Fast Graph Stream Classification Using Discriminative Clique Hashing

Lianhua Chi[1,2], Bin Li[1], and Xingquan Zhu[1]

[1] Centre for Quantum Computation & Intelligent Systems, FEIT, University of Technology, Sydney, NSW 2007, Australia
[2] School of Computer Science & Technology, Huazhong University of Science & Technology, Wuhan 430074, China
{lianhua.chi,bin.li-1,xingquan.zhu}@uts.edu.au

Abstract. As many data mining applications involve networked data with dynamically increasing volumes, graph stream classification has recently extracted significant research interest. The aim of graph stream classification is to learn a discriminative model from a stream of graphs represented by sets of edges on a complex network. In this paper, we propose a fast graph stream classification method using DIscriminative Clique Hashing (DICH). The main idea is to employ 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 employed to compress the original edge set of the graph stream and map the unlimitedly increasing clique-patterns onto a fixed-size feature space, respectively. The hashed cliques are used to update an "in-memory" fixed-size pattern-class table, which will be finally used to construct a rule-based classifier. DICH essentially speeds up the discriminative clique-pattern mining process and solves the unlimited clique-pattern expanding problem in graph stream mining. Experimental results on two real-world graph stream data sets demonstrate that DICH can clearly outperform the compared state-of-the-art method in both classification accuracy and training efficiency.

Keywords: Graph classification, graph streams, cliques, hashing.

1 Introduction

The emergence of complex networks has led to a surge of research in graph data mining [1]. Graph classification is an important graph data mining task that aims to learn a discriminative model from training examples to predict class labels of test examples, where both training and test examples are graphs. Many real-world applications involve graph-represented data, such as chemical compounds, XML documents, and program flows. The essential challenge for graph classification is to extract features from graphs and represent graph data in instance-feature format to support model training. A variety of studies on substructure extraction (e.g., walks [2], paths [3], and subtrees [4,5]) for describing graphs have been proposed in the past decade. However, most of them only consider the learning problem of graph classification in batch mode (all data are available for training), which limits their applicability to large-scale and stream scenarios.

Due to the streaming nature of many real-world complex networks, such as social networks and sensor networks, graph stream classification has recently attracted increasing research interest [6,7,8,9]. Graph stream classification is defined on a complex network which comprises a massive universe of nodes, where the stream of graphs are represented as sets of edges on the underlying network. For example, co-authorships of research works continuously form graphs on a coauthor network (e.g., DBLP), dynamic communities of interest continuously form graphs on a social network (e.g., Facebook), and traffic flows continuously form graphs on a transportation network. Graph stream classification on a complex network with massive nodes is challenging, because

– **Subgraph Feature Generation:** Graph stream is defined on a massive universe of nodes, enumerating subgraph-patterns from such a large node set as features is time consuming and memory intensive. We need fast and inexpensive feature generation method for graph stream classification.
– **Increasing Stream Volumes:** The volumes of graph data are continuously growing, so graph streams can usually be accessed only once. Graph stream classification must be able to tackle dynamically increasing graph volumes and generate discriminative model with high speed.
– **Changing Feature Distributions:** The marginal distributions of subgraph-patterns (features) may continuously change over the graph stream (i.e., the concept-drift problem [10]), a dynamic updating scheme is required to update the discriminative model.

Few studies have investigated the graph stream classification problem. To the best of our knowledge, only two works [9,11] may be applied to the considered problem. Both of them employ hashing techniques to sketch the graph stream for saving computational cost and controlling the size of the subgraph-pattern set. In [11], the authors proposed a hash kernel to project arbitrary graphs onto a compatible feature space for similarity computing, but it can only be applied to node-attributed graphs. Recently, Aggarwal [9] 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 [9] has exhibited promising performance on graph stream classification, there are two inherent 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 comprises massive transactions.

In this paper, we propose a fast graph stream classification method using DIscriminative Clique Hashing (DICH) to address the aforementioned challenges. The main idea is to decompose a compressed graph into a number of cliques (fully connected subgraphs) to sequentially extract clique-patterns over the graph stream as features. Two random hashing schemes are employed to compress the original edge set of the graph stream and map the unlimitedly increasing clique-patterns onto a fixed-size feature space, respectively. The hashed cliques are then used to update an "in-memory" fixed-size pattern-class table, which will be finally used to generate a rule-based classifier. *Since DICH adopts connected subgraphs as features and needs no additional frequent pattern mining procedure, it can achieve very fast training speed for graph stream classification.* The experimental results on two real-world graph stream data sets clearly show that DICH outperforms the compared state-of-the-art [9] in both classification accuracy and training efficiency.

