Hashing for Adaptive Real-Time Graph Stream Classification With Concept Drifts

Lianhua Chi, Bin Li, Xingquan Zhu, *Senior Member, IEEE*, Shirui Pan, and Ling Chen

***Abstract*—Many applications involve processing networked streaming data in a timely manner. Graph stream classification aims to learn a classification model from a stream of graphs with only one-pass of data, requiring real-time processing in training and prediction. This is a nontrivial task, as many existing methods require multipass of the graph stream to extract subgraph structures as features for graph classification which does not simultaneously satisfy "one-pass" and "real-time" requirements. In this paper, we propose an adaptive real-time graph stream classification method to address this challenge. We partition the unbounded graph stream data into consecutive graph chunks, each consisting of a fixed number of graphs and delivering a corresponding chunk-level classifier. We employ a random hashing function to compress the original node set of graphs in each chunk for fast feature detection when training chunk-level classifiers. Furthermore, a differential hashing strategy is applied to map unlimited increasing features (i.e., cliques) into a fixed-size feature space which is then used as a feature vector for stochastic learning. Finally, the chunk-level classifiers are weighted in an ensemble learning model for graph classification. The proposed method substantially speeds up the graph feature extraction and avoids unbounded graph feature growth. Moreover, it effectively offsets concept drifts in graph stream classification. Experiments on real-world and synthetic graph streams demonstrate that our method significantly outperforms existing methods in both classification accuracy and learning efficiency.**

***Index Terms*—Cliques, concept drifts, graph stream classification, hashing.**

I. Introduction

THE SURGE of real-world structure data, such as chemical compounds, biological data, XML documents, program flows, images, and social networks, has led to the rise of graph mining research. Graph classification aims to learn a discriminative model from training data to predict class labels of test data, where both training and test examples are graphs containing structure information. Because graphs do not have features, the underlying challenge is to represent graphs in feature vectors to facilitate classifier training, by using a generic machine learning framework. A variety of studies [1]–[4] have been proposed to extract features to represent graphs. However, most of them carry out graph classification in a static setting, requiring that all graph data are provided for training, which prevents them from being applied to dynamic streams.

Dynamic networked data are common in many applications, where data are presented with increasing volumes and change over time. For example, a social network is made up of a population of individuals, and their interactions constantly evolve resulting in new connections and changed networks over time. A transportation network is a complex network made up of interconnected routes, with dynamically generated traffic flows over time. These evolving networked data can be defined as graph streams, which not only inherit features of static graphs but also possess special characteristics, such as frequent update and necessary real-time response [5]. To solve these emerging problems, graph stream mining has recently attracted an increasing number of research interests [6]–[10].

In this paper, we focus on the graph stream classification problem. Graph stream is defined on a dynamic network which comprises a massive universe of nodes, where the stream of graphs are represented as sets of interconnected edges on the underlying dynamic network [8], [10]. For example, coauthors of research publications continuously form graphs on a coauthor network (e.g., DBLP), communities of special interest continuously form graphs on a social network (e.g., Facebook), and traffic flows continuously form graphs on a transportation network. In these scenarios, users may want to categorize papers according to their research topics by using discriminative co-author or co-citation patterns. Social network analysts may seek to organize interpersonal relationships by applying an efficient classifier [11]. Traffic managers may categorize transportation routes into different classes to better utilize resources and services. All these realistic requirements incur an urgent need of developing efficient learning methods for graph stream classification.

However, graph stream classification on a dynamic network with massive nodes is by no means an easy problem because of the following challenges.

1) *Increasing Graph Volumes*: The volume of graph data is continuously increasing. Due to the "real-time"

Manuscript received October 20, 2016; revised March 6, 2017, April 5, 2017, and April 20, 2017; accepted April 28, 2017. This work was partially supported by the Program for Professor of Special Appointment (Eastern Scholar) at Shanghai Institutions of Higher Learning. This paper was recommended by Associate Editor J. Wang. *(Corresponding author: Xingquan Zhu.)*

L. Chi is with Melbourne Research Laboratory, IBM Research Australia, Southbank, VIC 3006, Australia (e-mail: lianhuac@au1.ibm.com).

B. Li is with Data61, CSIRO, Eveleigh, NSW 2015, Australia (e-mail: bin.li@data61.csiro.au).

X. Zhu is with the Department of Computer and Electrical Engineering and Computer Science, Florida Atlantic University, Boca Raton, FL 33431 USA, and also with the School of Computer Science, Fudan University, Shanghai 201203, China (e-mail: xzhu3@fau.edu).

S. Pan and L. Chen are with the University of Technology Sydney, Broadway, NSW 2007, Australia (e-mail: shirui.pan@uts.edu.au; ling.chen@uts.edu.au).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCYB.2017.2708979

requirement, one cannot rely on existing subgraph detection methods designed for off-line accurate subgraph mining. Thus, we need to develop an approximate subgraph extraction method to tackle fast increasing graph volumes.

2) *Expanding Feature Space*: The continuously received graphs in the stream would lead to an increasing number of subgraph patterns. Due to the "one-pass" requirement, it's computationally prohibitive to prescan the graph stream and enumerate all subgraph patterns as features. Thus, we need to design a projection that can map arbitrary subgraph patterns onto a fixed-size feature space to represent graphs in the stream.
3) *Concept Drifting*: Another challenge is that the graph data distributions and the decisions for graph classification may evolve and change over time. Accordingly, graph stream classification needs to ensure that the classification model can timely discover and adapt to such concept drifts in the stream for accurate classification.

Aggarwal [8] and Chi *et al.* [10] have investigated the graph stream classification problem. Both of them employ hashing techniques to sketch the graph stream, in order to save computational cost and control the size of the subgraph-pattern set. Aggarwal [8] proposed a 2-D hashing scheme to construct an "in-memory" summary for sequentially presented graphs and used a simple heuristic to select a set of most discriminative frequent patterns to build a rule-based classifier. Although their method [8] can achieve good performance for graph stream classification, it has two major limitations.

1) The selected subgraph-patterns are composed with disconnected edges, which may have less discriminative capability than connected subgraph-patterns due to a lack of semantic meaning.
2) The computational cost is high because an additional frequent pattern mining procedure is required to perform on the summary table which contains massive transactions.

Our previous work, discriminative clique hashing (DICH) [10], has addressed these limitations to a certain extent by employing a fast clique detection algorithm from hashed graphs. However, DICH also has two limitations in handling graph streams: 1) the hashing space of DICH is fixed, so it cannot adapt to changing nodes and structures in the graph streams and 2) the predictor of DICH is based on simple decision rules, which makes it inefficient for handling concept drifts in graph streams. To address the challenge of concept drifting, an instance weighting mechanism has been proposed in gSLU [12] to adjust the subgraph feature selection module for emerging concept drifting graphs. However, gSLU is a frequent subgraph pattern mining-based framework, so it cannot achieve real-time responses for graph stream classification.

Motivated by the aforementioned challenges and the limitations of the existing approaches, in this paper we propose an adaptive real-time method for graph stream classification by integrating differential hashing, stochastic learning, and a batch-incremental learning mechanism to build a real-time classification framework for graph streams. More specifically, to tackle the increasing graph volumes (Challenge #1), the whole graph stream is partitioned into a number of nonoverlapping graph chunks each containing the same number of graphs. For each chunk, we employ a random hashing scheme to compress the original node set of the graphs for fast feature detection. By doing so, even if the node space of the graphs may continuously change, the hashing module will ensure that nodes are mapped to a confined space for learning (Challenge #2). To tackle concept drifting in graph streams (Challenge #3), a differential hashing scheme is used to map unlimitedly increasing features (cliques) onto a fixed-size feature space. The distribution of the hashed cliques in each received graph is used as a feature vector for stochastic learning of the real-time chunk classifier. Then, a chunk level weighting mechanism is used to form an ensemble for graph stream classification. The proposed method substantially speeds up the graph feature extraction process, solves the unlimited graph feature expanding problem, and effectively adapts to the concept drifting in graph stream classification. The contributions of this paper are highlighted as follows.

1) We propose a fast graph feature extraction method by detecting cliques from compressed graphs via hashing. The method significantly improves the efficiency of feature extraction and online learning to satisfy the real-time requirement for graph stream classification.
2) We propose a graph feature reduction method by mapping unlimitedly expanding clique patterns onto a fixed-size compatible feature space via differential hashing. This can avoid a prescan of graphs to further speed up the learning process, satisfy the one-pass requirement, and also benefit to concept drifts.
3) We incrementally train an online graph classifier, which achieves better classification performance than majority voting methods used in the previous research [8], [10].
4) Combined with the differential hashing scheme, we propose a chunk level weighting mechanism to form a weighted classifier ensemble for graph stream classification, which can effectively adapt to concept drifts and achieve better performance than the instance-level weighting mechanism [12].

The remainder of this paper is organized as follows. Section II introduces related work. Section III formally defines the graph stream classification problem, followed by the proposed graph stream classification method in Section IV. Experimental results are reported in Section V and we conclude the paper in Section VI.

II. Related Work

A. Graph Stream Classification

Many existing studies of graph classification focus on designing efficient kernels for measuring graph similarity. Most of them are based on the similar idea of extracting substructures from graphs to compare their co-occurrences. Typical substructures for describing graphs include walks [1], [13], paths [2], subtrees [3], [4], and subgraphs (usually based on a frequent subgraph mining technique, e.g., [14]). For walk-based approaches, Kashima *et al.* [1] proposed a kernel between graphs with vertex labels and edge labels within the framework of the marginalized kernel, where the random walk comparison can be reduced to a system of linear simultaneous equations. Vishwanathan *et al.* [13] further proposed a unified framework for the random walk and marginalized kernels, and extended linear algebra to efficiently compute these kernels by exploiting common structures inherent in these problems. For path-based approaches, Borgwardt and Kriegel [2] proposed a graph kernel based on the shortest paths, which require polynomial time to compute, but it cannot be applied to the graphs whose edges do not represent distances but attributes. For subtree-based approaches, Mahé and Vert [3] proposed a family of graph kernels based on the detection of common subtree patterns as features to represent graphs. Shervashidze and Borgwardt [4] designed a fast subtree kernel on graphs with the ability to deal with node labels. In this paper, we extract cliques (fully connected subgraphs) as features to describe a graph.

Our problem is also related to data stream mining, where the rapid generation of continuous data streams brings not only unique opportunities but also new challenges to data mining researchers [15]–[17]. Babcock *et al.* [18] proposed a general architecture for a data stream management system. Muthukrishnan [19] discussed the emerging area of algorithms for processing data streams and associated applications. Mining high-speed data streams was first studied in [20], which learns online from the high-volume data streams using constant memory and constant time per example. Later the idea of using ensemble learning for data stream classification was proposed in [21], which introduces an ensemble-based solution that can facilitate anytime learning on a problem of arbitrary size. A classical ensemble learning framework for addressing the concept drifting problem in data stream mining was proposed in [22]. Bifet and Gavaldà [23] and Brzezinski and Piernik [24] proposed to handling XML streams. In order to address the concept drifting problem in graph streams, both [12], [25] adopted an instance weighting mechanism to adjust the subgraph feature selection module for emerging concept drifting graphs. However, graph stream is generally defined on a massive universe of nodes. It is extremely time-consuming and memory intensive to enumerate subgraph features from such a huge node set. In our graph stream classification problem, instead of using subgraphs, we use cliques as features to represent graphs. Combined with a novel hashing scheme, named differential hashing, we adopt a chunk level weighting mechanism to form a weighted classifier ensemble for graph stream classification. By integrating stochastic learning [26], we incrementally update the underlying graph classifier using extracted feature representations of continuously received graphs. The advantage of using stochastic learning is threefold: 1) more suitable for the streaming situation; 2) simpler in terms of the model itself; and 3) faster in learning.

