Atlantic University. E-mail: xzhu3@fau.edu;

Manuscript received ** **, 2013; revised ** **, 201-.

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.

The second challenge arisen in real-life applications is the dynamic increase and change of structural information over time, *i.e.*, graph streams [2][3][20]. 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 [3], 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 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 classifies 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 paper, 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 paper 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 paper is structured as follows. We review the related work in Section II. Problem definitions and overall framework are discussed in Section III. Section IV reports the proposed algorithm for learning from noisy and imbalanced graph data. Our gEBoost algorithm is detailed in Section V. Experimental results are presented in Section VI, and we conclude the paper in Section VII.

## II. Related work

**Graph Classification:** As graphs involve node-edge structures whereas most existing learning algorithms use 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 in the area mainly fall into two categories (1) global distance based methods (including graph kernel [10], [21], [5] and graph embedding [22]), and (2) local subgraph feature based methods. For global distance based methods, graph kernels and graph embedding are used to calculate distances between a pair of graphs by comparing their common paths. The calculated distance matrix can be fed into a learning algorithm, such as $k$-NN and SVM, for graph classification.

For local subgraph feature based methods, the major goal is to identify local subgraph structures as features [11], [19], [8], and transfer graph data into vector space so existing machine learning methods can be applied for classification [11], [19].

After obtaining the subgraph features, one can also employ Boosting algorithm for graph classification [23], [8], [9]. In [8], Saigo *et al.* proposed a mathematical LP-boost style of algorithm and demonstrated it is effective and converge very fast. Recently we have extended gBoost to igBoost [6] for imbalanced data, but not for graph streams.

For all existing graph classification methods, they are designed for balanced graph datasets, and assume that the underlying graph set is static and there is no treatment to handle dynamic graph streams.

**Data Imbalance and Noise Handling:** Many methods exist for imbalanced data classification, including sampling [13], ensembling [14], [15], and support vector machine adapting [16], [17]. Some recent reviews and monograph on imbalanced data classification are also available [24], [25], [18]. 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 relative balanced graph set, and then apply existing graph classification methods [11], [19]. This solution has shown positive results on general data [26], but in 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) [17]. 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 on samples in minority class to achieve performance gain may falsely emphasize on noise and suffer severe performance loss instead.

**Imbalanced Data Stream Classification:** The task of data stream classification [27], [28], [29], [30] 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 [29], [30] or to actively detect changes [31] and retrain models accordingly. Some recent works have considered both stream classification and data imbalance [32], [33], [34], [35]. In [33], 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 [36] 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 applying Learn++.CDS to the vector data, or integrating existing gBoost algorithm [8] as a base graph classifier into Learn++.NIE. However, all these straightforward solutions may fail to identify discriminative features for imbalanced graphs, and eventually lead to inferior accuracy, as our experiments will show in Section VI.

**Graph Stream Classification:** Data stream classification has been recently extended to structural graph data [5], [2], [3], [37]. In [2], the authors proposed to hash graph edges into random numbers and used discriminative edges as patterns for classification. In [5], we 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, we proposed to project subtree patterns of different chunks onto a set of common low-dimensional feature spaces by using hashing algorithm [37]. Graph classification was also studied recently in a semi-supervised setting [3], with both labeled and unlabeled graphs being used to find discriminative subgraphs with minimum redundancy. In comparison, our algorithm considers both data imbalance and noise, and presents a stream based algorithm for graph classification.

## III. Problem Definitions and Overall Framework

### A. Problem Definitions

***Definition 1:*** **Connected Graph**: A graph is denoted by $G = (\mathcal{V}, E, \mathcal{L})$, where $V = \{v_1, \cdots, v_{n_v}\}$ is the vertices set, $E \subseteq \mathcal{V} \times \mathcal{V}$ is the edge set, and $\mathcal{L}$ is a labelling function assigning 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 this paper, we focus on connected graphs and assume that each graph $G_i$ has a class label $y_i$, $y_i \in \mathcal{Y} = \{-1, +1\}$, which may indicate overall property of the graph, such as the active/negative response of a chemical compound (*i.e.* a graph) [4], [5], or the categorization of a publication [3]. In addition, $y_i = +1$ denotes the minority (positive) class, and $y_i = -1$ is the majority class (negative). We only focus on binary-class classification tasks, but our solutions can be easily extended to multi-class tasks.

***Definition 2:*** **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 3:*** **Subgraph**: Given two graphs $G = (\mathcal{V}, E, \mathcal{L})$ and $g_i = (\mathcal{V}', E', \mathcal{L}')$, $g_i$ is a subgraph of $G$ (*i.e* $g_i \subseteq G$) if there is an injective function $f$: $\mathcal{V}' \to \mathcal{V}$, such that $\forall(a, b) \in E'$, we have $(f(a), f(b)) \in E$, $\mathcal{L}'(a) = \mathcal{L}(f(a))$, $\mathcal{L}'(b) = \mathcal{L}(f(b))$, $\mathcal{L}'(a, b) = \mathcal{L}(f(a), f(b))$. If $g_i$ is a subgraph of $G$ ($g_i \subseteq G$), $G$ is a supergraph of $g_i$ ($G \supseteq g_i$).

***Definition 4:*** **Subgraph Features**: Let $\mathbf{g} = \{g_1, \cdots, g_m\}$ denote a set of subgraph patterns discovered from a given graph set (In this paper, subgraph patterns and subgraph features are equivalent terms). For each graph $G_i$, we can use a subgraph feature vector $x_i = [x_i^{g_1}, \cdots, x_i^{g_m}]$ 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. 1, three subgraph $g_1$, $g_2$, and $g_3$ are used to represent graph $G_2$ as $x_2 = [1, 1, 0]$.

***Definition 5:*** **Noisy Graph**: Given a graph dataset $T = \{(G_1, y_1), \cdots, (G_n, y_n)\}$, 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).

**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 be highly imbalanced in class distributions and have noisy class labels.

