GSCS – Graph Stream Classification with Side Information

Amit Mandal[1], Mehedi Hasan[1], Anna Fariha[1], and Chowdhury Farhan Ahmed[2]

[1] Department of Computer Science and Engineering, University of Dhaka, Bangladesh
[2] ICube Laboratory, University of Strasbourg, France
amitducse17@gmail.com, mha_bd@yahoo.com, anna@cse.univdhaka.edu, cfahmed@unistra.fr

Abstract. With the popularity of applications like Internet, sensor network and social network, which generate graph data in stream form, graph stream classification has become an important problem. Many applications are generating side information associated with graph stream, such as terms and keywords in authorship graph of research papers or IP addresses and time spent on browsing in web click graph of Internet users. Although side information associated with each graph object contains semantically relevant information to the graph structure and can contribute much to improve the accuracy of graph classification process, none of the existing graph stream classification techniques consider side information. In this paper, we have proposed an approach, **G**raph **S**tream **C**lassification with **S**ide information (**GSCS**), which incorporates side information along with graph structure by increasing the dimension of the feature space of the data for building a better graph stream classification model. Empirical analysis by experimentation on two real life data sets is provided to depict the advantage of incorporating side information in the graph stream classification process to outperform the state of the art approaches. It is also evident from the experimental results that *GSCS* is robust enough to be used in classifying graphs in form of stream.

Keywords: Graph Classification, Graph Streams.

1 Introduction

With the expansion of technologies that generate graph data in form of stream like Internet, social network, sensor network etc. into our everyday life, graph stream classification is gaining significant research interest in data mining and machine learning community. Graph stream classification is a two step process, where in the learning step, features are extracted from each training graph and a classification model is built, and in the classification step, class labels of each test graphs are predicted using that classification model.

R. Cheng et al. (Eds.): APWeb 2015, LNCS 9313, pp. 387–398, 2015.
DOI: 10.1007/978-3-319-25255-1_33

Graph is a very popular data structure, suitable for representing schemaless data like representing various activities among nodes in a network. In many graph streams, various side attributes are associated with each graph which may contain semantically relevant information to the graph linkage structure. These informations may contribute to build a more discriminative classification model. Let us look at some examples. *First*, each article residing in scientific repositories(e.g. Google Scholar [1], DBLP [2]) can be represented using a bidirected authorship graph where authors are nodes and their co-author relationships constitute bidirectional edges among the nodes. Along with each graph, various types of side attributes like terms and keywords of the paper can be associated. *Second*, communication via message passing among users in a social network in a small time window can form a directed graph where users are nodes and messages sent and received among users make directed edges among the corresponding nodes of users of those messages. Different types of side attributes like user locations, user profile informations, message types (e.g. personal message, group chat) and platforms (e.g. PC, mobile) can be associated with each graph in the stream. Hence, side information should be incorporated along with graph linkage structure in the graph classification process to improve classification accuracy.

Features that only occur with high frequency in objects with one particular class label, used to discriminate among objects with two or more different class labels, are known as discriminative features. The aim of any classification algorithm is to build a classification model using discriminative features, which help to make finer distinctions among objects with different class labels, thus improving classification accuracy. By considering side information along with the graph linkage structure, dimension of the feature representation of a graph object can be increased. This increased dimensionality with features related to the graph structure, can be of great use to find more discriminative features from graph objects, hence increasing classification accuracy.

One of the inherent challenges in graph stream classification is storing the received objects from the stream and extracting features from the enormous volume of data. For example, there are 4.2×10^9 IPv4 addresses. So there can be 4.2×10^9 distinct nodes and 9.2×10^{18} distinct edges in the web graph stream. Storing this huge amount of data is intractable. Traditional graph classification algorithms require multiple scans over the whole data, which is not possible in stream scenario. So a summary of the graph stream is saved for future mining. As information loss is occurring while saving a summary of the graph stream, the results will be approximate rather than exact.