B. Hashing for Structured Data

There have been a handful of research works on structured data hashing. Shi *et al.* [27] introduced a kernel for hashing sequential data and sparse data. Li *et al.* [9] proposed the nested subtree hash kernel (NSHK) for recursively hashing node labels of a graph; however, NSHK can be hardly applied to unlabeled graphs generated on a dynamic network. Recently, some studies introduce hashing techniques for specific structured data, such as hierarchical data [28], tree-structured data [29], nested-set data [30], image-structured data [31] and graph-structured data [8], [10].

Although hashing techniques have been introduced for dealing with structured data, very few methods have considered hashing graphs generated on a dynamic network stream. Besides, our previous work [8], [10] also considers this problem setting. It employs a 2-D hashing scheme to construct an in-memory summary for sequentially received graphs. The first random-hash scheme is used to reduce the size of the edge set. The second min-hash scheme is used to dynamically update a number of hash-codes, which is able to summarize the frequent patterns of co-occurrence edges in the graph stream observed thus far. In this paper, we propose an approximate clique detection method and graph feature reduction method, both based on hashing techniques, for solving the same problem with a better performance in terms of both classification accuracy and learning efficiency.

III. Problem Formalization

A. Notations and Definitions

Definition 1 (Connected Graph): A graph is represented as $G = (\mathcal{V}, \mathcal{E}, \mathcal{L})$, where $\mathcal{V} = \{v_1, \ldots, v_V\}$ is the set of vertices, $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is the set of edges, and $\mathcal{L}$ is the label[1] set of the vertices and edges. A connected graph is a graph in which there exists a path between any pair of vertices.

Definition 2 (Graph Stream): A graph stream is defined on a dynamic network which comprises a massive universe of nodes, where the stream of graphs $\mathcal{S} = \{G_1, G_2, \ldots, G_n, \ldots\}$ generated one after another are represented as sets of interconnected edges on the dynamic network.

Definition 3 (Clique): A clique $C = (\mathcal{V}', \mathcal{E}', \mathcal{L}')$ in a graph $G = (\mathcal{V}, \mathcal{E}, \mathcal{L})$ is a subgraph that satisfies $\mathcal{V}' \subseteq \mathcal{V}$ and any pair of vertices in $\mathcal{V}'$ are connected (i.e., a complete graph).

Clique is widely used as a fundamental unit for structural analysis and knowledge discovery in graphs. Although finding maximum clique is NP-complete [32], many algorithms for finding cliques have been developed in exponential time and even polynomial time for a certain type of graphs. In this paper, we will employ the Bron–Kerbosch algorithm [33], which is very fast in practice, for clique detection in undirected graphs.

Definition 4 (Clique Features): Let $\mathcal{C} = \{C_1, \ldots C_M\}$ denote a set of clique patterns (or clique features). For a graph G_n, we use a vector $\mathbf{x}^{(n)} = [x_1^{(n)}, \ldots x_M^{(n)}]^\top$ to represent its clique-feature space spanned by $\mathcal{C}$, where $x_m^{(n)}$ is the number of clique pattern C_m found in G_n.

Instead of using frequent subgraph patterns, we propose to use cliques as features to represent graphs. This approach has two major advantages.

1) For dynamic network streams, frequent subgraph patterns are rapidly changing, which makes the feature

[1]Note that the labels of nodes and edges are different notations from the class labels of graphs.

TABLE I
KEY NOTATIONS USED IN THIS PAPER

Notation	Description
$\mathcal{V} = \{v_1, \ldots, v_V\}$	A set of vertices
$\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$	A set of edges
$\mathcal{L}$	A set of graph labels
$G = (\mathcal{V}, \mathcal{E}, \mathcal{L})$	A graph
$\bar{G}$	A compressed graph
$\mathcal{S} = \{G_1, \ldots, G_{n-1}, G_n, G_{n+1}, \ldots\}$	A graph stream
$C = (\mathcal{V}', \mathcal{E}', \mathcal{L}')$	A clique
$\mathcal{C} = \{C_1, \ldots C_M\}$	A set of clique patterns
$\mathbf{x}^{(n)} = [x_1^{(n)}, \ldots x_M^{(n)}]^\top$	Clique-feature space of G_n
$S_1, S_2, \ldots, S_n$	Sequential chunks
$\{\Gamma_{t-1}, \Gamma_{t-2}, \ldots, \Gamma_{t-K}\}$	K weighted chunk classifiers

space exponentially grow and drastically change. A classifier built from histogram graphs might not be used to classify future graphs because they have different subgraph feature space. Meanwhile, cliques are basic graph structures remaining relatively stable in graph streams. By using cliques as features to represent graphs, we can ensure historical and future graphs to have shared common feature space for learning.

2) Finding cliques is much more efficient than finding frequent subgraph patterns (we will elaborate our clique finding details in the next section), so our method can rapidly discover graph features for learning.

Key notations used in this paper are summarized in Table I.

B. Problem Setting

We consider a stream of graphs $\{G_1, G_2, \ldots, G_n, \ldots\}$ presented/received in a sequential order, where subscript n denotes the receiving order of the graph in the stream. So G_n denotes a graph which arrives right after the graph G_{n-1}. Specifically, the edge set of G_n is denoted by $\mathcal{E}_n = \{e_1, \ldots, e_E\} \subset \mathcal{E}$. Each graph G_n has a class label $l_n \in \{1, \ldots, L\}$. We represent each received graph G_n in the form of $\langle n, e_1, \ldots, e_E, l_n\rangle$.

In this paper, we assume that each edge in a graph has a numerical weight w_{ij}, where i and j are the indices of the two vertices of the edge (for simplicity, we do not consider edge labels). For the weight, in many applications, each edge in a graph has an associated non-negative value, which is usually called "weight." The meaning of the weight depends on the applications. It can be a distance between two nodes, or a capacity of an edge, etc.

Given the graph stream, our goal is to learn graph classification models from the graphs received thus far to accurately predict class labels of future test graphs G_{test} (test $> n$) which are yet to arrive, under the conditions that graphs in the stream can only be scanned once and the decision needs to be made in real-time. In this paper, the real-time constraint is not strictly enforced because different applications have their own interpretations and requirements for real-time response. Accordingly, we use the runtime efficiency to justify the performance of the algorithm for real-time response.

IV. GRAPH STREAM CLASSIFICATION

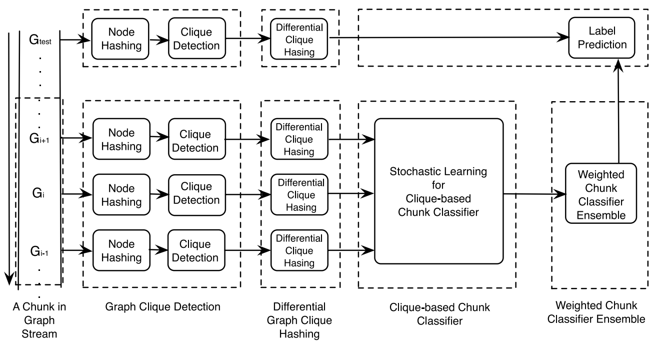
The framework of our method for graph stream classification is shown in Fig. 1, which includes four modules.

1) *Graph Clique Detection*: Each received graph G_n from the stream is first compressed into a small graph $\bar{G}_n$ using a random hashing scheme. Then, we employ a fast algorithm to decompose the compressed graph into a number of cliques as features to represent G_n.

2) *Differential Graph Clique Hashing*: Since the number of clique patterns (features) will unlimitedly increase as new graphs are fed in, the underlying feature space will keep expanding accordingly. Thus, in this stage, a clique hashing scheme is performed to map the unlimitedly emerged clique patterns onto a fixed-size clique-pattern set, denoted by $\mathcal{C} = \{\bar{C}_1, \ldots \bar{C}_M\}$. Now, all the graphs in the stream can be represented in this compatible space, for G_n, we have $\mathbf{x}^{(n)} = [x_1^{(n)}, \ldots x_M^{(n)}]^\top$.

3) *Clique-Based Chunk Classifier Learning*: For each chunk, the hashed clique-pattern representation $\mathbf{x}^{(n)}$ of each received graph and its corresponding class label l_n are used to incrementally update the underlying chunk classifier online, using a stochastic learning algorithm.

4) *Weighted Chunk Classifier Ensemble*: In order to adapt to the concept-drifting in graph streams, we propose the weighted chunk classifier ensemble to reduce the impact of concept drifting on the classification performance. In this module, multiple weighted chunk classifiers form an ensemble to predict future graph labels.



Gtest
Node Hashing
Clique Detection
Differential Clique Hasing
Label Prediction
Gi+1
Gi
Gi-1
Stochastic Learning for Clique-based Chunk Classifier
Weighted Chunk Classifier Ensemble
A Chunk in Graph Stream
Graph Clique Detection
Differential Graph Clique Hashing
Clique-based Chunk Classifier
Weighted Chunk Classifier Ensemble

Fig. 1. Framework of the proposed adaptive real-time graph stream classification method.

To classify a test graph G_{test} in the future graph stream, G_{test} is processed in the first two modules and the obtained hashed clique-pattern representation $\mathbf{x}^{\text{test}}$ is input to the ensemble classifier for class label prediction. The detailed approaches used in the four modules are introduced as follows.