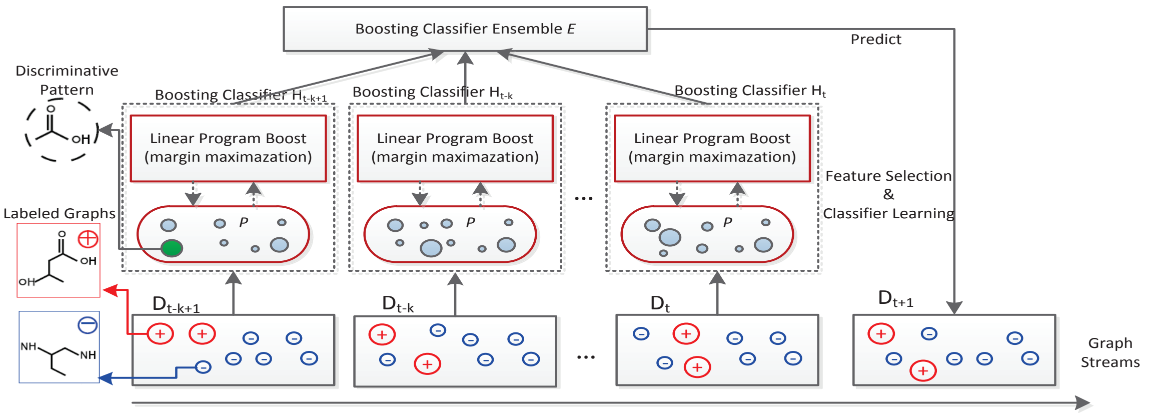

Boosting Classifier Ensemble E
Predict
Discriminative Pattern
OH
Boosting Classifier H_{t-k+1}
Boosting Classifier H_{t-k}
Boosting Classifier H_t
Linear Program Boost (margin maximazation)
Linear Program Boost (margin maximazation)
Linear Program Boost (margin maximazation)
P
P
P
...
Feature Selection & Classifier Learning
Labeled Graphs
OH
OH
NH
NH
D_{t-k+1}
D_{t-k}
D_t
D_{t+1}
...
Graph Streams

Fig. 2. 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 $\beta$ 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. (3) and Section IV). A classifier ensemble $E$ is built from the most recent $k$ chunks to predict graphs in a yet-to-come chunk $D_{t+1}$.

### B. Overall Framework

In this paper, 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. 2, contains 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.
- **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 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 V.

## IV. 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. 1.

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

$$\hbar(G_i; g_j, \pi_j) = \begin{cases} \pi_j & : \; g_j \in G_i \\ -\pi_j & : \; g_j \notin G_i \end{cases} \qquad (1)$$

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 [8], 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:

$$\mathcal{H}_t(G_i) = \sum_{(g_j, \pi_j) \in \mathcal{F} \times \mathcal{Y}} w_j \hbar(G_i; g_j, \pi_j) \qquad (2)$$

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.

### A. Framework of Linear Boosting Algorithm

Our linear boosting algorithm for noisy and imbalanced graphs is shown in Fig. 3. 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.

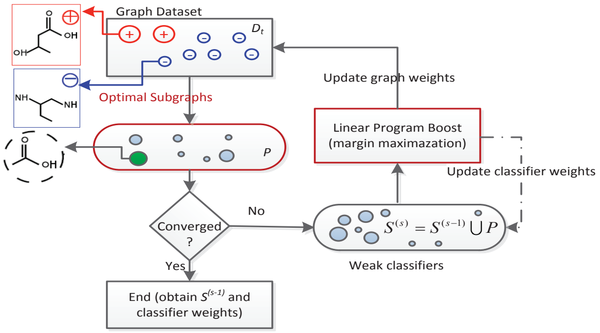


Fig. 3. The proposed boosting framework for learning from noisy and imbalanced graphs in each chunk. The initial weight of each positive graph in $D_t$ is $\beta$ times larger than a negative graph (the circle size corresponds to graph weight), and the weight will be further adjusted to capture "difficult" samples (detailed in Section V). 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.

- **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, *i.e.*, margin maximization.
- **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, 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.

## B. 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:

$$\begin{array}{ll} \max\limits_{\boldsymbol{w},\rho,\boldsymbol{\xi}\in\Re_+^N} & \rho - C(\beta \sum_{\{i|y_i=+1\}}^{n^+} \delta_i\varphi_i\xi_i + \sum_{\{i|y_i=-1\}}^{n^-} \delta_i\varphi_i\xi_i) \\ s.\ t. & y_i \sum_{j=1}^{m} w_j \cdot \hbar(G_i; g_j, \pi_j) + \xi_i \geq \rho, \;\; i = 1,2\cdots n \\ & \sum_{j=1}^{m} w_j = 1, \;\; w_j \geq 0 \;\; j = 1,2\cdots m \end{array} \tag{3}$$

The above objective function aims to maximize the margin $\rho$ 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 $\xi_i$. Here, $n^+$ and $n^-$ denote the number of graphs in positive and negative classes ($n = n^+ + n^-$) in chunk $D_t$. $C = \frac{1}{\nu n}$ is a parameter controlling the magnitude of misclassification in the algorithm. The idea of margin maximization is similar to gBoost [8] and SVM formulation. To handle graph streams with imbalanced distributions and noise, we incorporate three key components: $\delta_i$, $\beta$, and $\varphi_i$ in the objective function. $\delta_i$ indicates the weight factor for handling *disturbing graph samples* in a data stream setting (In this section, $\delta_i$ is set as a fixed value, and Section V will show that $\delta_i$ 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. In our formulation, the weights of positive graphs are $\beta$ 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 $\varphi_i$, for each graph $G_i$, to indicate how likely $G_i$ is a noisy graph. By using $\varphi_i$ to adjust the weight of each graph, we can reduce the impact of noisy graphs on the learning process.

To calculate $\varphi_i$ in Eq.(3), 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 $\varphi_i$ will have a smaller value). Therefore, our approach to calculate $\varphi_i$ is given as follows:

$$\varphi_i = \frac{2}{1 + e^{\tau d(G_i)}}; \tag{4}$$

In Eq.(4), $d(G_i)$ 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.

## C. Linear Boosting with Graph Data

The objective function in Eq.(3) 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, which is NP-complete. Therefore, Eq.(3) cannot be solved directly. *Column generation (CG)* [38], a classic optimization technique, provides an alternative solution to solve this problem. Instead of directly solving the primal problem in Eq.(3), 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.(3) as follows [1]:

$$\begin{array}{lll} \min\limits_{\boldsymbol{\mu},\gamma} \gamma & & \\ s.\ t. & \sum_{i=1}^{n} y_i\mu_i \cdot \hbar(G_i; g_j, \pi_j) \leq \gamma, & j = 1,2\cdots m \\ & 0 \leq \mu_i \leq \beta\delta_i\varphi_i C & if \quad y_i = +1 \\ & 0 \leq \mu_i \leq \delta_i\varphi_i C & if \quad y_i = -1 \\ & \sum_{i=1}^{n} \mu_i = 1. & \end{array} \tag{5}$$

[1] The derivation from Eq.(3) to Eq.(5) is illustrated in Appendix.