The remainder of this paper is organized as follows: Section 2 introduces the related work. The proposed framework for graph stream classification will be described in Section 3 and the detailed method of DICH will be presented in Section 4. We empirically evaluate our method to show its effectiveness and efficiency in Section 5 and conclude the paper in Section 6.

2 Related Work

The considered problem is closely related to graph classification. Most existing works focus on designing effective yet efficient kernels for measuring graph similarity. A large number of graph kernels have been proposed in the last decade, most of which are based on the similar idea of extracting substructures from graphs to compare their co-occurrences. Typical substructures for describing graphs include walks [2,12], paths [3], subtrees [4,5,13], and subgraphs (usually based on a frequent subgraph mining technique, e.g., [14]). In this paper, we extract cliques as features for describing a graph.

Our problem is also related to data stream mining. Mining high-speed data streams was first studied in [15] and the idea of using ensemble learning for data stream classification was proposed soon after [16]. A classical ensemble learning framework for addressing the concept-drift problem in data stream mining was proposed in [10]. In our graph stream classification problem, we employ a rule-based classification approach rather than the ensemble learning framework for faster processing speed and easier model updating.

The most relevant work to this paper is [9], which also considers graph stream classification on a complex network. It employs 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. The second min-hash scheme is used to dynamically update a number of hash-codes (i.e., corresponding to random sorting samples), which is able to summarize the frequent patterns of co-occurrence edges in the graph stream observed thus far. Finally, a simple heuristic is used to select a set of most discriminative frequent patterns to build a rule-based classifier. In this paper, we propose a clique-based hashing scheme for solving the same problem with a better performance in both classification accuracy and training efficiency as well as avoiding the the limitations of [9] discussed in Section 1.

3 Framework

We first introduce the problem setting for graph stream classification. Suppose there is a complex network which comprises a massive universe of nodes. The edges connecting these nodes are denoted by the edge set $\mathcal{E}$. The stream of graphs $\{G_1, G_2, \ldots, G_i, \ldots\}$ are presented continuously as subsets of $\mathcal{E}$, where the subscript i denotes the receiving order in the graph stream. In particular, the edge set of G_i are denoted by $\{E_1, \ldots, E_e\} \subset \mathcal{E}$, where e denotes the number of edges in G_i. Each graph G_i has a class label $L_i \in \{1, \ldots, M\}$. We assume G_i is received in the form $\langle i, E_1, \ldots, E_e, L_i \rangle$. In this paper, we assume that each edge has a default weight 1 for simplicity. The underlying graph stream can only be accessed once and our **goal** is to learn a discriminative model from $\{G_1, G_2, \ldots, G_i, \ldots\}$ at a high efficiency to accurately predict the class label of a test graph G_{test} in the future graph stream.

We next give an overview of DICH for graph stream classification. The corresponding framework, illustrated in Figure 1, comprises three modules. The graphs in the stream are received and processed one by one. The first module is for clique detection from each graph in the stream. The incoming edges of G_i are first randomly hashed to a compressed edge set and then we adopt a fast algorithm to decompose the compressed graph into a number of cliques (fully connected subgraphs) as the features of G_i. Since the number of clique-patterns will unlimitedly increase as new graphs are fed in, the underlying feature space will keep expanding accordingly. Thus, in the second module, a clique hashing scheme is performed to map the unlimitedly emerged clique-patterns onto a fixed-size clique-pattern set. In the last module, an "in-memory" fixed-size pattern-class table is updated using the clique-pattern and class label information of G_i; and a rule-based classifier is constructed based on the pattern-class table by identifying frequent and discriminative clique-patterns associated to each class. To test a graph G_{test} in the future graph stream, G_{test} is processed in the first two modules and the obtained hashed clique-patterns are input to the rule-based classifier for class label prediction. The detailed approaches to the three modules are described in the following section.