A. Graph Clique Detection

As introduced in Section III-B, a graph in the stream is a connected subset of edges of the dynamic network. For dynamic networks with constantly evolving new nodes and edges, the edge set of the complex network can be extremely large, which makes it inefficient, or even infeasible, to detect cliques from such a large network in real-time. Therefore, it is necessary to sketch the graph stream and accelerate the clique detection (feature extraction) process. To this end, we first employ a random hashing scheme to compress each graph G_n on the large network into a small graph $\bar{G}_n$

$$\bar{G}_n := \text{Graph-Hash}\big(G_n, \bar{V}\big) \tag{1}$$

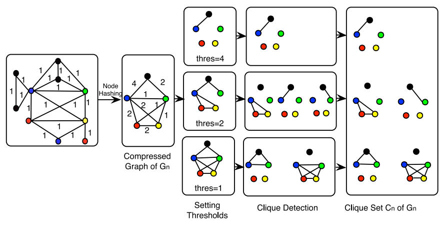


Fig. 2. Example of clique detection on a compressed graph.

where $\bar{V}$ denotes the desired number of vertices in the compressed network. The graph hashing operation includes two steps: first, we use a random hash function

$$\hbar : \{1, \ldots, V\} \mapsto \{1, \ldots, \bar{V}\} \tag{2}$$

to map the indices of the vertices in G_n to $\{1, \ldots, \bar{V}\}$ ($\bar{V} < V$) as the set of vertices in $\bar{G}_n$. Second, for each edge in $\bar{G}_n$, its weight $\bar{w}_{ij}$ is calculated as follows:

$$\bar{w}_{ij} = \sum_{i=\hbar(p), j=\hbar(q)} w_{pq} \tag{3}$$

The above graph compression in Eq. (1) enjoys a zero false-negative property: if two graphs G and G' on the original network have a same subgraph g, the compressed graphs $\bar{G}$ and $\bar{G}'$ also have the same subgraph $\bar{g}$. This property implies that, if two original graphs are similar in terms of subgraphs, the compressed graphs are also similar. We may have collisions (i.e., false-positive) that different subgraph patterns in the original graphs result in the same subgraph pattern in the compressed graphs. In reality, the probabilities of such cases are rather low because the collided edges in the compressed graphs are unlikely to form a connected graph (we only extract cliques as features which are fully connected subgraphs). The leftmost two columns in Fig. 2 illustrate this operation.

After obtaining compressed graph $\bar{G}_n$, we employ a fast algorithm to detect cliques in $\bar{G}_n$. To this end, we adopt the graphlet basis estimation algorithm used in [34]. The first step for clique detection is to apply different thresholds to compressed graph $\bar{G}_n$ by using various weight levels in a descending order, say $t \in \{\max(\bar{G}_n), \ldots, \min(\bar{G}_n)\}$, where $\max(\bar{G}_n)$ and $\min(\bar{G}_n)$ denote the largest and the smallest edge weights in $\bar{G}_n$, respectively. We define this graph thresholding operation as

$$\bar{G}_n^{(t)} := \mathbf{1}(\bar{G}_n \geq t) \tag{4}$$

where $\mathbf{1}(\cdot)$ denotes an indicator function, which sets the weight of an edge in $\bar{G}_n^{(t)}$ to be 1, i.e., $\bar{w}_{ij}^{(t)} = 1$, if $\bar{w}_{ij} \geq t$. We use the Bron–Kerbosch algorithm [33] to identify all the cliques from $\bar{G}_n^{(t)}$ at each threshold. The Bron–Kerbosch is an algorithm for finding maximal cliques in an undirected graph. The union set of the cliques found in $\{\bar{G}_n^{(t)}\}_{t=\min(\bar{G}_n)}^{\max(\bar{G}_n)}$ is represented as the clique set for G_n. This procedure is detailed in Algorithm 1.

An example of clique detection is illustrated in Fig. 2. After graph hashing, we obtain the compressed graph of G_n (second col). Then, four weight thresholds $\{4, 3, 2, 1\}$ are set to generate three graphs $\{\bar{G}_n^{(1)}, \bar{G}_n^{(2)}, \bar{G}_n^{(4)}\}$ (third col). Note that $\bar{G}_n^{(3)}$ is not shown. Because in this example the edges with weight ≥ 3 are the same as the edges with weight ≥ 4, $\bar{G}_n^{(3)}$ is actually identical to $\bar{G}_n^{(4)}$. The Bron–Kerbosch algorithm is applied to detect a set of cliques from each graph (fourth col). Finally, all cliques detected are merged to form the clique set $\mathcal{C}_n$ for G_n as its feature representation (fifth col).

Algorithm 1 Graph Clique Detection

```
Input: $G_n$: a graph in the graph stream; $\hbar$: node hash function
Output: $\mathcal{C}_n$: the clique set detected from $\bar{G}_n$
1: $\bar{G}_n$ = Graph-Hash($G_n$, $\bar{V}$)
2: $\mathcal{C}_n = \emptyset$
3: for $t = \max(\bar{G}_n) : 1 : \min(\bar{G}_n)$ do
4:     $\bar{G}_n^{(t)} = \mathbf{1}(\bar{G}_n \geq t)$
5:     $\mathcal{C}_n^{(t)}$ := Bron-Kerbosch($\bar{G}_n^{(t)}$)
6:     $\mathcal{C}_n := \mathcal{C}_n \bigcup \mathcal{C}_n^{(t)}$
7: end for
```

B. Differential Graph Clique Hashing

The cliques extracted from the compressed graph are used as features to represent the original graph. To learn a graph classifier, features of all training graphs should be in the same feature space. In reality, since new graphs continuously arrive and update the stream, the number of unique clique patterns (features) will unlimitedly grow and expand the underlying feature space. Meanwhile, in dynamic graph streams, the data distributions may gradually or abruptly change, leading to "concept drifts" [35] in the stream with entirely different feature space. To address these problems, we propose a differential feature hashing scheme to combine "new cliques" and "old cliques," and constraint the dimensionality of the feature space. Both the "new and old cliques" and "differential feature hashing" can help the learner adapt to changes in the stream and alleviate the impact of the concept drifts.

The differential clique hashing scheme is applied to both old and new cliques. It not only helps control the feature space, but also provides capability to identify emerging new features in the stream, so our algorithm differentially processes features in order to tackle the concept drifts. In the following, we first define new clique versus old clique and further address the differential clique hashing.

In order to tackle the increasing volume of the graph stream, we partition the stream into a set of sequential chunks. We can thus efficiently train classifiers from small chunks. The ultimate goal is to learn an ensemble classifier to predict the class label of each test graph G_{test} in a future chunk, where an ensemble includes a number of weighted chunk classifiers. We set the ensemble size as K which represents the number of chunk classifiers in the ensemble. Whenever a clique is detected from graphs in the current chunk, we will check this clique in the previous K chunks. If the same clique appears in one of the previous K chunks, we regard this clique as an old clique; otherwise it is set as a new clique. Because the K chunks will vary according to the current chunk, "old" or "new" is a relative concept.

After cliques are detected from the current chunk, a differential clique hashing scheme is performed to map unlimited emerging clique patterns onto a fixed-size clique-pattern set,

Algorithm 2 Differential Graph Clique Hashing

```
Input: $\mathcal{C}_n = \mathcal{C}_{new} \cup \mathcal{C}_{old}$: the clique set detected from $\bar{G}_n$
Output: $\mathbf{x}^{(n)}$: the feature vector of $G_n$
1: $\mathbf{x}^{(n)} = [0, \ldots, 0]^\top$
2: for $k = 1$ : length($\mathcal{C}_{new}$) do
3:     $h_{new,k} = \hbar(\mathrm{str}(C_{new,k}), M * R)$
4:     if $h_{new,k} = m (1 \leq m \leq M * R)$ then
5:         $x_m^{(new)} = x_m^{(new)} + 1$
6:     end if
7: end for
8: for $k = 1$ : length($\mathcal{C}_{old}$) do
9:     $h_{old,k} = \hbar(\mathrm{str}(C_{old,k}), M * (1 - R)$
10:    if $h_{old,k} = m (1 \leq m \leq M * (1 - R))$ then
11:        $x_m^{(old)} = x_m^{(old)} + 1$
12:    end if
13: end for
14: $\mathbf{x}^{(n)} = \mathbf{x}^{(n)} / \sum_{m=1}^{M} x_m^{(n)}$
```

denoted by $\mathcal{C} = \{\bar{C}_1, \ldots \bar{C}_M\}$ (M is the range of clique hash values). In order to distinguish new cliques from old cliques in the clique-pattern set, we set a ratio value as R. For new cliques, the size of hash space is $M * R$, whereas for old cliques, the size of hash space is $M * (1 - R)$. The differential clique hashing scheme will map new and old cliques onto corresponding fixed-size clique-pattern subset. By adjusting the value of R, we can easily tradeoff the importance of new versus old feature space, such that the learner can quickly adapt to the concept drifting in the stream.

Given a graph G_n received from the graph stream, we first use Algorithm 1 to detect its new clique set $\mathcal{C}_{\mathrm{new}}$ and old clique set $\mathcal{C}_{\mathrm{old}}$. Then, for each new clique $C_{\mathrm{new},k}$ in $\mathcal{C}_{\mathrm{new}}$ and each old clique $C_{\mathrm{old},k}$ in $\mathcal{C}_{\mathrm{old}}$, we differentially apply a random hash function $\hbar(\cdot)$ to $C_{\mathrm{new},k}$ and $C_{\mathrm{old},k}$ to generate corresponding index $h_{\mathrm{new},k} \in \{1, \ldots, M * R\}$ and $h_{\mathrm{old},k} \in \{M * R + 1, \ldots, M\}$ as follows:

$$h_{\mathrm{new},k} = \hbar\big(\mathrm{str}\big(C_{\mathrm{new},k}\big), M * R\big) \tag{5}$$

$$h_{\mathrm{old},k} = \hbar\big(\mathrm{str}\big(C_{\mathrm{old},k}\big), M * (1 - R)\big) \tag{6}$$

where $\mathrm{str}(C_{\mathrm{new},k})$ denotes the string of the ordered node indices of $C_{\mathrm{new},k}$ and $M * R$ constraints the range of hash values in $\{1, \ldots, M * R\}$. We use a vector $\mathbf{x}^{(n)} = [x_1^{(n)}, \ldots x_M^{(n)}]^\top$ to represent a graph G_n, where $x_m^{(n)}$ is the frequency of cliques in $\mathcal{C}_n$ whose indices are m based on the clique hash function (5), (6). This procedure is detailed in Algorithm 2. As the result of the above process, all graphs in the stream can be represented in an M-dimensional compatible feature space.

C. Clique-Based Chunk Classifier

When training a classifier from each graph chunk, we first construct a feature vector $\mathbf{x}^{(n)} = [x_1^{(n)}, \ldots x_M^{(n)}]^\top$ for each received graph G_n using the above method, and then use continuously obtained feature vectors as training instances to learn a classifier. Since graphs have been represented as feature vectors, a classifier can be learned using any generic learning algorithm following a standardized learning procedure.

Suppose there are N graphs in the entire stream (N may be infinite). As each feature vector $\mathbf{x}^{(n)}$ is M-dimensional, we assume that the entire data set for training is a feature matrix $\mathbf{X} \in \mathbb{R}^{M \times N}$, where N columns correspond to N graphs and M rows correspond to M hashed clique patterns. We also construct an label matrix $\mathbf{Y} \in \{0, 1\}^{L \times N}$ for supervised learning, where L rows correspond to L classes; if G_n belongs to the lth class, $\mathbf{Y}_{l,n} = 1$; otherwise $\mathbf{Y}_{l,n} = 0$.

Because the feature vector $\mathbf{x}^{(n)}$ of G_n actually describes the distribution of the M clique patterns, we adopt a regularized ridge regression model to fit label distributions of G_n

$$Q(\mathbf{W}) = \frac{1}{2}\|\mathbf{W}\mathbf{X} - \mathbf{Y}\|_2^2 + \frac{\lambda}{2}\|\mathbf{W}\|_2^2 \tag{7}$$

where $\mathbf{W} \in \mathbb{R}^{L \times M}$ is the weight matrix. We aim to minimize the following objective function to estimate $\mathbf{W}$:

$$\mathbf{W}^* = \arg\min_W Q(\mathbf{W}) \tag{8}$$

which is a linear least-squares problem.