According to the duality theory [39], Eq.(3) and Eq.(5) have the same solution (objective values) though they have different predicted variables ($\boldsymbol{w}, \rho, \boldsymbol{\xi}$ in Eq.(3) and $\boldsymbol{\mu}, \gamma$ in Eq.(5)).

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.(5) enforces restriction on a subgraph pattern $(g_j, \pi_j)$ 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.(3), 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.5) 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.(5) 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.3). Such a top-$l$ constraint technique is known as *Multiple Prices* [40] in column generation.

To apply multiple prices, we first define the discriminative score for each subgraph based on the constraints in Eq.(5).

***Definition 6:*** **Discriminative Score:** for a subgraph decision stump $(g_j, \pi_j)$, its discriminative score is defined as:

$$\mathbb{i}(g_j, \pi_j) = \sum_{i=1}^{n} y_i\mu_i \cdot \hbar(G_i; g_j, \pi_j) \qquad (6)$$

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:

$$\begin{array}{lll}
 & \min\limits_{\gamma^{(s)}, \boldsymbol{\mu}^{(s)}} \gamma^{(s)} & \\
s.\ t. & \sum_{i=1}^{n} y_i\mu_i^{(t)}\hbar(G_i; g_j, \pi_j) \leq \gamma^{(t)}, \forall (g_j, \pi_j) \in \mathcal{S}^{(s)} & \\
 & 0 \leq \mu_i^{(s)} \leq \beta\delta_i\varphi_i C & if \quad y_i = +1 \\
 & 0 \leq \mu_i^{(s)} \leq \delta_i\varphi_i C & if \quad y_i = -1 \\
 & \sum_{i=1}^{n} \mu_i^{(s)} = 1. &
\end{array} \qquad (7)$$

The solutions to Eq.(7) 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.(7), $\varphi_i$ changes in each iteration, because the class centers for positive and negative graphs are calculated by using current selected subgraphs $\mathcal{S}^{(s)}$ (transfer each graph as a vector based on $\mathcal{S}^{(s)}$). The changing subgraph features will result in updated class centers, and result in new $\varphi_i$ value according to Eq.(4).

**Algorithm 1** 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;
 1: $\mu_i = \begin{cases} \varsigma^+ & : \ y_i = +1 \\ \varsigma^- & : \ y_i = -1 \end{cases}$, where $\frac{\varsigma^+}{\varsigma^-} = \beta$, $\sum_{i=1}^{n} \mu_i = 1$;
 2: $\mu_i \leftarrow \mu_i\delta_i$, where $\sum_{i=1}^{n} \mu_i = 1$;
 3: $\mathcal{S}^{(0)} \leftarrow \emptyset$; $\gamma^{(0)} \leftarrow 0$;
 4: $s \leftarrow 0$;
 5: while true do
 6:     Obtain top-$l$ subgraph decision stumps $P = \{(g_i, \pi_i)\}_{i=1,\cdots,l}$;   //Algorithm 2;
 7:     $\mathbb{i}(g^\star, \pi^\star) = \max_{(g_j,\pi_j)\in P} \mathbb{i}(g_j, \pi_j)$
 8:     if $\mathbb{i}(g^\star, \pi^\star) \leq \gamma^{(s-1)} + \varepsilon$ then
 9:         break;
10:     $\mathcal{S}^{(s)} \leftarrow \mathcal{S}^{(s-1)} \bigcup P$;
11:     Obtain the membership value $\varphi_i$ for each graph example $G_i$ based on $S^{(s)}$ and Eq. (4);
12:     Solve Eq. (7) to get $\gamma^{(s)}$, $\boldsymbol{\mu}^{(s)}$, and Lagrange multipliers $\boldsymbol{w}^{(s)}$;
13:     $s \leftarrow s + 1$;
14: return $\mathcal{H}_t(G_i) = \sum_{(g_j,\pi_j)\in S^{(s-1)}} w_j^{(s-1)}\hbar(G_i; g_j, \pi_j)$;
```

**Algorithm 2** 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:     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 $P = \{(g_i, \pi_i)\}_{i=1,\cdots,l}$;
```

Our graph boosting framework is illustrated in Algorithm 1. To handle class imbalance, the weight of each positive graph $\mu_i$ is set to be $\beta$ times larger than the weights of negative graphs (step 1). The weight value is further updated by $\delta_i$ (step 2), which takes the concept drifting in streams into consideration (detailed in Section V). After that, the boosting algorithm iteratively selects top-$l$ subgraphs $P = \{(g_i, \pi_i)\}_{i=1,\cdots,l}$ in each round (step 6). On 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 ($\varepsilon$). On steps 10-12, the linear programming problem in Eq.(7) is solved based on the selected subgraphs using the open source software CVX (http://cvxr.com/cvx/). After Eq.(7) 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.(7). 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 1), we need to enumerate the entire set of subgraph patterns, with respect to a given threshold, from the training graphs $T$. In our boosting algorithm, we employ a Depth-First-Search (DFS) based algorithm gSpan [41] 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* [8] to prune the search space.

***Theorem** 1:* Given a subgraph feature $g_j$, let

$$\begin{aligned} \mathbb{i}_+^{(g_j)} &= 2\textstyle\sum_{\{i|y_i=+1,g_j\in G_i\}}\mu_i^{(t)} - \sum_{i=1}^n y_i\mu_i \\ \mathbb{i}_-^{(g_j)} &= 2\textstyle\sum_{\{i|y_i=-1,g_j\in G_i\}}\mu_i^{(t)} + \sum_{i=1}^n y_i\mu_i \\ \mathbb{i}(g_j) &= \max\left(\mathbb{i}_+^{(g_j)}, \mathbb{i}_-^{(g_j)}\right) \end{aligned} \tag{8}$$

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

Because a subgraph decision stump may have a positive or a negative label $\mathcal{Y} = \{+1, -1\}$, we calculate its maximum score based on each possible value, and select the maximum one as the upperbound.