4 Graph Stream Classification

4.1 Graph Clique Detection

As shown in Figure 1, instead of relying on expensive frequent subgraph mining to discover graph features, we propose to use frequent and discriminative clique-patterns for clique-based classifier construction. So our first step is to detect all the cliques from each graph in the graph stream. Since the edge set of the graph stream on the complex network can be extremely large, it is necessary to sketch the graph stream as a preprocessing step. In particular, we use a random hash function to map the original edge set $\mathcal{E}$ onto a significantly compressed edge set $\bar{\mathcal{E}}$ of size N. That is to say, the edges in the compressed graph of G_i will be indexed by $\{1, \ldots, N\}$. If multiple edges in G_i are hashed onto the same index, the weight

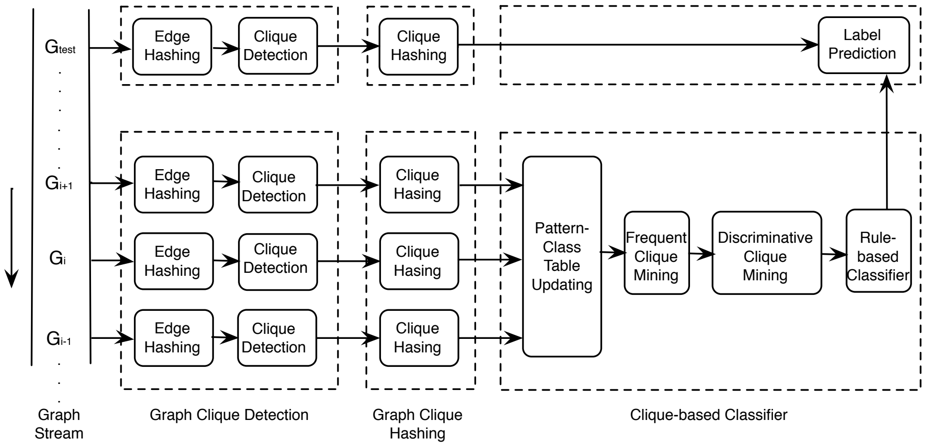


Fig. 1. The framework of DICH for graph stream classification

Algorithm 1. Clique Detection

```
Clique-Detect(): Detect the clique set $C_i$ from graph $G_i$.
begin
  $\bar{G}_i := \text{Edge-Hash}(G_i, N)$;
  $C_i := \emptyset$;
  for $t := \max(\bar{G}_i)$ to $\min(\bar{G}_i)$ step 1 do
    $\bar{G}_i^{(t)} := \mathbf{1}(\bar{G}_i \geq t)$;
    $C_i^{(t)} := \text{Bron-Kerbosch}(\bar{G}_i^{(t)})$;
    $C_i := C_i \bigcup C_i^{(t)}$;
  od
end
```

of the compressed edge is set to the number of the edges that get the same index. After hashing all the edges in G_i, we obtain a compressed graph $\bar{G}_i$ for clique detection. We define this edge hashing scheme using $\bar{G}_i := \text{Edge-Hash}(G_i, N)$. The leftmost two columns in Figure 2 illustrates this procedure.

Next we will employ a fast algorithm to detect cliques in each compressed graph. We adapt the graphlet basis estimation algorithm used in [17] to this end. The first step for clique detection is to threshold the compressed graph $\bar{G}_i$ at a number of weight levels in descending order, say $t = \{\max(\bar{G}_i), \ldots, \min(\bar{G}_i)\}$, where $\max(\bar{G}_i)$ and $\min(\bar{G}_i)$ denote the largest and the smallest edge weights in $\bar{G}_i$, respectively. We define this operation using $\bar{G}_i^{(t)} := \mathbf{1}(\bar{G}_i \geq t)$, where $\mathbf{1}(\cdot)$ denotes the indicator function and the resulting graph is denoted by $\bar{G}_i^{(t)}$. We use the Bron-Kerbosch algorithm [18] to identify all the cliques from $\bar{G}_i^{(t)}$ at each threshold. The union set of the cliques found in $\{\bar{G}_i^{(t)}\}_{t=\min(\bar{G}_i)}^{\max(\bar{G}_i)}$ is represented as the clique set for G_i. This procedure is detailed in Algorithm 1.