Many methods can be used to solve (8), such as gradient descent and Newton's method. The standard (or "batch") gradient descent method will perform the following iterations:

$$\mathbf{W} := \mathbf{W} - \alpha\nabla Q(\mathbf{W}) = \mathbf{W} - \alpha\sum_{n=1}^{N}\nabla Q_n(\mathbf{W}) \tag{9}$$

where α is the step size (i.e., learning rate) and

$$Q_n(\mathbf{W}) = \frac{1}{2}\left\|\mathbf{W}\mathbf{x}^{(n)} - \mathbf{y}^{(n)}\right\|_2^2 + \frac{\lambda}{2N}\|\mathbf{W}\|_2^2 \tag{10}$$

where $\mathbf{x}^{(n)}$ and $\mathbf{y}^{(n)}$ are the nth columns of $\mathbf{X}$ and $\mathbf{Y}$, respectively.

In our problem setting, these training pairs $(\mathbf{x}^{(n)}, \mathbf{y}^{(n)})$ for $n = 1, 2, \ldots$ are obtained in sequence rather than in batch. We cannot simply use the standard gradient descent method to optimize $\mathbf{W}$. Therefore, we resort to the stochastic gradient descent method to solve this problem, which updates $\mathbf{W}$ using a single example at each iteration, for $n = 1, 2, \ldots$

$$\mathbf{W} := \mathbf{W} - \alpha\nabla Q_n(\mathbf{W}) \tag{11}$$

where

$$\nabla Q_n(\mathbf{W}) = \mathbf{W}\mathbf{x}^{(n)}\Big[\mathbf{x}^{(n)}\Big]^\top - \mathbf{y}^{(n)}\Big[\mathbf{x}^{(n)}\Big]^\top + \frac{\lambda}{N}\mathbf{W}. \tag{12}$$

Based on the iteration function (12), the linear regression model $\mathbf{W}$ is updated online at the arrival of each new graph in the stream. After finishing updating $\mathbf{W}$ for the last graph in a chunk, we use the obtained $\mathbf{W}$ as the current chunk classifier. By doing so, the updating for $\mathbf{W}$ can be stopped at any time for testing tasks, in order to meet the real-time decision requirement.

D. Weighted Chunk Classifier Ensemble

To classify a test graph, we adopt a weighted chunk classifier ensemble. We use $S_1, S_2, \ldots, S_n$ to represent sequential chunks, each contains the same number of graphs, and S_n is the most up-to-date chunk. For each chunk S_i ($1 \leq i \leq n$), we use Γ_i to denote the classifier trained from chunk S_i. We set the ensemble size as K to use K consecutive chunk classifiers for prediction. For the current testing chunk S_t, we use the previous K weighted chunk classifiers $\{\Gamma_{t-1}, \Gamma_{t-2}, \ldots, \Gamma_{t-K}\}$ as an ensemble to predict the class label of each graph in the current testing chunk S_t separately. The process is as follows.

1) A weight value W_i is assigned to each individual chunk classifier Γ_{t-i} $(1 \leq i \leq K)$.
2) Each chunk classifier Γ_{t-i} is used to predict the class label of a test graph G_{test}, which results in K predicted labels for the test graph from K chunk classifiers.
3) We sum up all the weight values of each same label and choose the label with the largest aggregated weight as the final label of G_{test}.

More specifically, given a test graph G_{test}, we first detect its cliques using Algorithm 1, and then differentially hash its cliques to construct the clique-pattern feature vector $\mathbf{x}^{\text{test}}$ using Algorithm 2. After that, we calculate its class-label distribution vector $\mathbf{y}^{\text{test}}$ using

$$\mathbf{y}^{\text{test}} = \mathbf{W}\mathbf{x}^{\text{test}} \tag{13}$$

according to which the class label of G_{test} is predicted by

$$l_{\text{test}} = \arg\max_{l}\{y_1^{\text{test}}, \ldots, y_L^{\text{test}}\} \tag{14}$$

where y_l^{test} denotes the value of the lth dimension of $\mathbf{y}^{\text{test}}$.

For the current testing chunk S_t, according to the hash space ratio R, the weights $\{W_{t-1}, \ldots, W_{t-K}\}$ of the previous K classifiers $\{\Gamma_{t-1}, \ldots, \Gamma_{t-K}\}$ are set to $\{(1-R)^1, \ldots, (1-R)^K\}$. Such weight setting ensures that the closer a chunk to the testing chunk S_t, the larger the chunk weight is.

E. Theoretical Analysis

The proposed method uses random hashing and graph clique detection to compress and represent graphs for classification. One possible concern is that since the hashing scheme randomly maps a node in the original graph to a node in the compressed graph, it may lose information encoded in the edges and eventually lose important substructure features in the original graph for classification.

Consider our problem setting, where a graph stream is defined on a dynamic network. After random hashing of the nodes, the resulting compressed graphs only lead to aggregated weights of multiple edges in the original graph. If we detect cliques from such a "compressed" graph, we can ensure: 1) *no false negatives* and 2) *theoretically guaranteed false positives*.

False Negative: The proposed clique detection algorithm based on a randomly compressed graph $\bar{G}_n$ (Algorithm 1) will not bring false negatives. If there exists a clique in the original graph G_n, the clique must also exist in the compressed graph $\bar{G}_n$ (under a mild constraint that neighboring nodes are not allowed to hashed to a same node in $\bar{G}_n$).

False Positive: Algorithm 1 may introduce false positives. We show below that the expected error can be dramatically reduced if the size of the dynamic network and the size of cliques are sufficient large whereas the average degree of nodes is not too large compared to the size of the network: suppose we have a compressed graph obtained by $\bar{G}_n :=$ Graph-Hash$(G_n, \bar{V})$, where the probability of a node connecting to another in G_n is $p \in [0, 1]$ (thus the average degree of node is pV). For convenience, we use v to denote a vertex in G_n and $\bar{v}$ as a vertex in $\bar{G}_n$. The probability of v in $\bar{v}$ is not connected to another $\bar{v}'$ is $(1-p)^{V/\bar{V}}$; the probability that any v in $\bar{v}$ is not connected to any v in $\bar{v}'$ is $[(1-p)^{V/\bar{V}}]^{V/\bar{V}}$. Thus, the probability that two vertices in $\bar{G}_n$ are connected is $1-(1-p)^{V^2/\bar{V}^2}$. Since the proposed algorithm is to detect cliques (complete graphs), the probability to generate a false positive of a k-clique is $[1-(1-p)^{V^2/\bar{V}^2}]^{\binom{k}{2}} \approx [1-e^{-pV^2/\bar{V}^2}]^{\binom{k}{2}}$. For example, if $V = 100\,000$, $\bar{V} = 5000$, $p = 0.001$, and $k = 4$ (which means that the number of nodes in the compressed graph is 1/20 of the original graph), the probability of generating a false positive 4-clique is only 0.00128.

V. EXPERIMENTS

We empirically validate the performance of the proposed classifier adaptive real-time classification for graph stream (ARC-GS) on three real-world graph streams and one synthetic graph stream. In particular, in Sections V-C and V-D, we will evaluate the effectiveness and efficiency of ARC-GS under different configurations, respectively. In Section V-E we will validate the robustness of ARC-GS with various types of concept drifts (abrupt, gradual, and recurrent).

A. Benchmark Data Sets

1) *IBM Sensor Stream*[2]: This data stream records the local traffic information of a sensor network. The IP-addresses represent nodes and local traffic flows are edges. Each graph is associated with a particular intrusion type and there are over 300 different intrusion types (classes) in the data set. Our goal is to classify a traffic flow pattern into one intrusion type. Because the number of classes is very large (> 300), and many of them are rarely observed, we select 50 most dense classes in our experiments for multiclass classification. The data set contains 1.0×10^6 nodes, 1.25×10^6 edges, which generate a stream of 5.0×10^5 graphs over time.
2) *Citation Network Stream*[3]: In this citation network, each paper is represented as a graph with each node representing a paper ID and each edge representing citation relationship between papers. The stream is formed according to the chronological order of papers. In our experiment, we select 16 000 papers with authors and references attribute information from two research areas, artificial intelligence and computer vision (CV), to generate a graph stream for binary classification. The edges are citations between papers and co-authorships between authors. The final data stream contains 7.1×10^4 nodes, 5.8×10^4 edges, and 1.6×10^4 graphs.
3) *DBLP Graph Stream* [4]: The DBLP stream [12] 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. To build a graph stream, we select a list of conferences and

[2]http://www.charuaggarwal.net/sens1/gstream.txt
[3]http://arnetminer.org/citation
[4]https://snap.stanford.edu/data/com-DBLP.html

use the papers published in these conferences (in chronological order) to form a binary-class graph stream. The classification task is to predict whether a paper belongs to database and data mining or CV and pattern recognition field. 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. The final graph stream contains 1.9×10^5 nodes, 3.7×10^5 edges, and 1.875×10^4 graphs.

4) *GTGraph Stream* [5]: This graph stream is a synthetic data set generated by the graph generator GTGraph based on R-MAT model [36]. We choose default parameter values suggested by the authors during network generation. Our GTGraph network contains 6.0×10^5 nodes and 5.0×10^5 edges, and the edges from the same node are used as a graph for experimental evaluation, which generate a stream containing 1.0×10^5 graphs over time.

We divide each stream into 25 nonoverlapping chunks. For citation network stream (CNS), DBLP, IBM, and GTGraph, each chunk contains 640, 750, 20 000, and 4000 graphs, respectively.

B. Baseline Methods

1) *2-D [8]*: This method uses a 2-D hashing scheme to construct an in-memory summary for the sequentially presented graphs. The first random-hash scheme is used to reduce the size of the edge set, and the second min-hash scheme is used to dynamically update a number of hash-codes. Finally, a simple heuristic is used to select a set of most discriminative frequent patterns to build a rule-based classifier.
2) *DICH [10]*: This method employs a fast algorithm to decompose a compressed graph into a number of cliques to sequentially extract clique-patterns over the graph stream as features. Two random hashing schemes are used to compress the original edge set of the graph stream and map the unlimitedly increasing clique-patterns onto a fixed-size feature space, respectively.
3) *gSLU [12]*: This method employs an ensemble-based framework to partition graph streams into a number of graph chunks each containing some labeled and unlabeled graphs, and then uses unique measures to discover informative subgraph features with minimum redundancy. Meanwhile, an instance weighting mechanism is adopted to emphasize on emerging concept drifting graphs.

C. Effectiveness Evaluation

For all the methods, we investigate their performance with respect to: 1) the number of features M and 2) the ensemble size K. For the proposed ARC-GS classifier, we set hash ratio R to 0.2, which means that 20% of hashing space is used for new cliques discovered from the current chunk. In addition to a fixed value of R, we also investigate the performance of ARC-GS with different hash ratio values R.