According to Theorem 1, 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 2. The minimum value $\eta$ 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 $\eta$ or the current set $P$ has less than $l$ subgraph patterns, we add $(g_p, \pi_p)$ 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 [8], 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.

## V. gEBoost Algorithm

In this section, we discuss the proposed ensemble framework, which combines classifiers trained from local chunks (as described in Section IV) 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 can not be correctly classified by current models. Accordingly, we define disturbing graph samples by using models trained from historical data as follows:

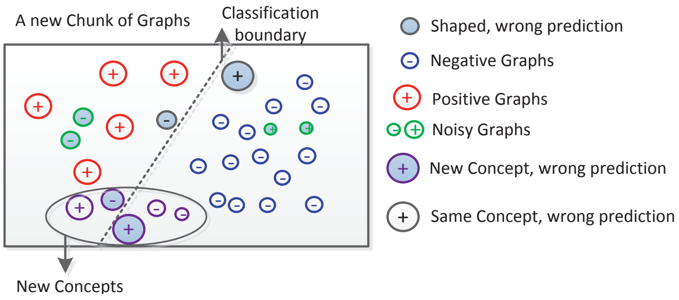


Fig. 4. A conceptual view of graph weighting scheme. Given a new chunk of graphs with positive and negative graphs (circle sizes indicate the weights), the current classifier may make incorrect prediction on three kinds of disturbing graph samples: (1) For a noisy graph $G_i$ (green circles), its weight will be first increased inversely proportional to the accuracy of the current ensemble classifier (measured by $\delta_i$), and be further decreased according to $G_i$'s distance to the class centres (correspond to $\varphi_i$). (2) For emerging new concept graphs (purple circles), if the current ensemble makes an incorrect prediction, their weights will be increased by $\delta_i$ 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 $\delta_i$) 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.

***Definition** 7:* **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.

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.(3) and step 2 of algorithm 1).

**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 sufficient knowledge to correctly classify this graph. This weighting mechanism is similar to Adaboost [42] and our semi-supervised graph stream classification method [3]. 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. 4.

**Algorithm 3** gEBoost

```
Require:
    $\mathcal{S} = \{D_1, D_2, \cdots\}$: Graph Stream
    $k$: The maximum capacity of the ensemble
 1: Initialize $E = \emptyset, t = 0$;
 2: while $\mathcal{S} != \emptyset$ do
        // Training Phase:
 3:     $D_t \leftarrow$ A new graph chunk;
 4:     $\mathcal{S} \leftarrow \mathcal{S}/D_t$; $t = t + 1$;
 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 1;
 8:         $E \leftarrow E \bigcup \mathcal{H}_t$
 9:     else
10:         $err \leftarrow \frac{\sum_{G_i \in D_t^l}(E(G_i) != y_i)}{|D_t^l|}$;
11:         $\delta_i = \begin{cases} \delta_i \sqrt{(1-err)/err} & : \ E(G_i) != y_i, G_i \in D_t \\ \delta_i \sqrt{err/(1-err)} & : \ E(G_i) = y_i, G_i \in D_t \end{cases}$;
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 1;
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:     $\mathcal{H}(G_p|E) = \arg\max \sum_{i=t-k-1}^{t} \alpha_i \mathcal{H}_i(G_p)$
```

**gEBoost Algorithm:** Algorithm 3 lists detailed procedures of gEBoost framework which combines instance weighting and graph boosting for graph stream classification.

The "while" loop in Algorithm 3 represents a stream processing cycle which repeats as graph data continuously arrive. For the first graph chunk $D_1$, gEBoost simply builds a linear boost classifier using Algorithm 1 without considering concept drifting ($\delta_i = 1$), and adds classifier $\mathcal{H}_t$ to initialize the ensemble $E$ (lines 5-9).

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 20-21), gEBoost first calculates the weighted accuracy $\alpha_i$ 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 $D_{t+1}$.

## VI. 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 [43].

TABLE I
DBLP Graph Stream Used in Experiments

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

### A. Experimental Settings

Three graph streams collected from real-world applications are used in our experiments.

**DBLP Graph Stream:** The DBLP dataset [2] consists of computer science bibliography. Each record in DBLP corresponds to one publication with information including paper ID, title, abstract, authors, year of publication, venue, references [44]. To build a graph stream, we select a list of conferences (as shown in Table I) and use papers published in these conferences (in chronological order) to form a graph stream. 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. DBDM+AIML and CV overlap in many aspects, such as machine learning and visual information retrieval, and research focuses and topics may vary in different years. All these changes make DBLP stream an ideal test ground for concept drifting graph stream classification. For example, there are an increasing number of papers to address social network research problems for both DBDM and CV fields (*i.e.*, community discovery for DBDM and social tagging in CV) which introduces gradual concept drifting in the stream.

In our experiments, each paper is represented as a graph with each node denoting a Paper ID or a keyword and each edge representing the citation relationship between papers or keyword relations in the title. In addition, (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. 5.

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 I lists the number of graphs in each category.

**NCI Chemical Compound Stream**. The NCI cancer screening datasets are commonly used as graph classification benchmark. We download two NCI datasets from PubChem [3]. Each NCI dataset belongs to a bioassay task for anticancer activity prediction, where each chemical compound is represented as

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

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

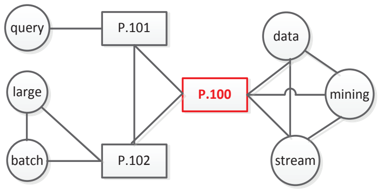


Fig. 5. Graph representation for a paper (P.100) in DBLP. The rectangles are paper ID nodes and circles are keyword nodes. Paper P.100 cites (connects) paper P.101 and P.102, and P.100 has keywords *Data*, *Stream*, and *Mining* in its title. Paper P.101 has keyword *Query* in its title, and P.102's title include keywords *Large* and *Batch*. For each paper, the keywords in the title are linked to each other.

TABLE II
NCI CANCER SCREEN DATASETS USED IN THE EXPERIMENTS

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

a graph, with atoms representing nodes and chemical bonding denoting edges. A chemical compound is positive if it is active against the corresponding cancer, or negative otherwise.