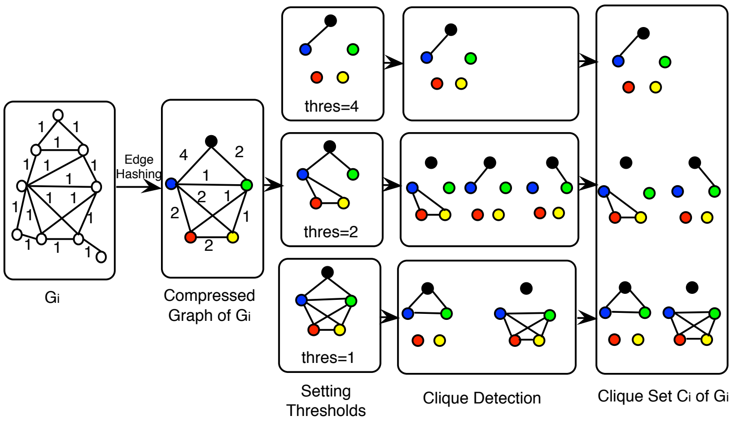


Fig. 2. Clique detection in a compressed graph

An example of clique detection is illustrated in Figure 2. After edge hashing, we obtain the compressed graph of G_i (2nd column). Then, four weight thresholds $\{4, 3, 2, 1\}$ are set to generate three graphs $\{\bar{G}_i^{(1)}, \bar{G}_i^{(2)}, \bar{G}_i^{(4)}\}$ (3rd column). Note that $\bar{G}_i^{(3)}$ is an empty graph which is not shown. The Bron-Kerbosch algorithm is applied to the three graphs and detect a set of cliques from each graph (4th column). Finally, the cliques detected at all weight thresholds are merged to form the clique set C_i for G_i as its feature representation (5th column).

4.2 Graph Clique Hashing

The cliques extracted from each graph are used to represent its features. To learn a classifier from the graph stream, it is required to make the features of all graphs be in the same feature space. In other words, we should count the occurrences of the same set of clique-patterns in all graphs in the stream. Since the number of clique-patterns will increase as new graphs are continuously fed in, the induced feature space will keep expanding accordingly. To address this problem, we adopt a feature hashing scheme used in [11] to randomly map the unlimitedly emerged clique-patterns onto a fixed-size set. In particular, we use an "in-memory" $P \times M$ pattern-class table $\mathit{\Delta}$, which can be dynamically updated, to count clique-pattern and class label information from the graph stream. In $\mathit{\Delta}$, P rows correspond to the indices of hashed clique-patterns while M columns correspond to all the classes of the graphs.

Given G_i in the graph stream, we first use Algorithm 1 to collect the clique set C_i. Then, for each clique in C_i, say $C_{i,j}$, we apply a random hash function $\hbar(\cdot)$ to the string of ordered edges in $C_{i,j}$ to generate an index $H_{i,j} \in \{1, \ldots, P\}$. If a clique with class label L_i is hashed to an index $H_{i,j}$, we add 1 to the entry $\mathit{\Delta}[H_{i,j}, L_i]$, which means clique-pattern $C_{i,j}$ has a contribution to class L_i. This fixed-size pattern-class table is continuously updated as new cliques are detected over the graph stream. This procedure is detailed in Algorithm 2.

Algorithm 2. Clique Hashing

```
Clique-Hash(): Hash the cliques in $G_i$ and update the pattern-class table.
begin
  for $j := 1$ to size($C_i$) step 1 do
      $H_{i,j} := \hbar(C_{i,j})$;
      $\Delta[H_{i,j}, L_i] := \Delta[H_{i,j}, L_i] + 1$;
  od
end
```

4.3 Clique-Based Classifier

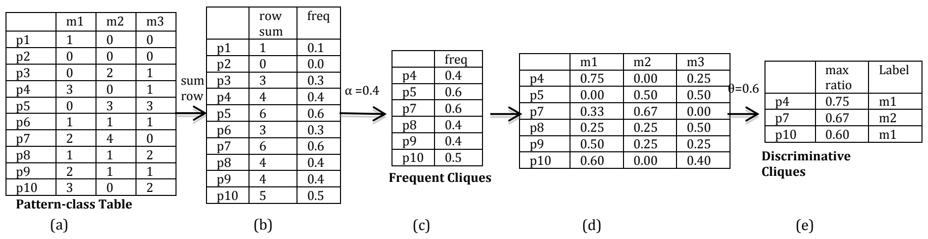
Given the "in-memory" pattern-class table Δ, we can construct a rule-based classifier by identifying frequent yet discriminative clique-patterns from Δ. To identify frequent clique-patterns, we first sum up the counts in each row of Δ and divide them by the number of graphs received thus far. The result for each row indicates the occurrence frequency of a set of cliques with the same hash value in the graph stream. Then we sort them in a descending order and set a threshold parameter α to select the clique-patterns whose frequencies $\geq \alpha$. These selected cliques are frequent clique-patterns which are also the candidates for the subsequent discriminative clique-pattern selection.