[5]http://www.cse.psu.edu/~madduri/software/GTgraph/

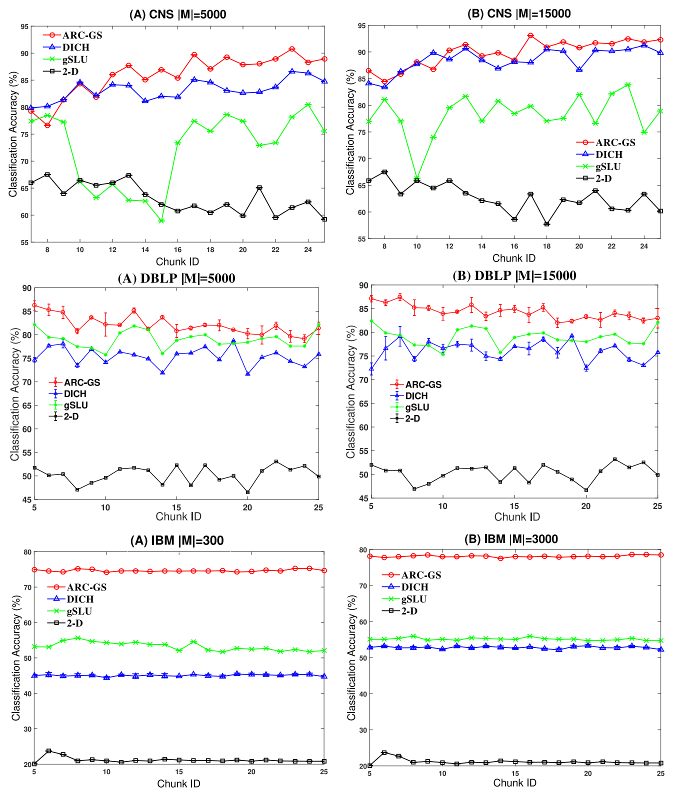


Fig. 3. Classification accuracy on the CNS (upper row, ensemble size $K = 6$), the DBLP (middle row, ensemble size $K = 4$), and the IBM (bottom row, ensemble size $K = 4$) with different numbers of features M.

Number of Features M: For ARC-GS and DICH classifiers, M represents the fixed size of hashed clique-pattern set. For 2-D classifier, M represents the number of discriminative patterns in the underlying graph used by the 2-D hashing scheme. For gSLU classifier, M represents the number of minimum-redundancy subgraph features.

Ensemble Size K: If the current testing chunk is S_T, our ARC-GS classifier will use the most recent K weighted chunk classifiers $\{C_{T-1}, C_{T-2}, \ldots, C_{T-K}\}$ as an ensemble to predict graphs in S_T; the gSLU classifier will build an ensemble of classifier from the most recent K chunks to predict graphs in the S_T; the 2-D and DICH classifiers will combine the most recent K chunks $\{S_{T-1}, S_{T-2}, \ldots, S_{T-K}\}$ as training data to train a classifier and predict graphs in S_T.

Hash Ratio R: For our ARC-GS classifier, R represents the proportion of allocated feature hashing space for new cliques, compared to the whole hashing space.

1) Performance With Respect to the Number of Features M: In this experiment, we fix the ensemble size K ($K = 4$ for IBM, $K = 6$ for CNS, and $K = 4$ for DBLP) and adjust the number of features M to evaluate the effectiveness of the methods. For IBM, we investigate the number of features M in {300, 1000, 3000}. For CNS and DBLP, we investigate the M in {5000, 10 000, 15 000}.

Fig. 3 reports the classification accuracy (y-axis) with respect to the chunk ID (x-axis) on CNS, DBLP, and IBM by using different numbers of features. The average classification accuracy over the CNS and IBM streams is reported separately

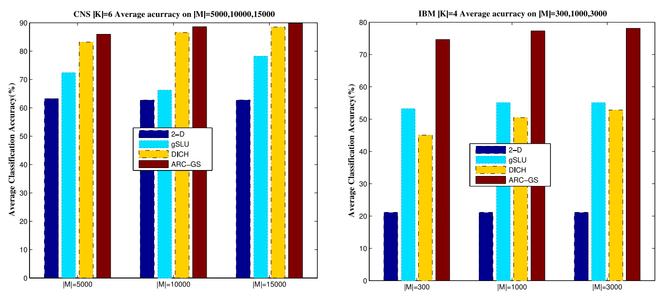


Fig. 4. Average accuracy on the CNS (left, ensemble size $K = 6$) and the IBM (right, ensemble size $K = 4$) with different numbers of features M.

in Fig. 4. We can see that the overall classification performance of ARC-GS is the best among the four compared methods on the three streams under different settings of M across the whole stream. For DBLP and IBM streams, the accuracy of ARC-GS is always higher than 2-D, gSLU, and DICH at all chunk IDs. For CNS stream, ARC-GS's accuracy is more stable than other classifiers, especially for gSLU with large M values. This is mainly attributed to the stochastic learning strategy adopted in ARC-GS which helps achieve better classification accuracy and real-time response. This experiment implies that ARC-GS can achieve significantly improved accuracy, compared to baselines.

Fig. 4 further validates that ARC-GS outperforms 2-D, gSLU, and DICH classifiers especially on the IBM stream. Among these classifiers, the effectiveness of the 2-D classifier is the worst. The reason is that the selected subgraph-patterns with disconnected edges may have less discriminative capability than connected subgraph-patterns used in the gSLU, DICH, and ARC-GS, because connected subgraph structure may have semantic meaning. For DICH and gSLU classifiers, DICH outperforms gSLU on CNS stream, but performs worse on IBM stream. The reason is that DICH is a majority voting classifier whose performance could degrade as the number of classes becomes large, and IBM stream has 50 classes which is much more than CNS stream.

An interesting observation is that a larger number of features can help improve the accuracies of ARC-GS and DICH classifiers on both streams. The reason is that a relatively large number of features would have better discriminative capability to differentiate graphs in different classes. We also find that the improvement of average accuracy becomes smaller with respect to the increase of the number of features. In IBM stream, when the number of features reaches 1000, all four classifiers have similar discriminative capability to classify graphs, and adding more features will not significantly help improve the accuracy. For gSLU classifier, when the number of features reaches 10 000, its average accuracy falls on the CNS stream. This is mainly because that those selected features are not beneficial and even misleading the classifier. For 2-D classifier, when the number of features reaches 300 and 5000, respectively, it already has enough discriminative capability to classify graphs on IBM and CNS streams.

2) Performance With Respect to the Ensemble Size K: In this experiment, we fix the number of features M ($M = 1000$ for IBM and $M = 10\,000$ for CNS and DBLP) and adjust the

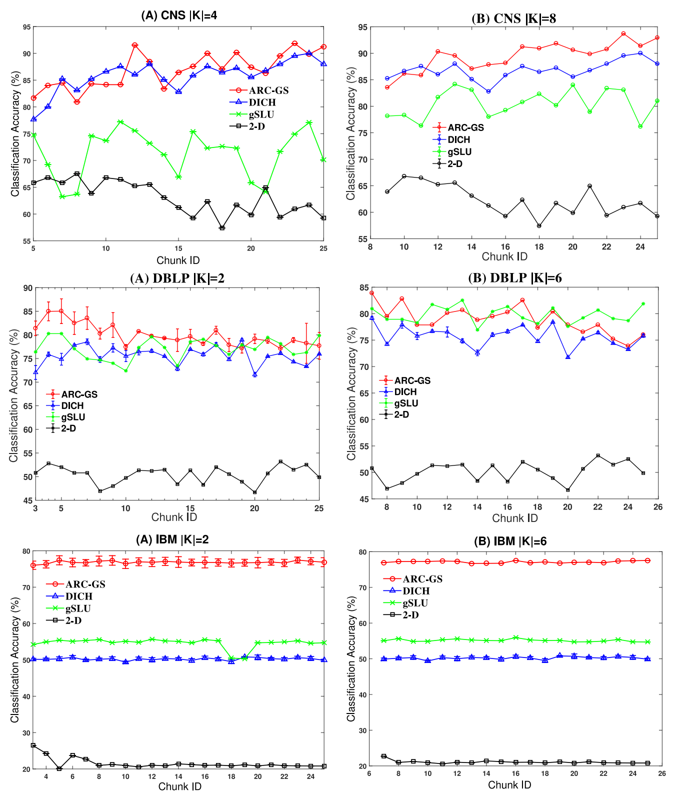


Fig. 5. Classification accuracy on CNS stream (upper row, number of features $M = 10\,000$), DBLP stream (middle row, number of features $M = 10\,000$), and IBM stream (bottom row, number of features $M = 1000$) with respect to different ensemble size K.

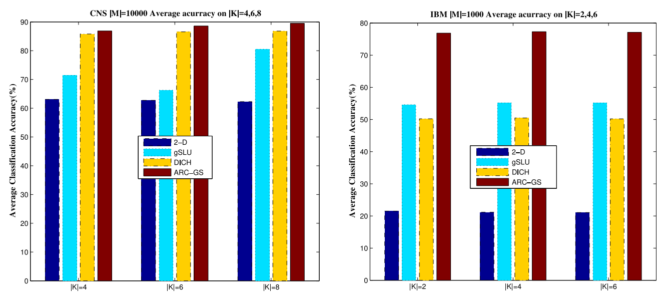


Fig. 6. Average accuracy on CNS stream (left, number of features $M = 10\,000$) and IBM stream (right, number of features $M = 1000$) with respect to different ensemble size K.

ensemble size K for effectiveness evaluation. For IBM and DBLP, we investigate the ensemble size K in $\{2, 4, 6\}$. For CNS, we investigate the ensemble size K in $\{4, 6, 8\}$.

Fig. 5 reports the classification accuracy (y-axis) with respect to the chunk ID (x-axis) on all four streams by using different ensemble sizes. The average classification accuracy over the entire streams is reported in Fig. 6, which further demonstrates that ARC-GS always outperforms DICH and gSLU and significantly outperforms 2-D hash compressed stream classifier, especially on IBM sensor stream. The reason is that a larger number of classes in the IBM stream provide a more challenging learning task to distinguish the classifier performance. By employing adaptive clique hashing to extract features for different classes, ARC-GS demonstrates

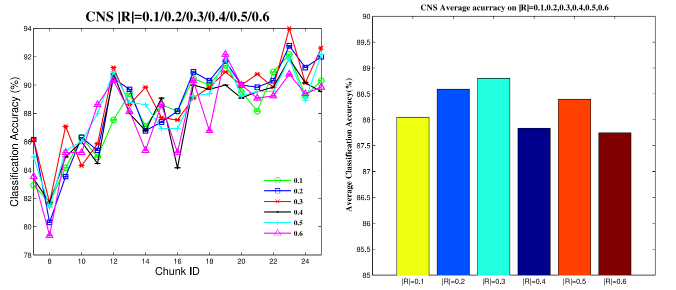


Fig. 7. Classification accuracy and average classification accuracy on CNS stream (number of features $M = 10\,000$ and ensemble size $K = 6$) with different hash ratio R.

good performance on IBM stream. Overall, we can find that a larger ensemble size helps improve the average classification accuracy, mainly because that a larger ensemble size will allow ARC-GS and DICH to accumulate more training graphs to generate better discriminative classification models. However, for 2-D classifier, the fixed number of features may be insufficient even if more training graphs are added, and a larger ensemble size may even decrease the average accuracy. For gSLU, its average accuracy is unstable under different ensemble sizes, especially on the CNS stream. This is mainly because that gSLU relies on frequent subgraph features for classification. As the graph stream continuously evolves, the selected features in gSLU may significantly change between chunks and result in unstable accuracy for training and classifying graphs.