Table II summarizes the NCI graph data used in our experiments, where columns 2-3 show the number of positive graphs and the total number of graphs in the original datasets. After removing disconnected graphs and graphs with unexpected atoms (some atoms are represented as '*'), we obtain new datasets with slightly different sizes, as shown in columns 4-5. Meanwhile, each NCI dataset is highly imbalanced, with less than $5\%$ graphs in the positive class (shown in column 6 of Table II). In our experiments, we concatenate two datasets as one stream with 74,371 graphs in total (4.38% samples belonging to positive class). In the NCI graph stream, the bioassay task changes from NCI-1 to NCI-33, which simulates the concept drifting in the stream (*i.e. sudden drift*). We sequentially construct graph chunks such that each chunk consists of 4.38% positive graphs and others are negative.

**Stanford Twitter Stream**. The twitter stream are extracted from twitter sentiment classification [4]. 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 stream, 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.

In our experiments, 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.

**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][37], 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][37] 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][37] are limited to their own learning algorithms (*e.g.*, [2] can only use k-NN classifier).

For gBoost, 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. The most

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

recent $k$ chunks form an ensemble to predict graphs in a future chunk.

- **gBoost+Stream** applies the gBoost algorithm 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 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 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 takes 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 [33] for both Learn++.CDS-DT and Learn++.NIE-gBoost algorithms.

## B. Experimental Results

*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 IV) 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.(3) 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.

The results in Fig. 6 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 g-Boost, 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, gEBoost 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 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 ($\varphi_i$ for graph $G_i$ of Eq. (3)) 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 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.

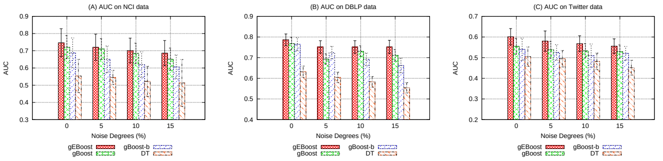


Fig. 6. Comparison of different algorithms for imbalanced graph stream classification. For each chunk $D_t$, we build a classifier $\mathcal{H}_t$ from chunk $D_t$ to predict graphs in $D_{t+1}$. The results represent the average AUC values and standard deviation over all chunks in each graph stream. (A) NCI stream; (B) DBLP stream; (C) Twitter stream.

*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. 7, 8, and 9, we vary the noise levels in each graph chunk, and report the results on NCI, DBLP, and Twitter streams.

The results in Figs. 7, 8, and 9 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

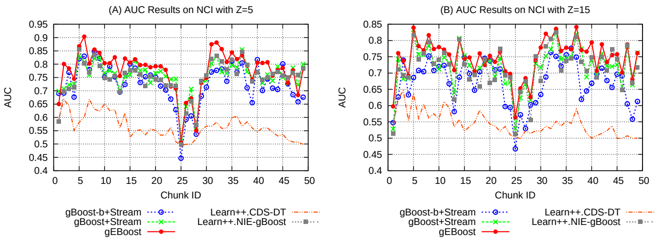


Fig. 7. AUC *w.r.t.* different noise levels on **NCI stream** with ensemble size $k$=10 and chunk size $D_t$= 1500. (A)$Z = 5$; (B) $Z = 15$.

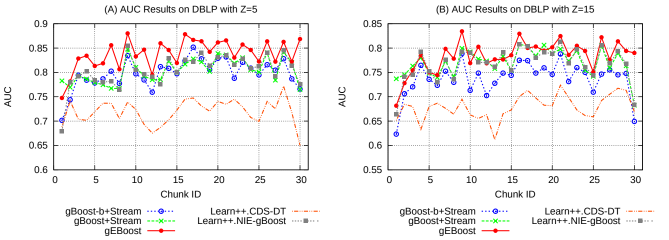


Fig. 8. AUC *w.r.t.* different noise levels on **DBLP stream** with ensemble size $k$=10 and chunk size $D_t$= 800. (A)$Z = 5$; (B) $Z = 15$.

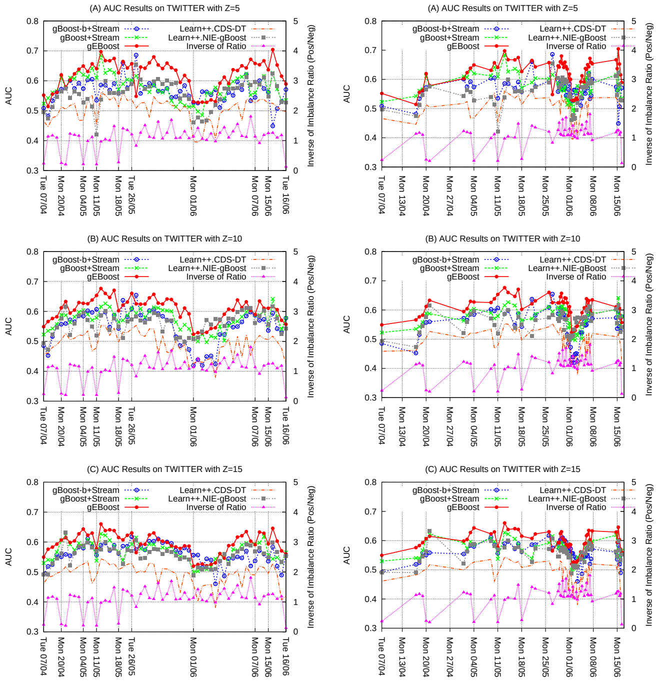


Fig. 9. AUC *w.r.t.* different noise levels on **Twitter stream**. Figures on the left panel are plotted with respect to uniform intervals of chunks in the $x$-axis, and figures on the right panel are plotted with respect to uniform intervals of weeks in the $x$-axis.

genuine discriminative 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 undersamples graphs in each chunk to alleviate the data imbalance, is significantly inferior to gBoost+Stream, Learn++.NIE-gBoost, and gEBoost, especially in Fig. 8 (B). This is because each graph chunk is extremely imbalanced (*e.g.*, only 66 positive graphs out of 1500 graphs for NCI stream), undersampling 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. 7, 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 receives 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. 7, 8, and 9 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. 9. 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. 9 (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. 9 (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.