Next we can determine whether a frequent clique-pattern is also a discriminative one by comparing its occurrence ratios on the M classes (corresponding to the M columns in Δ). For a candidate clique-pattern, the ratio in column j represents the probability that the clique-pattern belongs to class j. A higher probability on a certain class indicates a better discriminative capability. Similarly, we can set a threshold parameter θ to select the clique-patterns whose maximum ratios $\geq \theta$. Figure 3 gives a toy example for selecting the frequent and discriminative clique-patterns from a pattern-class table Δ.

Finally, based on the selected clique-patterns, we can classify a test graph using majority voting based on the detected cliques in the test graph. In particular, given a test graph G_{test}, we detect its cliques C_{test} using Algorithm 1 and hash its cliques $\{H_{test,j}\}_{j=1}^{\text{size}(C_{test})}$ to index its clique-patterns. Each clique corresponding to a discriminative clique-pattern will contributes a class label $L_{test,j}$:= Find-Rule($\Delta, H_{test,j}$). The class label of the test graph L_{test} is determined by the majority of class labels $\{L_{test,j}\}_{j=1}^{\text{size}(C_{test})}$. This procedure is detailed in Algorithm 3.

5 Experimental Results

In this section, we will test the proposed DICH method for graph stream classification on two real-world data sets. In particular, we will evaluate the effectiveness and efficiency of DICH by comparing it with the 2-D hash compressed stream classifier [9], which is the only state-of-the-art method applicable to graph stream classification. We use the following data sets in our experiments.


sum row
α =0.4
θ=0.6
Pattern-class Table
Frequent Cliques
Discriminative Cliques
(a)
(b)
(c)
(d)
(e)

	m1	m2	m3
p1	1	0	0
p2	0	0	0
p3	0	2	1
p4	3	0	1
p5	0	3	3
p6	1	1	1
p7	2	4	0
p8	1	1	2
p9	2	1	1
p10	3	0	2

	row sum	freq
p1	1	0.1
p2	0	0.0
p3	3	0.3
p4	4	0.4
p5	6	0.6
p6	3	0.3
p7	6	0.6
p8	4	0.4
p9	4	0.4
p10	5	0.5

	freq
p4	0.4
p5	0.6
p7	0.6
p8	0.4
p9	0.4
p10	0.5

	m1	m2	m3
p4	0.75	0.00	0.25
p5	0.00	0.50	0.50
p7	0.33	0.67	0.00
p8	0.25	0.25	0.50
p9	0.50	0.25	0.25
p10	0.60	0.00	0.40

	max ratio	Label
p4	0.75	m1
p7	0.67	m2
p10	0.60	m1

Fig. 3. A toy example of frequent and discriminative clique-pattern mining. (a) A pattern-class table with 10 clique-patterns $\{p_1, \ldots, p_{10}\}$ and 3 classes $\{m_1, m_2, m_3\}$. (b) The sums of individual clique-patterns in rows and the corresponding occurrence frequencies (i.e., the sums divided by the number of graphs, say 10 here). (c) The selected frequent clique-patterns whose frequencies are larger than the frequent pattern threshold $\alpha = 0.4$. (d) The occurrence ratios of the selected clique-patterns on the M classes. (e) The selected discriminative clique-patterns whose maximum ratios are larger than the discriminative pattern threshold $\theta = 0.6$.