Various graph mining approaches have been defined in recent times to solve different tasks like classification [3], correlation mining [4], recommendation on social networks [5] etc. Though there are some approaches for graph stream classification [3][6][7][8], to the best of our knowledge none of them considers side information in the mining process. Aggarwal et al. [3] proposed a probabilistic discriminative subgraph mining approach, where the received graphs are at first compressed and saved into a summary table using two random hashing schemes

and then frequent and discriminative subgraphs are mined from that table to build a rule-based classifier. A clique based approach was proposed by Chi et al. [6] where the graphs are at first compressed into a fixed size node space and then discriminative clique patterns are extracted from each compressed graph to build a rule-based classifier. Discriminative hash kernel approach was proposed by Li et al. [7] to classify graph streams. Guo et al. [8] proposed an approach for graph stream classification which uses a combination of hashing and factorization of graphs. Yu et al. [9] considered side information in the graph stream clustering process and their experimental results showed the benefit of considering side information in the clustering process. One major drawback of the existing graph stream classification techniques is that, they do not consider side information in the classification process and the side information can help to improve classification accuracy.

Inspired by this drawback of the current graph stream classification techniques, in this paper we are proposing an approach $GSCS$ for graph stream classification which considers side information. In $GSCS$, we have mostly followed the discriminative clique based approach proposed by Chi et al. [6] for graph structure mining. The volume of side information in the stream scenario can also be potentially infinite. So a hash based technique is used to tackle this storage problem and extract discriminative features from side information. Finally a majority voting classifier, inspired from Chi et al. [6] approach, is designed for classifying the future stream.

The rest of the paper is organized as follows: Section 2 explains our proposed approach in detail with necessary examples and algorithms. Section 3 focuses on the performance analysis of our proposed technique and shows the benefit of considering side information in the classification process. Finally we bring the paper to a close in Section 4.

2 Our Proposed Approach – GSCS

First, we introduce the problem formulation. Assume we have a stream of graphs GS denoted as $\{G_1, G_2, \ldots, G_n, \ldots\}$. E represents the set of all distinct edges, $\{(X_1, Y_1), (X_2, Y_2), \ldots, (X_n, Y_n), \ldots\}$ in the graph stream. (X_i, Y_i) represents an edge between the two nodes X_i and Y_i. Each graph G_i is drawn on the subset of massive node set V and contains a subset of edges from set E. GS also has d different types of side information associated with it, denoted by $\tau = \{T_1, T_2, \ldots, T_d\}$. Each type of side information T_l, where $l = 1, 2, \ldots, d$ has multiple values $S_l = \{S_{l1}, S_{l2}, \ldots, S_{ln}, \ldots\}$. Our aim is to incorporate side information in the graph stream classification process in order to improve the classification accuracy over the existing graph stream classification algorithms.

Now, we give an overview of our GSCS approach. GSCS is composed of four modules, showed in Figure 1. Each graph object consisting of graph structure and side information, is received from the stream and processed according to $GSCS$ approach. The *first* module hashes a graph's edges into a fixed-size edge set and then maximal cliques are mined from the compressed graph. In this

module, we have modified the approach in $DICH$ [6] by using a more efficient algorithm, for maximal clique detection for better performance. *Second* module is followed from the approach stated in $DICH$ [6], for managing the the exponential number of cliques into a fixed-size feature space. The *third* module is used for side information mining and its design is inspired from $DICH$ [6]. Finally, we have designed the *fourth* module to incorporate side information in the classification process.