The average accuracies over the whole graph stream, in Fig. 10, 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. 11and 12, we report

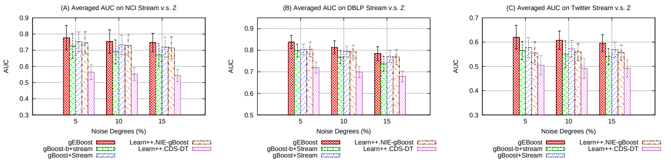


Fig. 10. Averaged AUC values (and standard deviation) *v.s.* different noise degrees $Z$, with ensemble size $k$=10.

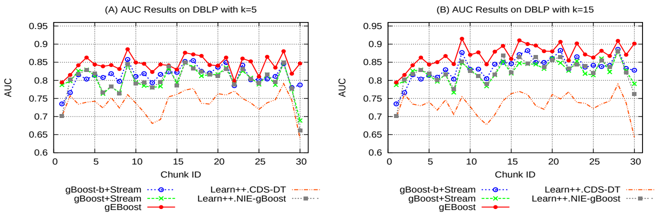


Fig. 11. AUC *w.r.t.* different ensemble sizes on **DBLP stream** with chunk size $|D_t|$=800.

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.

The results show that increasing ensemble size results in improved algorithm performance. For instance, when $k$=5 (in Fig. 11 (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 remaining experiments, we set $k$=10.

**Results on chunk size $D_t$:** In Figs. 13 and 14, 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. 13) and DBLP (Fig. 14) 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. 13 (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 *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 III.

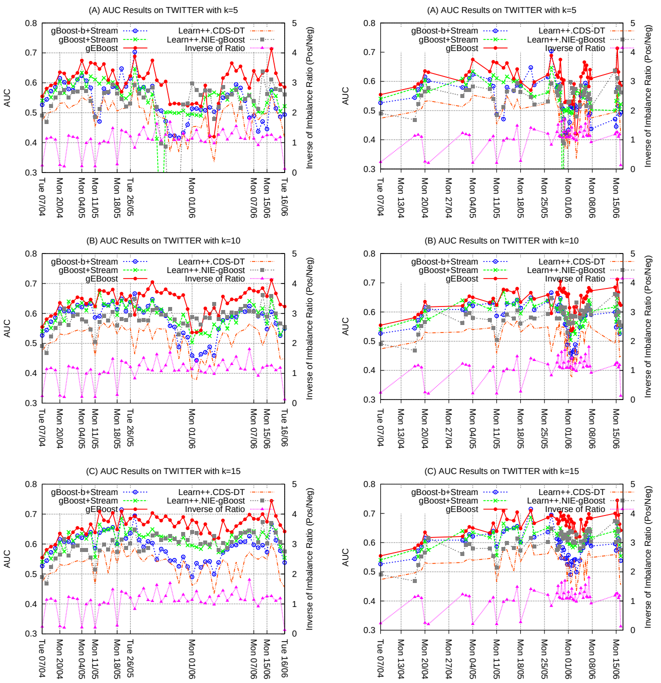


Fig. 12. AUC *w.r.t.* different ensemble sizes on **Twitter stream**. Figures on the left panel are plotted with respect to uniform intervals of chunks in the $x$-axis, and figures on the right panel are plotted with respect to uniform intervals of weeks in the $x$-axis.

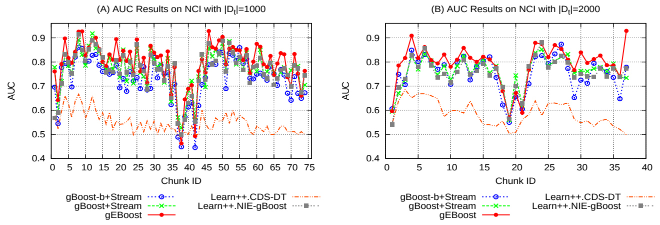


Fig. 13. AUC *w.r.t.* different chunk size on **NCI stream** with ensemble size $k$=10. (A) $|D_t| = 1000$; (B) $|D_t| = 2000$.

Table III 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 III
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±0.032 | 0.823±0.030 | **0.867**±0.028 | 0.736±0.028 | 0.819±0.039 |
| 10 | 0.798±0.031 | 0.819±0.032 | **0.836**±0.028 | 0.723±0.029 | 0.812±0.037 |
| 5 | 0.775±0.035 | 0.798±0.030 | **0.818**±0.031 | 0.710±0.031 | 0.801±0.039 |

*3) Time and Memory Comparisons:* **Time Efficiency.** The runtime efficiency in Figs. 15 and 16 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

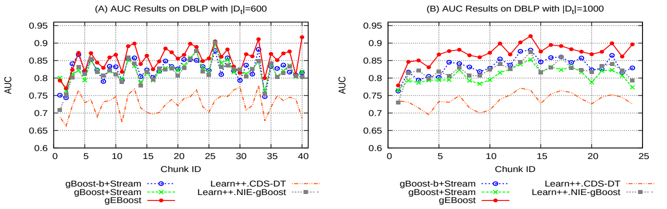


Fig. 14. AUC *w.r.t.* different chunk size on **DBLP stream** with ensemble size $k$=10. (A) $|D_t| = 600$; (B) $|D_t| = 1000$.

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. Meanwhile, the accumulated system runtime *w.r.t.* different chunk sizes, as shown in Fig. 16, 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.

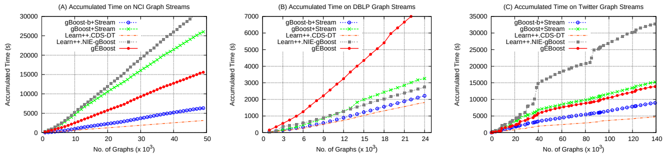


Fig. 15. System accumulated runtime *v.s.* number of graphs processed over stream. (A) NCI stream; (B) DBLP stream; (C) Twitter stream.

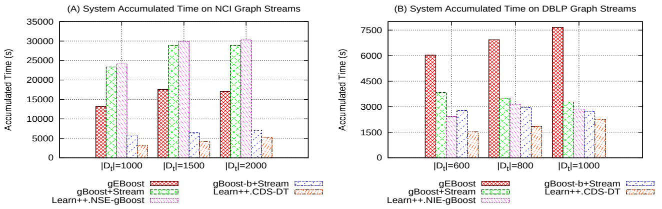


Fig. 16. System accumulated runtime *v.s.* different chunk sizes $|D_t|$.