Based on the overall evaluation results, we can conclude that ARC-GS can outperform DICH and gSLU, and significantly outperform 2-D hash compressed stream classifier in terms of the classification accuracy.

3) Performance With Respect to the Hash Ratio R: In this experiment, we fix the number of features $M = 10\,000$ and the ensemble size $K = 6$ for CNS, and adjust the hash ratio R for ARC-GS classifier in $\{0.1, 0.2, 0.3, 0.4, 0.5, 0.6\}$.

Fig. 7 reports the classification accuracy curves (y-axis) with respect to the chunk ID (x-axis) and average classification accuracy over the entire streams under different hash ratios. When the hash ratio R increases to 0.2 or 0.3, ARC-GS achieves the best performance. Overall, the classification performance of ARC-GS is relatively stable for different hash ratio values.

D. Efficiency Evaluation

Performance With Respect to the Number of Features M: In this experiment, we fix the ensemble size K ($K = 4$ for IBM, $K = 6$ for CNS) and adjust the number of features M for the efficiency evaluation. The experimental settings are the same as those in Fig. 3.

The average system runtime performance over two streams is reported in Fig. 8, which shows that the average runtime of ARC-GS and DICH are less than gSLU, and significantly less than 2-D classifier. The reason is that an additional frequent pattern mining procedure in the 2-D is required to perform on the summary table which comprises massive transactions. For gSLU, its subgraph search process is computationally much

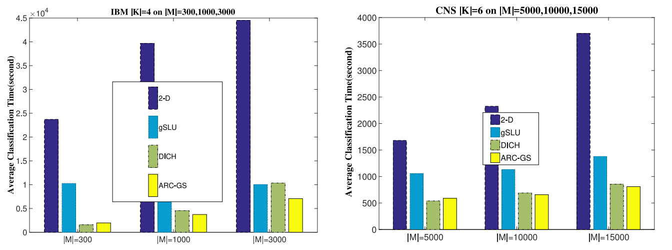


Fig. 8. Average runtime on IBM (left, ensemble size $K = 4$), and CNS streams (right, ensemble size $K = 6$) with respect to different numbers of features M.

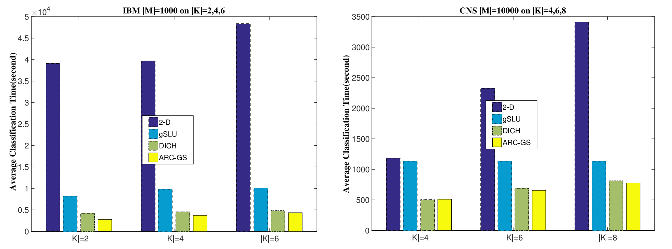


Fig. 9. Average runtime on IBM (left, number of features $M = 1000$), and CNS streams (right, number of features $M = 10\,000$) with respect to different ensemble size K.

more expensive than clique search process in both ARC-GS and DICH. As the number of features increases, the average runtime of all four classifiers also increase. This is because that the learning process needs extra time to discover and manage more features.

For both IBM and CNS streams, the average runtime of ARC-GS is close to DICH. As the number of features increases, the average runtime of ARC-GS increases gradually slower than DICH on IBM stream. When using $M = 3000$ features on the IBM stream, and using $M = 10\,000$ features on the CNS stream, the average runtime of ARC-GS is less than DICH, which shows that ARC-GS has better efficiency than DICH for large number features. This is mainly attributed to the incremental stochastic learning strategy in ARC-GS, which helps discover optimized weight values for high dimensional features for effective classification.

Performance With Respect to the Ensemble Size K: In this experiment, we fix the number of features M ($M = 1000$ for IBM and $M = 10\,000$ for CNS) and adjust the ensemble size K for efficiency evaluation. The experimental settings are the same as those in Fig. 5.

Fig. 9 reports the average system runtime performance of the four classifiers. Compared to 2-D classifier, ARC-GS, DICH, and gSLU require significant less runtime. The average runtime of the four classifiers increases as the ensemble size increases. This is because a larger ensemble size would result in more training graphs, which increase the training time accordingly. For the two benchmark streams, the overall average time of ARC-GS is close to DICH. For IBM stream, the overall average time of ARC-GS is less than DICH. When $K = 6$, the average runtime of ARC-GS is less than DICH on CNS stream. As K increases, the average runtime of ARC-GS is less than DICH. Therefore, ARC-GS has more stable and better efficiency as the ensemble size increases.

E. Concept Drifting Analysis

In this section, we extensively investigate the performance of different methods in handling various concept drifts, including temporary and permanent drifts. In particular, we evaluate the methods on four types of concept drifts.

1) *Temporary Abrupt Drift*: Under the temporary abrupt drift, the experimental data is from a single dataset, and the label distribution between two consecutive chunks within this dataset has an abrupt drift. This concept drift model includes a positive threshold parameter which determines how much drift is abrupt.
2) *Temporary Abrupt and Gradual Drifts*: This concept drift model is similar to the first above one. The only difference is that there occurs a gradual drift which means the change of the label distribution between two consecutive chunks is gradual. Also, a positive threshold parameter will be defined in this model to determine how much drift is gradual.
3) *Permanent Abrupt Drift*: For this drift type, the experimental data includes two datasets from different areas. There is a permanent abrupt change when the two consecutive chunks are from different datasets, and the first occurred dataset will never occur again.
4) *Recurrent Abrupt Drift*: In this drift, the experimental data also includes two datasets from different areas. The only difference with the permanent abrupt drift is that the two datasets will recurrently occur multiple times.

Next, the corresponding experiments on these four types of concept drift will be conducted to evaluate the performance of each algorithm. In our experiments, we evaluate the impact of the concept drifts on four classifiers in terms of: 1) the number of features M; 2) the ensemble size K; and 3) the chunk size. For ARC-GS, we adjust the hash ratio R within range $\{0.1, 0.2, 0.3, 0.4, 0.5, 0.6\}$ to evaluate the performance change. The default R in ARC-GS in other experiments is 0.2.

1) Results on Temporary Abrupt Drift: In this experiment, the IBM stream dataset is redesigned to bring in a temporary abrupt drift. We change the class distribution to simulate abrupt concept drifting on IBM graph stream. we select 50 different classes (1$\sim$50) of graphs from the whole IBM sensor stream (which contains 250 classes) as our experimental data. In our experiments, there are 25 chunks, each of which comprises 20 000 graphs. To introduce concept drift, we employ following approach. From chunks 1$\sim$14, and from chunks 18$\sim$25, graphs are randomly selected from original IBM experimental data containing 50 classes. After checking the overall class distributions in chunks 1$\sim$14 and chunks 18$\sim$25, we introduce abrupt concept drifts to chunk 15 by designing a highly different class distribution. After that, we gradually change the class distributions in chunks 16 and 17, which results in explicit but gradual concept drifts.

In the first group of experiments, we fix the ensemble size $K = 4$ and adjust the number of features M with $\{300, 3000\}$ for impact evaluation. And in the second group of experiments, the number of features M is fixed to 1000 and adjust the ensemble size K with $\{2, 6\}$.

Fig. 10 reports the classification accuracy (y-axis) with respect to the chunk ID (x-axis) by using different numbers

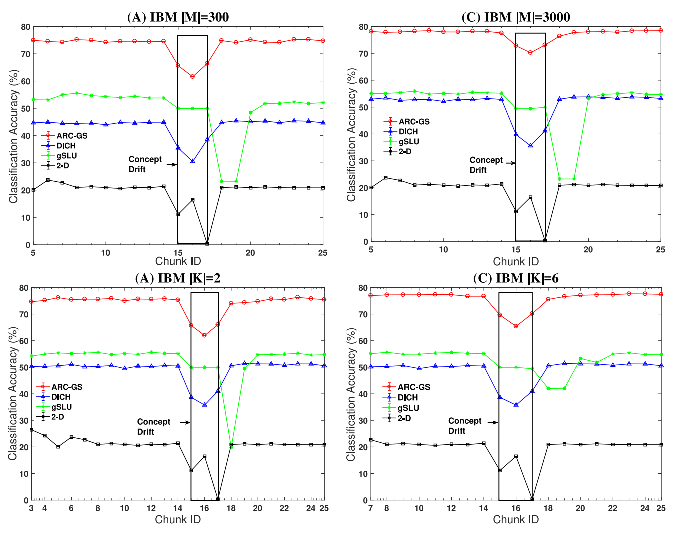


Fig. 10. Classification accuracy on the IBM stream for temporary abrupt drift. Upper row: results on different M and $K = 4$. Bottom row: results on different K and $M = 1000$.

of features. The results show that there is noticeable concept drifting from chunks 14$\sim$18 (marked by the rectangle boxes). As a result, all four classifiers experience performance loss. From chunks 14$\sim$15, all four classifiers experience large performance loss because there is an abrupt concept drift in chunk 15. For gSLU, there is a larger performance loss after the concept-drifting chunks. The reason is that the instance weighting mechanism employed by gSLU may be too sensitive to better adapt to the concept drifting, and the training results in the concept-drifting chunks mislead the classification of the following chunks. Among all classifiers, ARC-GS receives less loss than 2-D, gSLU, and DICH, which indicates that the overall impact of the concept drifts on ARC-GS is minimal among four compared classifiers. As the number of features increases, the impact of the concept drifts on the classification performance of ARC-GS becomes less significant. However, for DICH and 2-D classifiers, the impact almost remains the same. This experiment implies that ARC-GS can effectively handle temporary abrupt concept drift in graph streams.

2) Results on Temporary Abrupt and Gradual Drifts: In this experiment, we design a synthetic graph stream called GTGraph which includes the temporary abrupt and gradual drifts. We create a synthetic GTGraph stream with drifting concepts by changing parameters used to label the graphs in the stream. In the GTGraph network, we divide all nodes into d classes (d-dimensional space) and establish a hyperplane in d-dimensional space by $\sum_{i=1}^{d} f_i x_i = f_0$, where f_i denotes the ith feature weight, and x_i denotes the number of nodes in the ith feature in a graph. The feature weights f_i $(1 \leq i \leq d)$ are randomly initialized by values within the range [0, 1]. The f_0 is chosen to cut the graphs into two parts, that is, $f_0 = (1/2) \sum_{i=1}^{d} f_i$. Therefore, roughly a half of graphs are labeled as positive.