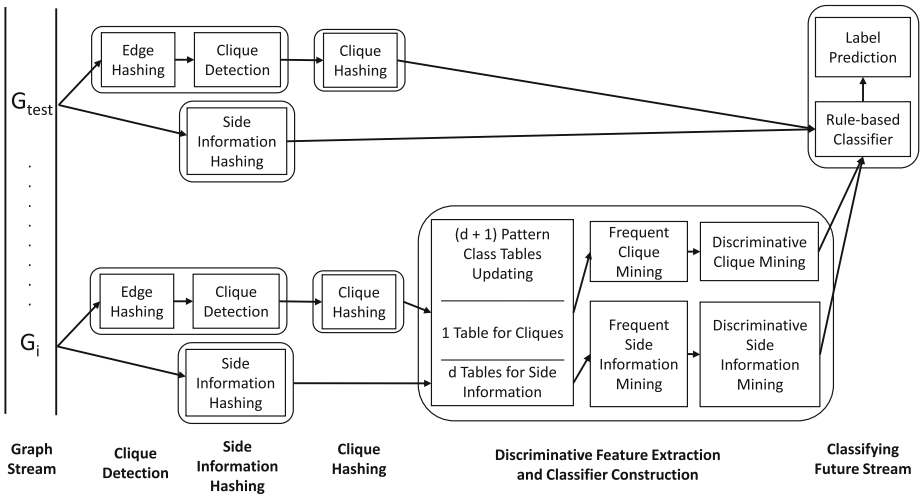


Fig. 1. $GSCS$ approach for graph stream classification with side information

As in stream scenario, due to the high incoming rate of enormous amount of data, a summary of stream needs to be saved for future mining. An in-memory data structure, enabling high speed access-update operations, is used for storing the summary. $DICH$ [6] used one in-memory table, where $GSCS$ will be using $(d+1)$ in-memory tables. One table for the summary of the graph linkage structure, the other d tables for the summary of the d types of side information. These modules are briefly described in Section 2.1-2.4.

2.1 Clique Detection

In this module, the original edge set of each received graph G_i is hashed into a compressed edge set, and maximal cliques are mined from the compressed edge set to be used as feature representation of the linkage structure of graph. In graph stream mining, massive universe of nodes and continuous arrival of data make storing the original stream intractable. So a summary of the stream needs to be saved for mining purpose. Hence, we hash the original edge set of graph G_i to a compressed edge set. Edge compression involves hashing the two node labels

of an edge into a fixed-size node space $\{1, \ldots, N\}$, and then considering the two hashed node label values as a compressed edge. Each time an edge is mapped into a compressed edge, the weight of the compressed edge is increased by 1. If multiple edges of G_i are mapped into the same index due to hash collision, the weight of that compressed edge is set to the number of edges that result into the same index after applying hash function.

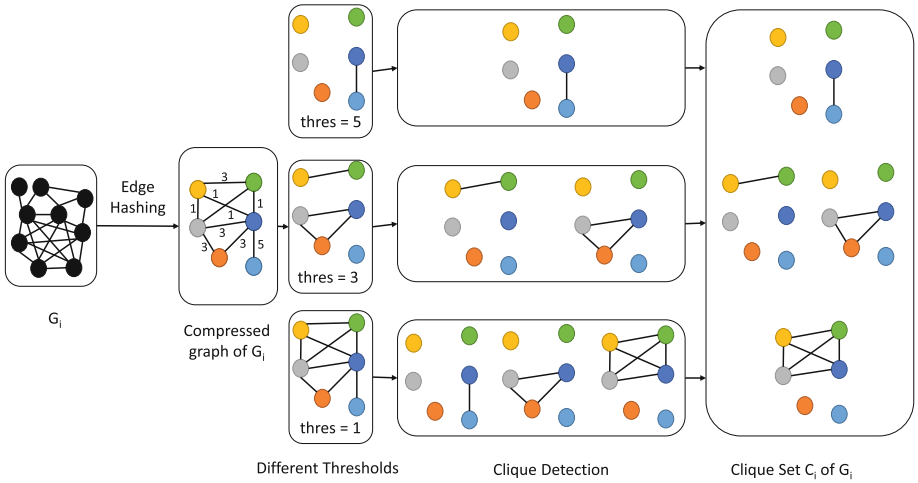


Fig. 2. Clique detection in a compressed graph

Compressed representation of the original graph is significantly smaller and it is now feasible to run clique detection algorithm. Different weights of compressed edges are taken into account and maximal cliques are mined at different edge weight threshold. Graphlet basis estimation algorithm [10] is adapted for this purpose. Though *DICH* [6] used the naive BronKerbosch algorithm [11], for improved performance, *GSCS* uses BronKerbosch with pivoting technique [12].