**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 obsoleted 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.

## VII. Conclusion

In this paper, 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 paper, 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 (1) 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; and (2) 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 validates the effectiveness of our algorithm.

## References

[1] H. Cheng, D. Lo, Y. Zhou, X. Wang, and X. Yan, "Identifying bug signatures using discriminative graph mining," in *ISSTA*, 2009, pp. 141–152.

[2] C. Aggarwal, "On classification of graph streams," in *Proc. of SDM*, Arizona, USA, 2011.

[3] S. Pan, X. Zhu, C. Zhang, , and P. S. Yu, "Graph stream classification using labeled and unlabeled graphs," in *International Conference on Data Engineering (ICDE)*. IEEE, 2013.

[4] M. Deshpande, M. Kuramochi, N. Wale, and G. Karypis, "Frequent substructure-based approaches for classifying chemical compounds," *IEEE Trans. on Knowl. and Data Eng.*, vol. 17, pp. 1036–1050, 2005.

[5] B. Li, X. Zhu, L. Chi, and C. Zhang, "Nested subtree hash kernels for large-scale graph classification over streams," in *Proc. of IEEE ICDM*. IEEE, 2012, pp. 399–408.

[6] S. Pan and X. Zhu, "Graph classification with imbalanced class distributions and noise," in *Proc. of IJCAI*. AAAI Press, 2013, pp. 1586–1592.

[7] Y. Zhu, J. Yu, H. Cheng, and L. Qin, "Graph classification: a diversified discriminative feature selection approach," in *Proc. of CIKM*. ACM, 2012, pp. 205–214.

[8] H. Saigo, S. Nowozin, T. Kadowaki, T. Kudo, and K. Tsuda, "gboost: a mathematical programming approach to graph classification and regression," *Machine Learning*, vol. 75, pp. 69–89, 2009.

[9] 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.

[10] H. Kashima, K. Tsuda, and A. Inokuchi, *Kernels for Graphs*. Cambridge (Massachusetts): MIT Press, 2004, ch. In: Schlkopf B, Tsuda K, Vert JP, editors. Kernel methods in computational biology.

[11] X. Kong and P. Yu, "Semi-supervised feature selection for graph classification," in *Proc. of ACM SIGKDD*, Washington, DC, USA, 2010.

[12] J. Wu, Z. Hong, S. Pan, X. Zhu, C. Zhang, and Z. Cai, "Multi-graph learning with positive and unlabeled bags," in *Proc. of SIAM Data Mining*. SIAM, 2014, pp. 1586–1592.

[13] X. Liu, J. Wu, and Z. Zhou, "Exploratory undersampling for class-imbalance learning," *IEEE Trns. on SMC-B*, vol. 39, pp. 539–550, 2009.

[14] M. Galar, A. Fernández, E. Barrenechea, H. Bustince, and F. Herrera, "A review on ensembles for the class imbalance problem: bagging-, boosting-, and hybrid-based approaches," *IEEE Trans. SMC-Part C*, vol. 42, no. 4, pp. 463–484, 2012.

[15] J. Leskovec and J. Shawe-Taylor, “Linear programming boosting for uneven datasets,” in *Proc. of ICML*. AAAI Press, 2003, pp. 456–463.

[16] K. Veropoulos, C. Campbell, N. Cristianini *et al.*, “Controlling the sensitivity of support vector machines,” in *Proc. of the IJCAI*, vol. 1999. Citeseer, 1999, pp. 55–60.

[17] R. Akbani, S. Kwek, and N. Japkowicz, “Applying support vector machines to imbalanced datasets,” *Machine Learning: ECML 2004*, pp. 39–50, 2004.

[18] H. He and Y. Ma, *Imbalanced Learning: Foundations, Algorithms, and Applications*. Wiley-IEEE Press, 2013.

[19] S. Ranu and A. Singh, “Graphsig: A scalable approach to mining significant subgraphs in large graph databases,” in *Proc. of ICDE*. IEEE, 2009, pp. 844–855.

[20] S. Pan and X. Zhu, “Cgstream: continuous correlated graph query for data streams,” in *Proc. of CIKM*. ACM, 2012, pp. 1183–1192.

[21] P. Mahe, N. Ueda, T. Akutsu, J. Pettet, and J. Vert, “Externsions of marginalized graph kernels,” in *Proc. of ICML*, Alberta, Canada, 2004.

[22] K. Riesen and H. Bunke, “Graph classification by means of lipschitz embedding,” *IEEE Trans. on SMC - B*, vol. 39, pp. 1472–1483, 2009.

[23] T. Kudo, E. Maeda, and Y. Matsumoto, “An application of boosting to graph classification,” in *Proc. of NIPS*, Vancouver, Canada, 2004.

[24] H. He and E. Garcia, “Learning from imbalanced data,” *IEEE TKDE*, vol. 21, no. 9, pp. 1263–1284, 2009.

[25] Y. Sun, A. Wong, and S. MOHAMED, “Classification of imbalanced data: A review,” *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 23, no. 04, pp. 687–719, 2009.

[26] J. Van Hulse and T. Khoshgoftaar, “Knowledge discovery from imbalanced and noisy data,” *Data & Knowledge Engineering*, vol. 68, no. 12, pp. 1513–1542, 2009.

[27] P. Domingos and G. Hulten, “Mining high-speed data streams,” in *Proc. of the 6th ACM KDD*. ACM, 2000, pp. 71–80.

[28] W. N. Street and Y. Kim, “A streaming ensemble algorithm (sea) for large-scale classification,” in *Proc. of 7th KDD*, 2001, pp. 377–382.

[29] H. Wang, W. Fan, P. S. Yu, and J. Han, “Mining concept-drifting data streams using ensemble classifiers,” in *Proc. of the 9th ACM KDD*. ACM, 2003, pp. 226–235.

[30] X. Zhu, P. Zhang, X. Lin, and Y. Shi, “Active learning from stream data using optimal weight classifier ensemble,” *IEEE Trans. on SMC - B*, vol. 40, pp. 1607–1621, 2010.

[31] M. Baena-García, J. del Campo-Ávila, R. Fidalgo, A. Bifet, R. Gavaldà, and R. Morales-Bueno, “Early drift detection method,” in *Proc. of ECML/PKDD*, 2006.

[32] J. Gao, W. Fan, J. Han, and S. Y. Philip, “A general framework for mining concept-drifting data streams with skewed distributions.” in *Proc. SIAM Int’l Conf. Data Mining*, 2007.