If the graph satisfies $\sum_{i=1}^{d} f_i x_i \geq f_0$, we label the graphs as positive; if the graph satisfies $\sum_{i=1}^{d} f_i x_i < f_0$, we label the graphs as negative. Accordingly, by changing f_0 value, we can simulate concept drifts in graph stream. In our examination,

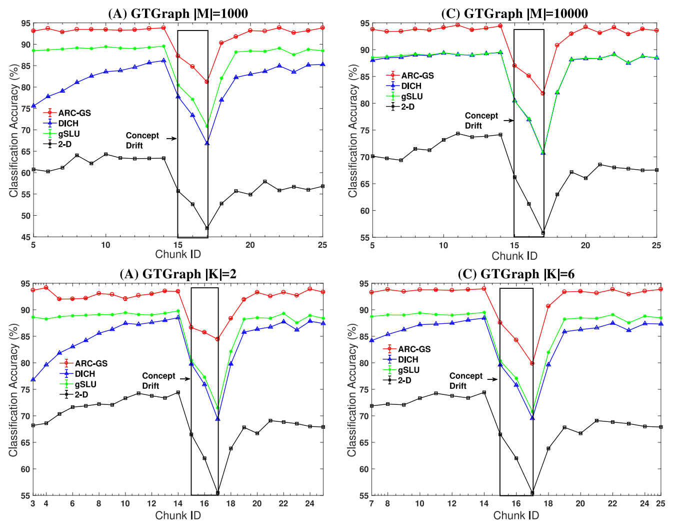


Fig. 11. Classification accuracy on the GTGraph stream for temporary abrupt and gradual drift. Upper row: results on different M and $K = 4$. Bottom row: results on different K and $M = 5000$.

we set d as 10, and insert abrupt concept drift in chunk 15 (the total number of chunks is 25) by introducing a significant change to f_0, and then gradually adjust f_0 to continuously introduce gradual concept drifts to chunks 16 $\sim$ 18.

In the first group of experiments, we fix the ensemble size $K = 4$ and adjust the number of features M with $\{1000, 10\,000\}$ for impact evaluation. And in the second group of experiments, the number of features M is fixed to 5000 and adjust the ensemble size K with $\{2, 6\}$.

Fig. 11 reports the classification accuracy (y-axis) with respect to the chunk ID (x-axis) in GTGraph streams by using different numbers of features. The results also show that there are noticeable concept drifting from chunks 14∼19 (marked by the rectangle boxes). From chunks 14∼15, all four classifiers also experience large performance loss because there is an abrupt drift when switching to chunk 15. Next from chunks 15∼17, the gradual drifts occur. This experiment implies that ARC-GS can effectively handle temporary abrupt and gradual concept drifts in graph streams.

3) Results on Permanent Abrupt Drift: In order to assess the performance on permanent abrupt concept drifts, we design a sudden permanent concept drift by combining the GTGraph stream (50 chunks * 2000 graphs) and the CNS stream (8 chunks * 2000 graphs).

Fig. 12 shows the experimental results on GTGraph + CNS stream. As expected, there is a decline in chunk 29 when the temporary drift occurs. When the concept permanently changes from GTGraph to CNS in chunk 50, all algorithms experience a sharp drop. However, ARC-GS algorithm recovers much faster than other algorithms. The results validate ARC-GS is better at handling permanent abrupt drift.

4) Results on Recurrent Abrupt Drift: In this experiment, we use the DBLP and CNS streams to simulate recurrent concept drifts. Specifically, we construct a graph stream by sequentially adding eight chunks of DBLP graphs (each chunk consists of 2000 graphs), following by eight chunks of CNS graphs. The procedure repeats three times to generate a stream that both DBLP and CNS concepts recur three times.

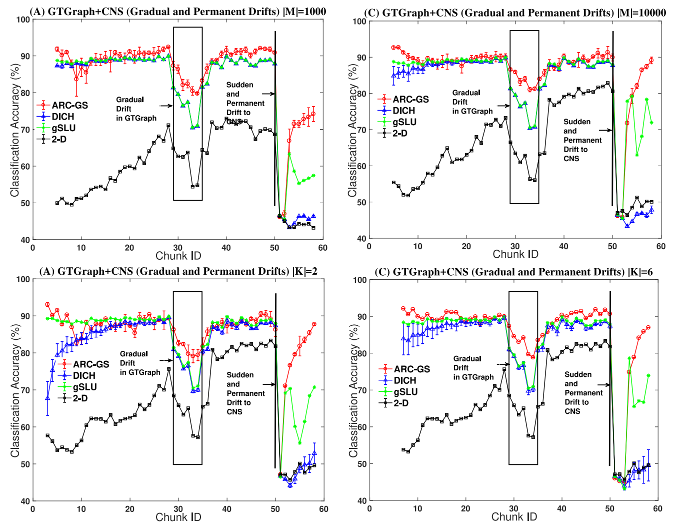


Fig. 12. Classification accuracy on the GTGraph+CNS stream for mixed temporary and permanent abrupt drift (chunk size is 2000). Upper row: results on different M and $K = 4$. Bottom row: results on different K and $M = 5000$.

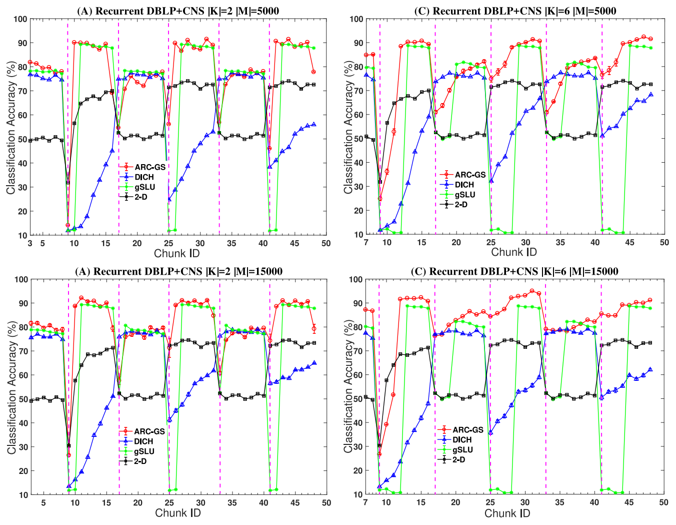


Fig. 13. Classification accuracy on the GTGraph+CNS stream for recurrent abrupt drift. Upper row: results on different K with $M = 5000$. Bottom row: results on different K with $M = 15\,000$.

Fig. 13 shows the experimental results for recurrent abrupt drift. The results show that when the concept change from DBLP to CNS (or from CNS to DBLP), all algorithms will drop significantly in their performance and then recover in following chunks. As K increases, gSLU will recover more slowly. However, GCS-CH will recover much faster and better than all other algorithms. This is because the GCS-CH algorithm assigns much larger weights into the most recent chunk, so it is better to capture the underlying concept drift.

F. Further Analysis

Results With Respect to Wider Range of Ensemble Size K: To better understanding the role of K, we experiment K from 2 to 20 with interval 2 in the dataset GTGraph-CNS. The experimental result is reported in Fig. 14. The result in Fig. 14 shows that when K keeps increasing, gSLU and DICH algorithms

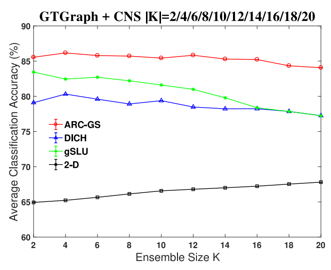


Fig. 14. Performance change over K in GTGraph-CNS ($M = 5000$ and chunk size $= 2000$).

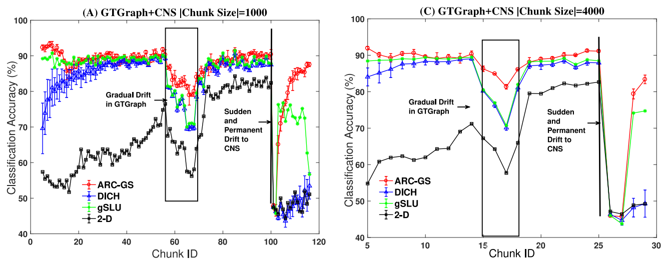


Fig. 15. Classification accuracy on the GTGraph-CNS ($M = 5000$ and $K = 4$) with different chunk size.

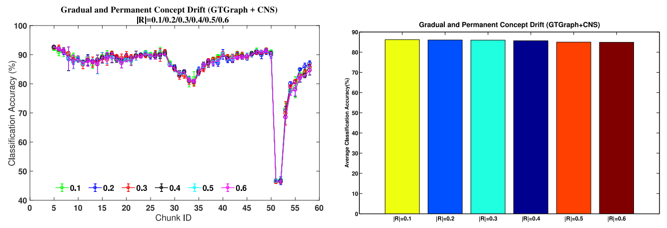


Fig. 16. Classification accuracy and average classification accuracy on GTGraph-CNS stream ($M = 5000$ and ensemble size $K = 4$) with different hash ratio R.

experience a slight drop rather than an increase in accuracy. This is because when the ensemble increase, the older classifiers in the ensemble will affect the algorithm (e.g., gSLU), so the ability of the algorithm to handle concept drift may decrease. However, our algorithm ARC-GS remains relatively stable when K increases, because it assigns larger weights on the most recent classifier and it is more sensitive in handling concept drift.

Results With Respect to the Chunk Size: We report the experimental results in Fig. 15 for different chunk size. The results show that no matter how to adjust the chunk size, our algorithm ARC-GS always performs the best.

Results With Respect to the Hash Ratio R: In this experiment, for ARC-GS, we adjust the hash ratio R within range $\{0.1, 0.2, 0.3, 0.4, 0.5, 0.6\}$ to evaluate the performance change in the dataset GTGraph-CNS. Fig. 16 reports our algorithm performs stable even under different ratio settings.

Results With Respect to Nonparametric Test: To summarize the comparison of all classifiers over all four datasets with concept drifts, we report a Friedman *post-hoc* test using Bergman and Hommel correction [37], [38]. In Table II, each value represents a raw *p-value* for each pair of classifiers. A summary with the average values of each classifier over all datasets is shown in Table II. From the result of Friedman *post-hoc* test, ARC-GS statistically outperforms the best.

TABLE II
NONPARAMETRIC TEST: FRIEDMAN *POST-HOC* TEST

	2-D	gSLU	DICH	*ARC-GS*
2-D	n/a	0.16570	0.58388	0.01555
gSLU	0.16570	n/a	0.17090	0.54664
DICH	0.58388	0.17090	n/a	0.04113
ARC_GS	0.01554	0.54664	0.04113	n/a
Summary	0.53250	0.71225	0.67025	***0.83150***

VI. CONCLUSION

In this paper, we proposed an adaptive real-time classification method for concept-drifting graph streams. We argued that graph stream classification has three major challenges: 1) increasing graph volumes; 2) expanding feature space; and 3) concept drifting. An effective graph classification model should tackle these challenges to classify graphs in real-time with only one-pass of the stream data. Accordingly, we employed two hashing schemes to speed up the graph feature extraction, and combined incremental stochastic learning strategy and chunk level weighting mechanism for graph stream classification. In particular, we proposed an approximate method for fast graph feature extraction by detecting cliques from compressed graphs via hashing, which significantly improves the efficiency of feature extraction to satisfy the real-time requirement. A graph feature reduction method is used to map expanding clique patterns onto corresponding fixed-size compatible feature spaces via differential hashing, which can avoid a prescan of graphs to address the one-pass and "concept drifting" challenges. As a result, the stream of graphs is converted into feature vectors without additional parsing such that we can directly adopt a stochastic learning strategy to train a graph classifier online. A chunk level weighting mechanism is adopted to build an ensemble for classifying graph stream with concept drifts. Experiments and comparisons on real-world and synthetic graph streams demonstrate that the proposed method outperforms the state-of-the-art methods in both classification accuracy and learning efficiency.