Let $\overline{G_i}$ be the compressed graph of G_i. $\overline{G_i^{(t)}}$ denotes the compressed graph which consists of only compressed edges with weight $\geq t$. Let $max(\overline{G_i})$ denotes the largest and $min(\overline{G_i})$ denotes the smallest edge weights in $\overline{G_i}$, respectively. For maximal clique detection, at first we apply threshold to the compressed $\overline{G_i}$ at different weight level t to get $\overline{G_i^{(t)}}$, where $t = \{max(\overline{G_i}), \ldots, min(\overline{G_i})\}$. BronKerbosch with pivoting algorithm [12] is used at every $\overline{G_i^{(t)}}$ to identify all the maximal cliques. All the cliques found in $\overline{G_i^{(t)}}$ at different weight levels, $\{\overline{G_i^{(t)}}\}_{t=min(\overline{G_i})}^{max(\overline{G_i})}$, are represented as the clique union set C_i of graph G_i. This procedure is shown in Figure 2 and detailed in Algorithm 1.

Algorithm 1. Clique Detection in Compressed Graph

Input : Graph G_i, size of the compressed node space N
Output: Clique set C_i of graph G_i

```
1 begin
2     G_i' ← Edge − Hash(G_i, N)
3     C_i ← φ
4     for t ← max(G_i') to min(G_i') do
5         G_i^(t) ← 1(G_i' ≥ t)
6         C_i^(t) ← Bron − Kerbosch − with − pivoting(G_i^(t))
7         C_i ← C_i ∪ C_i^(t)
8     end for
9 end
```

2.2 Clique Hashing

Each graph received from the stream is compressed and decomposed into maximal cliques and then frequent and discriminative cliques are identified to build a rule based classifier. As there can be $3^{n/3}$ maximal cliques [13] from a graph of n vertices, so there can be exponential number of cliques generated from the compressed node space. It makes the task of quantifying clique patterns for discriminative feature mining infeasible. To tackle this problem, each clique is hashed into a fixed-size feature space and the regarding information is updated in an in-memory pattern class table.

An in-memory pattern class table Δ_0, consisting of P rows and M columns,

Algorithm 2. Clique Hashing

Input : Clique set C_i of graph G_i
Output: Update graph linkage structure summary table Δ_0

```
1 begin
2     for j ← 1 to size(C_i) do
3         H_{i,j} = hash(C_{i,j})
4         Δ_0[H_{i,j}, L_i] = Δ_0[H_{i,j}, L_i] + 1
5     end for
6 end
```

where P is the size of the feature space and M is the number of distinct class labels in stream, is maintained for storing the graph structure summary. For each clique $C_{i,j}$ in clique set C_i of graph G_i, a hash value $H_{i,j} \in \{1, \ldots, P\}$ is generated. When a clique with class label L_i generates hash value $H_{i,j}$, the entry at $\Delta_0[H_{i,j}, L_i]$ is incremented by 1, so that the clique pattern $C_{i,j}$ contributes

to the class label L_i. This in-memory clique pattern class table Δ_0 is continuously updated by cliques as with the progression of the stream. The steps of this procedure is given in Algorithm 2.