Algorithm 3. Graph Classification

```
Graph-Classify(): Predict the class label of a test graph $G_{test}$ in the stream.
begin
  $C_{test}$ = Clique-Detect($G_{test}$);
  for $j := 1$ to size($C_{test}$) step 1 do
    $H_{test,j} := \hbar(C_{test,j})$;
    $L_{test,j} :=$ Find-Rule($\Delta, H_{test,j}$);
  od
  $L_{test} :=$ Majority-Vote($\{L_{test,j}\}_{j=1}^{\text{size}(C_{test})}$);
end
```

- DBLP Data Set[1]: In this data set, authors are nodes and co-authorship forms edges, and a graph is constituted by the co-authors of a paper. There are three classes in the data set: 1) Database related conferences, 2) Data mining related conferences, and 3) All remaining conferences. Our goal is to classify a test paper into one of three classes. The final data set contains over 5×10^5 authors, 9.75×10^5 edges, and 3.55×10^5 different graphs as the training data. We divide the data set into five splits and choose four splits as the training data and the remaining split as the test data.
- IBM Sensor Data Set[2]: This data set records the information from local traffic constituted by each graph on a sensor network. The IP-addresses are 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

[1] http://www.charuaggarwal.net/dblpcl/
[2] http://www.charuaggarwal.net/sens1/gstream.txt

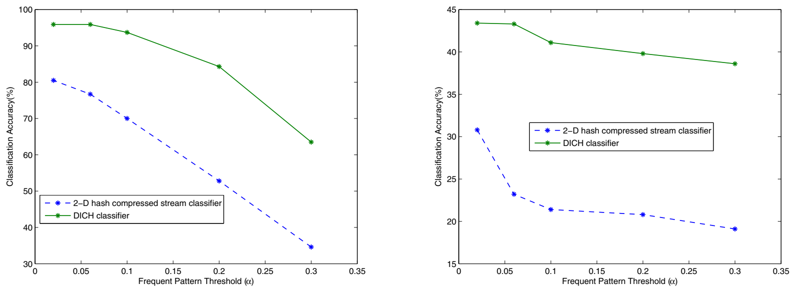


Fig. 4. Effectiveness evaluation w.r.t. α on the DBLP data set (left) and the IBM sensor data set (right), $N = 5000$ and $\theta = 0.4$

of intrusion types. Because the number of classes is extremely large (> 300), we expect the overall accuracy to be relatively low. The data set contains more than 1.57×10^6 edges. We choose 90% of the data as the training data and the remaining 10% as the test data.

5.1 Effectiveness Evaluation

In this experiment, we evaluate the effectiveness of DICH by comparing it with the 2-D hash compressed stream classifier proposed in [9]. We will investigate the classification performance and sensitivity of the two methods by varying 1) the frequent pattern threshold α, 2) the discriminative pattern threshold θ, and 3) the size of the compressed edge set N.

First, we adjust the frequent pattern threshold[3] α for performance evaluation and fix the other two parameters by setting $N = 5000$ and $\theta = 0.4$. Figure 4 plots the classification accuracy curves (y-axis) w.r.t. α (x-axis) on the two data sets. We can see that the classification performance of DICH is much higher than the 2-D hash compressed stream classifier on both data sets and in all values of α. The performance of both classifiers trends to decline as α becomes larger since more graph features will be eliminated and such information loss will affect classification performance. By comparing the curve slopes of two classifiers, the 2-D hash compressed stream classifier is more sensitive to α. In the case of $\alpha = 0.3$, the classification accuracy of the 2-D hash compressed stream classifier is much lower. From this experiment, we can validate the effectiveness of DICH, which can clearly outperform the 2-D hash compressed stream classifier and is more insensitive to the frequent pattern threshold.

Second, we adjust the discriminative pattern threshold θ for performance evaluation and fix the other two parameters by setting $N = 5000$ and $\alpha = 0.05$. Figure 5 plots the classification accuracy curves (y-axis) w.r.t. θ (x-axis) on the two data sets. On the DBLP data set, DICH was significantly superior to the 2-D hash compressed classifier in classification accuracy. The classification performance of both methods was insensitive to θ, which may be due to the fact

[3] In [9], the frequent pattern threshold α is used to screen out subgraph patterns.