REFERENCES

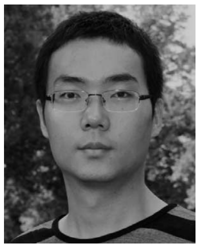
[1] H. Kashima, K. Tsuda, and A. Inokuchi, "Marginalized kernels between labeled graphs," in *Proc. ICML*, Washington, DC, USA, 2003, pp. 321–328.
[2] K. M. Borgwardt and H.-P. Kriegel, "Shortest-path kernels on graphs," in *Proc. ICDM*, Houston, TX, USA, 2005, pp. 74–81.
[3] P. Mahé and J.-P. Vert, "Graph kernels based on tree patterns for molecules," *Mach. Learn.*, vol. 75, no. 1, pp. 3–35, Apr. 2009.
[4] N. Shervashidze and K. M. Borgwardt, "Fast subtree kernels on graphs," in *Proc. NIPS*, Vancouver, BC, Canada, 2009, pp. 1660–1668.
[5] L. Chen and C. Wang, "Continuous subgraph pattern search over certain and uncertain graph streams," *IEEE Trans. Knowl. Data Eng.*, vol. 22, no. 8, pp. 1093–1109, Aug. 2010.
[6] J. Feigenbaum, S. Kannan, A. McGregor, S. Suri, and J. Zhang, "Graph distances in the data-stream model," *SIAM J. Comput.*, vol. 38, no. 5, pp. 1709–1727, 2008.
[7] C. C. Aggarwal, Y. Zhao, and P. S. Yu, "On clustering graph streams," in *Proc. SDM*, Columbus, OH, USA, 2010, pp. 478–489.
[8] C. C. Aggarwal, "On classification of graph streams," in *Proc. SDM*, Mesa, AZ, USA, 2011, pp. 652–663.

[9] B. Li, X. Zhu, L. Chi, and C. Zhang, "Nested subtree hash kernels for large-scale graph classification over streams," in *Proc. ICDM*, Brussels, Belgium, 2012, pp. 399–408.
[10] L. Chi, B. Li, and X. Zhu, "Fast graph stream classification using discriminative clique hashing," in *Proc. PAKDD*, Gold Coast, QLD, Australia, 2013, pp. 225–236.
[11] M. A. Shrivastava and B. Pant, "Opinion extraction and classification of real time Facebook status," *Glob. J. Comput. Sci. Technol.*, vol. 12, no. 8, pp. 35–40, Apr. 2012.
[12] S. Pan, X. Zhu, C. Zhang, and P. S. Yu, "Graph stream classification using labeled and unlabeled graphs," in *Proc. ICDE*, Brisbane, QLD, Australia, 2013, pp. 398–409.
[13] S. V. N. Vishwanathan, N. N. Schraudolph, R. Kondor, and K. M. Borgwardt, "Graph kernels," *J. Mach. Learn. Res.*, vol. 11, no. 2, pp. 1201–1242, 2010.
[14] X. Yan and J. Han, "gSpan: Graph-based substructure pattern mining," in *Proc. ICDM*, Maebashi, Japan, 2002, pp. 721–724.
[15] M. M. Gaber, A. B. Zaslavsky, and S. Krishnaswamy, "Mining data streams: A review," *ACM SIGMOD Rec.*, vol. 34, no. 2, pp. 18–26, Jun. 2005.
[16] J. Gama, *Knowledge Discovery From Data Streams*. Boca Raton, FL, USA: CRC Press, 2010.
[17] G. Ditzler, M. Roveri, C. Alippi, and R. Polikar, "Learning in nonstationary environments: A survey," *IEEE Comput. Intell. Mag.*, vol. 10, no. 4, pp. 12–25, Nov. 2015.
[18] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom, "Models and issues in data stream systems," in *Proc. PODS*, Madison, WI, USA, 2002, pp. 1–16.
[19] S. Muthukrishnan, "Data streams: Algorithms and applications," in *Foundations and Trends in Theoretical Computer Science*. Hanover, MA, USA: Now, Aug. 2005.
[20] P. Domingos and G. Hulten, "Mining high-speed data streams," in *Proc. KDD*, Boston, MA, USA, 2000, pp. 71–80.
[21] W. N. Street and Y. Kim, "A streaming ensemble algorithm (SEA) for large-scale classification," in *Proc. KDD*, San Francisco, CA, USA, 2001, pp. 377–382.
[22] H. Wang, W. Fan, P. S. Yu, and J. Han, "Mining concept-drifting data streams using ensemble classifiers," in *Proc. KDD*, Washington, DC, USA, 2003, pp. 226–235.
[23] A. Bifet and R. Gavaldà, "Adaptive XML tree classification on evolving data streams," in *Proc. Joint Eur. Conf. Mach. Learn. Knowl. Disc. Databases*, Bled, Slovenia, 2009, pp. 147–162.
[24] D. Brzezinski and M. Piernik, "Structural XML classification in concept drifting data streams," *New Gener. Comput.*, vol. 33, no. 4, pp. 345–366, Jul. 2015.
[25] S. Pan, J. Wu, X. Zhu, and C. Zhang, "Graph ensemble boosting for imbalanced noisy graph stream classification," *IEEE Trans. Cybern.*, vol. 45, no. 5, pp. 954–968, May 2015.
[26] L. Bottou, *Stochastic Learning* (Lecture Notes in Computer Science). 2004, pp. 146–168.
[27] Q. Shi *et al.*, "Hash kernels for structured data," *J. Mach. Learn. Res.*, vol. 10, no. 11, pp. 2615–2637, 2009.
[28] S. Gollapudi and R. Panigrahy, "The power of two min-hashes for similarity search among hierarchical data objects," in *Proc. 27th ACM SIGMOD SIGACT SIGART Symp. Principles Database Syst.*, Vancouver, BC, Canada, 2008, pp. 211–220.
[29] S. Tatikonda and S. Parthasarathy, "Hashing tree-structured data: Methods and applications," in *Proc. IEEE 26th Int. Conf. Data Eng. (ICDE)*, Long Beach, CA, USA, 2010, pp. 429–440.
[30] L. Chi, B. Li, and X. Zhu, "Context-preserving hashing for fast text classification," in *Proc. SIAM Int. Conf. Data Min.*, Philadelphia, PA, USA, 2014, pp. 100–108.
[31] X. Liu, Y. Mu, D. Zhang, B. Lang, and X. Li, "Large-scale unsupervised hashing with shared structure learning," *IEEE Trans. Cybern.*, vol. 45, no. 9, pp. 1811–1822, Sep. 2015.
[32] U. Feige, S. Goldwasser, L. Lovász, S. Safra, and M. Szegedy, "Approximating clique is almost NP-complete (preliminary version)," in *Proc. 32th FOCS*, 1991, pp. 2–12.
[33] C. Bron and J. Kerbosch, "Algorithm 457: Finding all cliques of an undirected graph," *Commun. ACM*, vol. 16, no. 9, pp. 575–577, Sep. 1973.
[34] H. A. Soufiani and E. Airoldi, "Graphlet decomposition of a weighted network," in *Proc. 15th Int. Conf. Artif. Intell. Stat. (AISTATS)*, 2012, pp. 54–63.
[35] A. Tsymbal, "The problem of concept drift: Definitions and related work," Dept. Comput. Sci., Trinity College, Univ. at Dublin, Dublin, Ireland, Tech. Rep. TCD-CS-2004-15, 2004.
[36] D. Chakrabarti, Y. Zhan, and C. Faloutsos, "R-MAT: A recursive model for graph mining," in *Proc. SDM*, Toronto, ON, Canada, 2004, pp. 442–446.
[37] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *J. Mach. Learn. Res.*, vol. 7, no. 1, pp. 1–30, 2006.
[38] B. Calvo and G. S. Rodrigo, "scmamp: Statistical comparison of multiple algorithms in multiple problems," *R J.*, vols. 1–8, pp. 108–131, Aug. 2016.

Lianhua Chi received the dual Ph.D. degrees in computer science from the University of Technology Sydney, Broadway, NSW, Australia, and the Huazhong University of Science and Technology, Wuhan, China, in 2015.

She is currently a Post-Doctoral Researcher with IBM Research, Melbourne, VIC, Australia. Her current research interests include time series data mining, graph stream data mining, data correlation analysis, and big data hashing.

Dr. Chi was a recipient of the Best Paper Award in PAKDD in 2013.



Bin Li received the Ph.D. degree in computer science from Fudan University, Shanghai, China, in 2009.

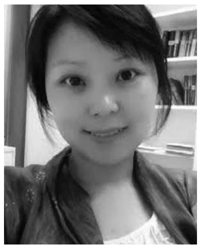
He was a Lecturer with the University of Technology Sydney, Broadway, NSW, Australia, and a Research Fellow with Institut TELECOM SudParis, Évry, France. He is currently a Senior Research Scientist with Data61 (formerly NICTA), CSIRO, Eveleigh, NSW, Australia. His current research interests include stochastic processes and randomized algorithms in machine learning and their applications to recommender systems, and smartcity data analytics.

Xingquan Zhu (SM'12) received the Ph.D. degree in computer science from Fudan University, Shanghai, China.

He is an Associate Professor with the Department of Computer and Electrical Engineering and Computer Science, Florida Atlantic University, Boca Raton, FL, USA, and a Distinguished Visiting Professor (Eastern Scholar) with the Shanghai Institutions of Higher Learning, Shanghai. His current research interests include data mining, machine learning, and computational advertising.

Shirui Pan received the Ph.D. degree in computer science from the University of Technology Sydney (UTS), Broadway, NSW, Australia.

He is a Research Associate with the Centre for Artificial Intelligence, UTS. He has published over 30 research papers in top-tier journals and conferences, including the IEEE TRANSACTIONS ON NEURAL NETWORKS AND LEARNING SYSTEMS, the IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, the IEEE TRANSACTIONS ON CYBERNETICS, *Pattern Recognition*, IJCAI, ICDE, ICDM, SDM, CIKM, and PAKDD. His current research interests include data mining and machine learning.



Ling Chen received the Ph.D. degree from Nanyang Technological University, Singapore.

She was a Post-Doctoral Research Fellow with L3S Research Center, University of Hanover, Hanover, Germany. She is a Senior Lecturer with the Centre for Quantum Computation and Intelligent Systems, University of Technology Sydney, Broadway, NSW, Australia. Her current research interests include data mining and machine learning, social network analysis, and recommender systems.