2.3 Side Information Hashing

As the total number of aggregated side attributes associated with each received graph from the stream can be potentially infinite and it is not feasible to store them directly, so hashing is applied to save a summary of the side information. There are d pattern class tables for storing the summary of the d different types of side information. Each table consists of P rows and M columns where P is predefined size of the feature space and M is the number of distinct class labels in the stream. All types of side information values are hashed into the same fixed-size feature space $\{1, \ldots, P\}$ using the same hash function. Each side information value S_{lj}, where $j = 1, 2, \ldots, q$ and q is a finite number, of type T_l, where $l = 1, 2, \ldots, d$, associated with each graph G_i of the graph stream is hashed to generate a hash index $H_{i,lj} \in \{1, \ldots, P\}$. The i'th graph has class label L_i. Then the entry $\Delta_l[H_{i,lj}, L_i]$, in the side information pattern-class table Δ_l is incremented by 1. These steps are shown in Algorithm 3.

Algorithm 3. Side Information Hashing

Input : An array of sets of all types of side information SI_i of graph G_i
Output: Update the corresponding side information summary table Δ_l for each type l of side information, where $l = 1, \ldots, d$

```
1 begin
2     for l ← 1 to d do
3         for j ← 1 to size(SI_i[l]) do
4             H_{i,l_j} = hash(SI_i[l]_j)
5             Δ_l[H_{i,l_j}, L_i] = Δ_l[H_{i,l_j}, L_i] + 1
6         end for
7     end for
8 end
```

2.4 Discriminative Feature Extraction and Classifier Construction

GSCS uses $(d+1)$ in-memory pattern class tables where Δ_0 is for saving graph linkage structure summary and $\Delta_1, \ldots, \Delta_d$ are for storing a sketch of each types of side information in their corresponding table. α and θ, two parameters are used for discriminative feature extraction from these tables. α is the frequent pattern threshold used for selecting frequent features and θ is the discriminative pattern threshold used for selecting frequent yet discriminative patterns [6].

At first, for identifying frequent features, the values in each row of a table are summed up and divided by the maximum row sum in that table. The resulting

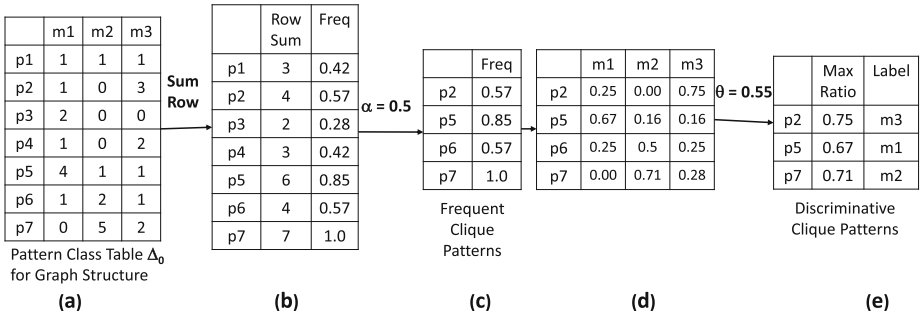


Fig. 3. A small example of frequent and discriminative feature mining. (a) Pattern class table Δ_0 which stores graph structure summary, composed of 7 clique patterns $\{p1, \ldots, p7\}$ (feature space size is 7) and 3 class labels $\{m1, m2, m3\}$. (b) The row sum of each clique pattern with corresponding occurrence frequency (i.e. each row sum in a table is divided by the maximum row sum of that table). (c) Frequent clique patterns with frequencies above the frequent pattern threshold α. (d) Each clique patterns occurrence ratio in each classes. (e) Finally, selected frequent and discriminative clique patterns where maximum ratio of each pattern is greater or equal to discriminative pattern threshold θ.

values indicate the occurrence frequency of a set of cliques in the graph stream. Frequent threshold parameter α is then used to filter out the infrequent patterns whose occurrence frequency are less than α. The remaining patterns are candidate for the discriminative feature selection.

To find the discriminative features, we need to start comparing the occurrence ratios of the features in M classes. For a candidate feature, its occurrence ratio in column i, where $i = 1, 2, \ldots, M$, indicates the probability that the feature belongs to class label i. Higher occurrence ratio of a feature on a certain class indicates a better discriminative capability. Discriminative threshold parameter θ is used to select features whose maximum ratios $\geq \theta$. The process of finding discriminating features from pattern class table using the threshold pair (α, θ) is shown in Figure 3.