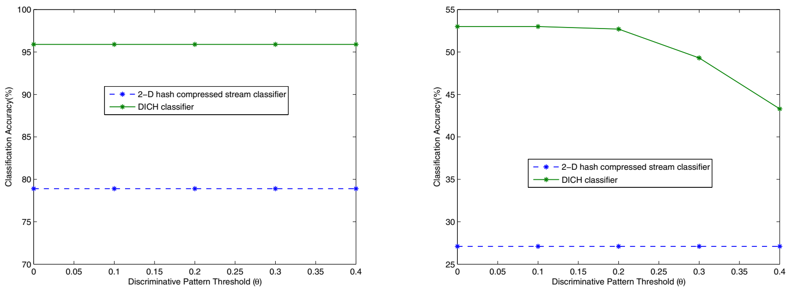


Fig. 5. Effectiveness evaluation w.r.t. θ on the DBLP data set (left) and the IBM sensor data set (right), $N = 5000$ and $\alpha = 0.05$

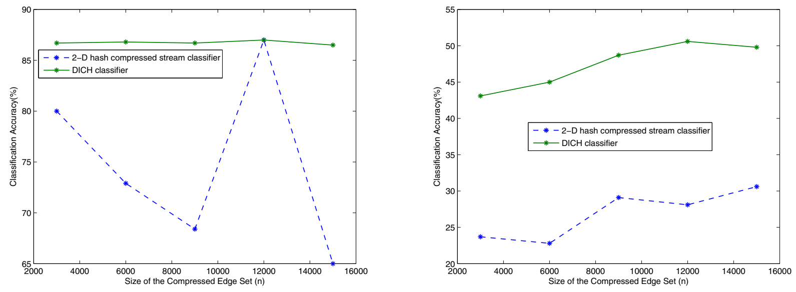


Fig. 6. Effectiveness evaluation w.r.t. the edges compression size N on the DBLP data set (left) and the IBM sensor data set (right), $\alpha = 0.06$ and $\theta = 0.3$

that the two classes (Database related conferences and Data mining related conferences) in DBLP data are extremely rare, the identified frequent patterns have already had relatively high discriminative capability. On the IBM sensor data set, although DICH is somewhat more sensitive to θ than the 2-D hash compressed classifier, it has much higher classification accuracy in all cases. This experiment further demonstrates that DICH has higher effectiveness than the 2-D hash compressed classifier in terms of the discriminative pattern threshold.

Third, we adjust the size of the compressed edge set N for performance evaluation and fix the other two parameters by setting $\alpha = 0.06$ and $\theta = 0.3$. Intuitively, the classification performance of both classifiers will increase at the expense of more space. Figure 6 plots the classification accuracy curves (y-axis) w.r.t. N (x-axis) on the two data sets. We can see that the 2-D hash compressed classifier is very sensitive to N, especially on the DBLP data set; while DICH is more steady on the DBLP data set but no clear performance improvement can be observed as N becomes larger. On the IBM sensor data set, the performance of both classifiers is improved as N becomes larger. Again, DICH outperforms the 2-D hash compressed classifier in all cases.

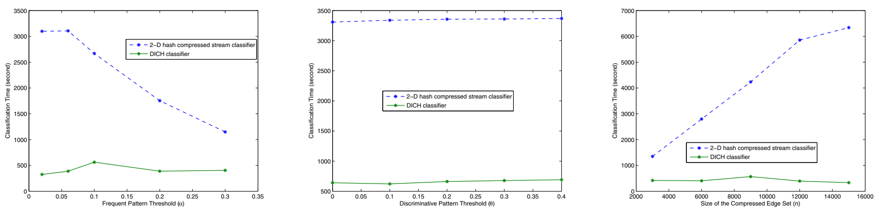


Fig. 7. Efficiency evaluation (1) w.r.t. α, $N = 5000$ and $\theta = 0.4$ (left); (2) w.r.t. θ, $N = 5000$ and $\alpha = 0.05$ (middle); and (3) w.r.t. N, $\alpha = 0.06$, $\theta = 0.3$ (right), on the DBLP data set

5.2 Efficiency Evaluation

In this experiment, we evaluate the efficiency of the two compared methods on the DBLP data set by adjusting the frequent pattern threshold α, the discriminative pattern threshold θ, and the size of the compressed edge set N. The settings of these parameters are the same as those in the above effectiveness evaluation. All the experiments are conducted on a Linux Cluster which comprises 24 nodes with 3.33GHz Intel Xeon CPU (64bit). Both DICH and the 2-D hash compressed stream classifier are implemented using R studio.