[33] G. Ditzler and R. Polikar, “Incremental learning of concept drift from streaming imbalanced data,” *IEEE Transactions on Knowledge and Data Engineering*, 2013.

[34] S. Wang, L. L. Minku, and X. Yao, “A learning framework for online class imbalance learning,” in *IEEE Symposium Series on Computational Intelligence (SSCI)*, 2013.

[35] S. Chen and H. He, “Towards incremental learning of nonstationary imbalanced data stream: a multiple selectively recursive approach,” *Evolving Systems*, vol. 2, no. 1, pp. 35–50, 2011.

[36] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, “S-mote: synthetic minority over-sampling technique,” *Journal of Artificial Intelligence Research*, vol. 16, no. 1, pp. 321–357, 2002.

[37] L. Chi, B. Li, , and X. 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.

[38] S. Nash and A. Sofer, “Linear and nonlinear programming,” *Newyark McGraw-Hill*, 1996.

[39] D. Bertsimas and J. N. Tsitsiklis, *Introduction to linear optimization*. Athena Scientific Belmont, 1973, ch. 4.

[40] D. Luenberger, *Optimization by vector space methods*. Wiley-Interscience, 1997.

[41] X. Yan and J. Han, “gspan: Graph-based substructure pattern mining,” in *Proc. of ICDM*, Maebashi City, Japan, 2002.

[42] Y. Freund and R. Schapire, “A desicion-theoretic generalization of on-line learning and an application to boosting,” in *Computational Learning Theory*. Springer, 1995, pp. 23–37.

[43] S. Pan, X. Zhu, and C. Zhang, “Imblanced noisy graph stream classification: Results and source code.” [Online]. Available: http://www.cse.fau.edu/∼xqzhu/Stream/geboost/index.html

[44] 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*, 2008, pp. 990–998.

## Appendix

**Lagrangian Dual of Eq.(3).** The Lagrangian function of Eq.(3) can be written as:

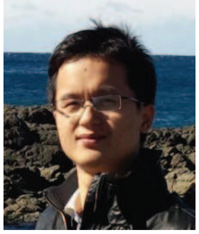
$$\begin{aligned} L \quad & (\boldsymbol{\xi}, \boldsymbol{w}, \rho) = \rho - C(\beta \textstyle\sum_{\{i|y_i=+1\}}^{n^+} \delta_i \varphi_i \xi_i + \sum_{\{i|y_i=-1\}}^{n^-} \delta_i \varphi_i \xi_i) \\ & + \sum_{i=1}^{n} \mu_i \{ y_i \textstyle\sum_{j=1}^{m} w_j \cdot \hbar(G_i; g_j, \pi_j) + \xi_i - \rho \} \\ & -\gamma(\textstyle\sum_{j=1}^{m} w_j - 1) + \sum_{j=1}^{m} q_j \cdot w_j + \sum_{i=1}^{n} p_i \cdot \xi_i \end{aligned} \tag{9}$$

Where, we have $\mu_i \geq 0, p_i \geq 0, q_i \geq 0$, and $\gamma$ 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 $\rho$) must vanish,

$$\begin{aligned} \frac{\partial L}{\partial \xi_{i|y_i=1}} &= -C\beta\delta_i\varphi_i + \mu_i + p_i = 0 \Rightarrow 0 \leq \mu_i \leq C\beta\delta_i\varphi_i \\ \frac{\partial L}{\partial \xi_{i|y_i=-1}} &= -C\delta_i\varphi_i + \mu_i + p_i = 0 \Rightarrow 0 \leq \mu_i \leq C\delta_i\varphi_i \\ \frac{\partial L}{\partial \rho} &= 1 - \textstyle\sum_{i=1}^{n} \mu_i = 0 \Rightarrow \sum_{i=1}^{n} \mu_i = 1 \\ \frac{\partial L}{\partial w_j} &\Rightarrow \textstyle\sum_{i=1}^{n} y_i \mu_i \cdot \hbar(G_i; g_j, \pi_j) - \gamma + q_j = 0 \\ &\Rightarrow \textstyle\sum_{i=1}^{n} y_i \mu_i \cdot \hbar(G_i; g_j, \pi_j) \leq \gamma \end{aligned}$$

Substituting these variables in Eq. (9), we obtain the its dual problem as Eq. (5).



**Shirui Pan** received his master degree in computer science from Northwest A&F University, Yangling, Shaanxi, China, in 2011. Since September 2011, he has been working toward the PhD degree in the Centre for Quantum Computation and Intelligent Systems (QCIS), Faculty of Engineering and Information Technology, University of Technology, Sydney (UTS). His research focuses on data mining and machine learning.

**Jia Wu** received his bachelor degree in computer science from China University of Geosciences (CUG), Wuhan, China, in 2009. Since September 2009, he has been working toward the PhD degree under the Master-Doctor combined program in computer science from CUG. Besides, he is also pursuing a PhD degree in QCIS Centre, Faculty of Engineering and Information Technology, University of Technology, Sydney (UTS), Australia. His research focuses on data mining and machine learning.

**Xingquan Zhu** received the PhD degree in computer science from Fudan University, Shanghai, China. He is an associate professor in the Department of Computer & Electrical Engineering and Computer Science, Florida Atlantic University. Prior to that, he was with the Centre for Quantum Computation & Intelligent Systems, University of Technology, Sydney, Australia. His research interests mainly include data mining, machine learning, and multimedia systems. Since 2000, he has published more than 170 refereed journal and conference papers in these areas, including two Best Paper Awards and one Best Student Paper Award.

**Chengqi Zhang** received the Ph.D. degree from the University of Queensland, Brisbane, Australia, in 1991 and the D.Sc. degree (higher doctorate) from Deakin University, Geelong, Australia, in 2002. Since December 2001, he has been a Professor of Information Technology with the University of Technology, Sydney (UTS), Sydney, Australia, where he has been the Director of the UTS Priority Investment Research Centre for Quantum Computation and Intelligent Systems since April 2008. His research interests mainly focus on data mining and its applications. He is General Co-Chair of KDD 2015 in Sydney and the Local Arrangements Chair of IJCAI-2017 in Melbourne, and a Fellow of the Australian Computer Society and a Senior Member of the IEEE.