Then, a majority voting rule-based classifier is constructed, which has $(d+1)$ sets of discriminative features, 1 set corresponds to the *discriminative clique patterns* and another d sets correspond to the sets of discriminative features extracted from the d side information pattern class tables. Given, a test graph G_{test}, the clique set C of G_{test} is extracted using Algorithm 1. All distinct types of side information are directly hashed and stored into the array of sets SI according to their side information type. Then, all the extracted features from graph G_{test} are given to the majority voting based classifier and for each hashed feature that has a corresponding discriminative pattern of its own type, the class label of that discriminative pattern is taken a vote for the class label of the graph G_{test}. Finally the label of the test graph G_{test} is determined from the majority of the class label votes from the classifier. This is detailed in Algorithm 4.

Algorithm 4. Graph Stream Classification with Side Information

Input : A test graph G_{test} from the stream
Output: Predict the class label of G_{test}

```
1  begin
2      L ← φ
3      C_test = Clique − Detect(G_test);
4      for i ← 1 to size(C_test) do
5          H_{0,i} ← hash(C_i)
6          L_{0,i} ← Find − Rule(Δ_0, H_{0,i})
7      end for
8      for l ← 1 to d do
9          for i ← 1 to size(SI_l) do
10             H_{l,i} ← hash(SI_{l,i})
11             L_{l,i} ← Find − Rule(Δ_l, H_{l,i})
12         end for
13     end for
14     Label = Majority − Voting − Classifier(L)
15 end
```

3 Experimental Results

In this section we will present the experimental results and techniques. We tested our proposed approach, GSCS for effectiveness and efficiency and compared the results with existing DICH [6] approach. We used two real word data sets, CORA [14] and IMDB [15].

CORA Data Set. The CORA data set contains scientific articles on computer science. To create a graph stream from the articles we considered the co-author relationship as edges of the graph. Research topics were used as class labels. Terms and citations were used as side information.

IMDB Data Set. The Internet Movie Database, IMDB, is a website which contains detailed information about movies and TV shows. We scraped a sample of movies from IMDB which contains 3535 movies released in United States during 2000-2015. We created graph object from each movies using actor-pair as edges. The genre of the movies were used as class labels. We extracted two types of side information: (a) plot keywords and (b) directors.

We used 90% of each data set as training data and used the other 10% as testing data. There are 4 parameters to consider while using GSCS and DICH, frequent pattern threshold α, dicriminative pattern threshold θ, the node space size N, the hash space size P. We vary the parameters and show how GSCS performs in comparison to DICH. The default values for parameters are: α = .05, θ = .3, N = 500, P = 10000.

All tests were run on an Asus K550JK running Windows 8.1 x64 with a 2.8 GHz Intel Core i5-4200H CPU and 8 GB of main memory. Both approaches were implemented with C++ and were compiled with tdm-gcc 4.9.2.

Figure 4, 5, 6 and 7 shows different effectiveness measures for GSCS and DICH in both data sets. We measured precision, recall, balanced accuracy and F1 score for multi-class classification [16]. From the graph it is apparent that GSCS performs better than DICH in terms of classification effectiveness. The extra dimensionality provided by the side information helps GSCS to be more effective.