Figure 7 plots the training time curves of the two compared methods w.r.t. α, θ, and N on the DBLP data set. We can see that the training time of DICH is significantly less than the compressed hash-based classifier in all cases. The computational cost of the 2-D hash compressed classifier is much higher because it requires an additional frequent pattern mining procedure to perform on the edge co-occurrence table which comprises massive transactions. In contrast, DICH employs a fast clique detection algorithm, which can directly find cliques (connected subgraphs) from the graph stream as features for classifier construction, such that no additional frequent pattern mining procedure is required to find connected subgraph patterns. This experiment shows that DICH clearly outperforms the 2-D hash compressed classifier in not only classification accuracy but also training efficiency.

6 Conclusion

In this paper, we propose a fast graph stream classification method using DIscriminative Clique Hashing (DICH). The main idea is to employ 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 employed to speed up the discriminative clique-pattern mining process and address the unlimitedly clique-pattern expanding problem. The hashed cliques are used to update an "in-memory" fixed-size pattern-class table, which is finally used to construct a rule-based classifier. We test DICH on two real-world graph stream data sets. Because DICH directly extracts cliques (connected subgraphs) from the graph stream as features for classifier training, rather than mining unconnected co-occurrence edge sets as that in the compared state-of-the-art method [9], DICH can significantly outperform [9] in both classification accuracy and learning efficiency.

Acknowledgements. This work was supported in part by Australian Research Council (ARC) Discovery Project DP1093762, Australian Research Council (ARC) Future Fellowship FT100100971, and a UTS Early Career Researcher Grant.

References

1. Aggarwal, C.C., Wang, H.: Managing and Mining Graph Data. Springer, New York (2010)
2. Kashima, H., Tsuda, K., Inokuchi, A.: Marginalized kernels between labeled graphs. In: ICML, pp. 321–328 (2003)
3. Borgwardt, K.M., Kriegel, H.P.: Shortest-path kernels on graphs. In: ICDM, pp. 74–81 (2005)
4. Mahé, P., Vert, J.P.: Graph kernels based on tree patterns for molecules. Machine Learning 75(1), 3–35 (2009)
5. Shervashidze, N., Borgwardt, K.: Fast subtree kernels on graphs. In: NIPS, pp. 1660–1668 (2009)
6. Feigenbaum, J., Kannan, S., McGregor, A., Suri, S., Zhang, J.: Graph distances in the data-stream model. SIAM Journal on Computing 38(5), 1709–1727 (2008)
7. Aggarwal, C.C., Zhao, Y., Yu, P.S.: On clustering graph streams. In: SDM, pp. 478–489 (2010)
8. Aggarwal, C.C., Li, Y., Yu, P.S., Jin, R.: On dense pattern mining in graph streams. In: PVLDB, pp. 975–984 (2010)
9. Aggarwal, C.C.: On classification of graph streams. In: SDM, pp. 652–663 (2011)
10. Wang, H., Fan, W., Yu, P.S., Han, J.: Mining concept-drifting data streams using ensemble classifiers. In: KDD, pp. 226–235 (2003)
11. Li, B., Zhu, X., Chi, L., Zhang, C.: Nested subtree hash kernels for large-scale graph classification over streams. In: ICDM, pp. 399–408 (2012)
12. Vishwanathan, S., Schraudolph, N.N., Kondor, R., Borgwardt, K.M.: Graph kernels. Journal of Machine Learning Research 11, 1201–1242 (2010)
13. Hido, S., Kashima, H.: A linear-time graph kernel. In: ICDM, pp. 179–188 (2009)
14. Yan, X., Han, J.: gSpan: Graph-based substructure pattern mining. In: ICDM, pp. 721–724 (2002)
15. Domingos, P., Hulten, G.: Mining high speed data streams. In: KDD, pp. 71–80 (2000)
16. Street, W.N., Kim, Y.: A streaming ensemble algorithm (SEA) for large-scale classification. In: KDD, pp. 377–382 (2001)
17. Soufiani, H.A., Airoldi, E.: Graphlet decomposition of a weighted network. Journal of Machine Learning Research – Proceedings Track 22, 54–63 (2012)
18. Bron, C., Kerbosch, J.: Algorithm 457: Finding all cliques of an undirected graph. Communications of the ACM 16(9), 575–577 (1973)