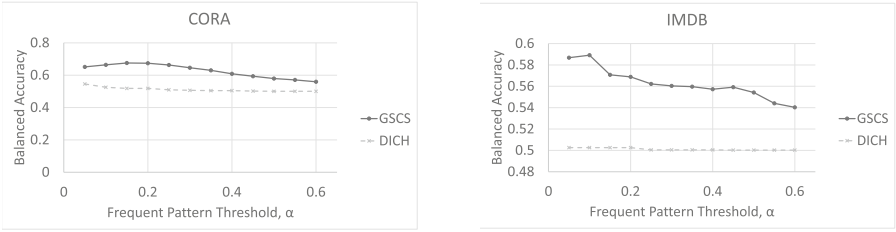


Fig. 4. Balanced accuracy comparison between GSCS and DICH by varying α

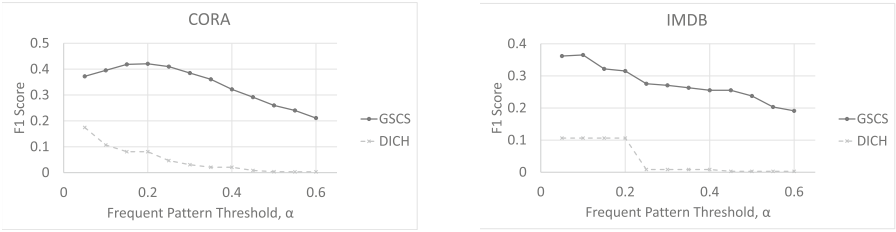


Fig. 5. F1 score comparison between GSCS and DICH by varying α

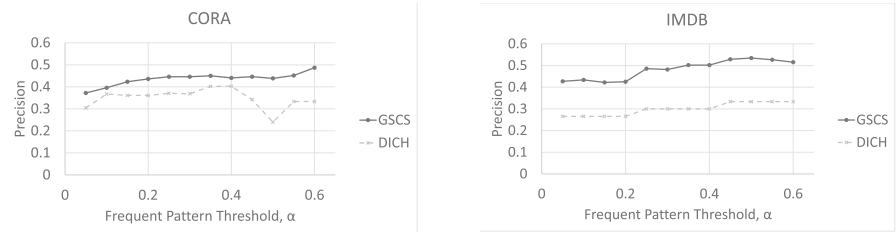


Fig. 6. Precision comparison between GSCS and DICH by varying α

Figure 8 shows that GSCS takes a little bit extra time than DICH. This is the overhead of processing the side information along with the graph structure.

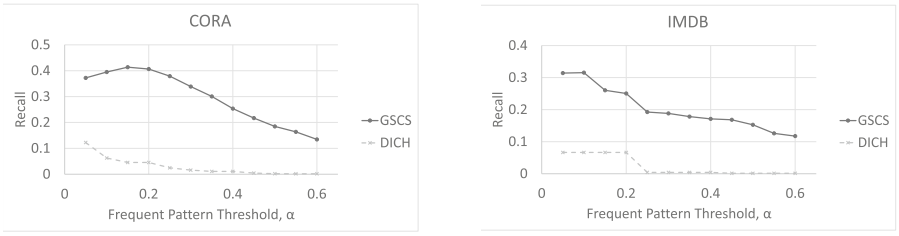
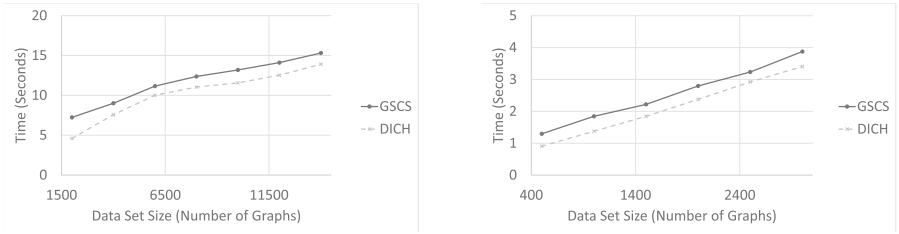


Fig. 7. Recall comparison between GSCS and DICH by varying α

GSCS spends most of its time in feature extraction from the graph and side information. GSCS can process on an average 600 graphs per second in both data sets which is quite good for the stream scenario.



(a) CORA (b) IMDB

Fig. 8. Time comparison between GSCS and DICH by varying data set size

Since GSCS does not store any data other than in-memory tables for graph and side information, the memory usage is almost constant. In our experiments with both data sets the maximum memory used by GSCS was 100MB. This makes GSCS ideal for the stream scenario.

From the above experimentation, we can conclude that, GSCS is effective in classification of graphs with side information and efficient in the graph stream scenario. Also GSCS outperforms the state of the art graph stream classification approach DICH by providing better performance.

4 Conclusions

In this paper we proposed the first approach which incorporates side information in the classification process of graph streams. The existing graph stream classification algorithms only consider the graph structure and do not utilize side information in the classification process. Many real life applications generate graph streams where side information is associated with each graph, which contains semantically meaningful information relevant to the graph structure, thus can help to build a more accurate classification model. Mining graph streams is a challenging problem because of the high computational cost for graph structure mining and storage difficulty in stream scenario. In our proposed approach

GSCS, a graph is first compressed and decomposed into maximal cliques. Then both clique patterns and side information are hashed and stored into corresponding in-memory summary tables for discriminative feature extraction to build a majority voting classifier. The experimental results show that our proposed approach significantly outperforms state-of-the-art method [6] which only considers graph structure and thus depicts the potential of side information in the graph classification process. Experimental results also infer that our approach is efficient and scalable enough to be applied in real life graph stream scenario.

References

1. Google scholar. Website `http://scholar.google.com/` (last access: March 27, 2015)
2. Dblp: computer science bibliography. Website `http://dblp.uni-trier.de/` (last access: March 27, 2015)
3. Aggarwal, C.C.: On classification of graph streams. In: SDM 2011, pp. 652–663 (2011)
4. Samiullah, M., Ahmed, C.F., Nishi, M.A., Fariha, A., Abdullah, S.M., Islam, M.R.: Correlation mining in graph databases with a new measure. In: Ishikawa, Y., Li, J., Wang, W., Zhang, R., Zhang, W. (eds.) APWeb 2013. LNCS, vol. 7808, pp. 88–95. Springer, Heidelberg (2013)
5. Wang, Z., Tan, Y., Zhang, M.: Graph-based recommendation on social networks. In: APWeb 2010, pp. 116–122 (2010)
6. Chi, L., Li, B., Zhu, X.: Fast graph stream classification using discriminative clique hashing. In: Pei, J., Tseng, V.S., Cao, L., Motoda, H., Xu, G. (eds.) PAKDD 2013, Part I. LNCS, vol. 7818, pp. 225–236. Springer, Heidelberg (2013)
7. Li, B., Zhu, X., Chi, L., Zhang, C.: Nested subtree hash kernels for large-scale graph classification over streams. In: ICDM 2012, pp. 399–408 (2012)
8. Guo, T., Chi, L., Zhu, X.: Graph hashing and factorization for fast graph stream classification. In: CIKM 2013, pp. 1607–1612 (2013)
9. Yu, P.S., Zhao, Y.: On graph stream clustering with side information. In: SIAM, pp. 139–150 (2013)
10. Soufiani, H.A., Airoldi, E.: Graphlet decomposition of a weighted network. In: AISTATS 2012, pp. 54–63 (2012)
11. Bron, C., Kerbosch, J.: Finding all cliques of an undirected graph (algorithm 457). Commun. ACM 16(9), 575–576 (1973)
12. Tomita, E., Tanaka, A., Takahashi, H.: The worst-case time complexity for generating all maximal cliques. In: Chwa, K.-Y., Munro, J.I. (eds.) COCOON 2004. LNCS, vol. 3106, pp. 161–170. Springer, Heidelberg (2004)
13. Moon, J.W., Moser, L.: On cliques in graphs. Israel Journal of Mathematics 3(1), 23–28 (1965)
14. Cora data set. Website `http://people.cs.umass.edu/~mccallum/data.html` (last access: March 27, 2015)
15. Imdb - internet movie database. Website `http://www.imdb.com` (last access: March 27, 2015)
16. Sokolova, M., Lapalme, G.: A systematic analysis of performance measures for classification tasks. Inf. Process. Manage. 45(4), 427–437 (2009)