Data Min Knowl Disc (2017) 31:400–423
DOI 10.1007/s10618-016-0478-6

Discovering rare categories from graph streams

Dawei Zhou[1] · **Arun Karthikeyan**[1] · **Kangyang Wang**[1] · **Nan Cao**[2] · **Jingrui He**[1]

Received: 6 May 2016 / Accepted: 13 September 2016 / Published online: 28 September 2016

Abstract Nowadays, massive graph streams are produced from various real-world applications, such as financial fraud detection, sensor networks, wireless networks. In contrast to the high volume of data, it is usually the case that only a small percentage of nodes within the time-evolving graphs might be of interest to people. Rare category detection (RCD) is an important topic in data mining, focusing on identifying the initial examples from the rare classes in imbalanced data sets. However, most existing techniques for RCD are designed for static data sets, thus not suitable for time-evolving data. In this paper, we introduce a novel setting of RCD on time-evolving graphs. To address this problem, we propose two incremental algorithms, *SIRD* and *BIRD*, which are constructed upon existing density-based techniques for RCD. These algorithms exploit the time-evolving nature of the data by dynamically updating the detection models enabling a "time-flexible" RCD. Moreover, to deal with the cases where the exact priors of the minority classes are not available, we further propose a modified version named *BIRD*-LI based on *BIRD*. Besides, we also identify a critical task in RCD named query distribution, which targets to allocate the limited budget among multiple time steps, such that the initial examples from the rare classes are detected as early as possible with the minimum labeling cost. The proposed incremental RCD algorithms and various query distribution strategies are evaluated empirically on both synthetic and real data sets.

Keywords Rare category detection · Time-evolving graph · Incremental learning

Responsible editor: Jian Pei.

✉ Dawei Zhou
dzhou23@asu.com

[1] CIDSE, Arizona State University, Tempe, AZ 85281, USA

[2] New York University Shanghai, Shanghai 200122, China

1 Introduction

Compared with the tremendous and rapidly changing data, the examples of interest to us only hold a very small portion. For instance, in financial synthetic identity detection (Phua et al. 2010), only a tiny proportion of identities are fraudulent, generated by mixing the identifying information from multiple sources. Such identities are created with the sole purpose of committing financial fraud. Another example is insider threat detection (Eberle et al. 2010), where only a small population amongst a big organization are malicious insiders involved in treacherous behaviors, such as sabotage, espionage, etc. The small percentage of data of interest to us is called the minority class or rare category, since such examples are often self-similar. Due to the rarity of the minority classes and the limited budget on querying the labeling oracle who can provide the true label of any example at a fixed cost, it is difficult to identify examples from such classes via random sampling. To efficiently deal with this problem, rare category detection (RCD) has been proposed to identify the very first example from the minority class, by requesting only a small number of labels from the oracle (Pelleg and Moore 2004).

Most, if not all, of existing RCD techniques are designed for static data. However, in many real-world applications, the data is not static but evolves with time, and so are the minority classes. Examples of such scenarios are listed as follows.

- In financial synthetic identity detection, within the transaction network, each identity could correspond to one specific node, and each transaction activity could correspond to one edge. Since each identity may keep updating his or her information, such as daily transactions and real-time online banking activities, the data is evolving over time. Our goal is to identify the identities and transactions, which have unusual characteristics and significantly differ from the majorities in the networks.
- In insider threat detection, the insiders intentionally change their behavior patterns over time to avoid being caught. In other words, the insiders may not be abnormal all the time when compared with normal employees. Thus, how to distinguish insiders and normal employees from evolving data is a challenge.
- In event detection in social networks, the snapshots of social networks are evolving every single second with updated vertex sets and updated edge sets, which means the event related vertex sets may shrink, expand or shift within the time-evolving social networks. Hence, how to model, capture and track the changing target events over evolving social networks would be the main task.

Straight-forward applications of existing RCD techniques in the preceding scenarios would be very time-consuming by constructing the models from scratches at each time step. Additionally, it is critical to allocate queries among different time steps from labeling oracle, which may help detect the initial rare examples as early as possible to avoid further damage.

Addressing this issue, in this paper, for the first time, we study the problem of incremental RCD. Specifically, we first propose two incremental algorithms, i.e., *SIRD* and *BIRD*, to detect the initial examples from the minority classes under different dynamic settings. The key idea is to efficiently update our detection model by local changes

instead of reconstructing it from scratches based on the updated data at a new time step, so as to reduce the time cost of redundant and repeating computations. Furthermore, we relax the requirement of the exact priors with a soft upper bound for all the minority classes to provide a modified version—*BIRD*-LI. Finally, we study a unique problem of query distribution under the dynamic settings, which distributes allocated labeling budget among different time steps, and propose five query distribution strategies. This paper is extended from our previous work (Zhou et al. 2015b) in terms of the detailed algorithm, theoretical justification and the comprehensive experiments on real time-evolving graph data sets.

The rest of our paper is organized as follows. In Sect. 2, we briefly review the related work on both RCD and time-evolving graph mining. In Sect. 3, we study incremental RCD and propose three algorithms, i.e., *SIRD*, *BIRD* and *BIRD*-LI, to address different dynamic settings. Then, in Sect. 4, we introduce the unique problem of query distribution under the dynamic settings, and propose five strategies for allocating the labeling budget among different time steps. In Sect. 5, we demonstrate our models on both synthetic and real data sets. Finally, we conclude this paper in Sect. 6.

2 Related work

2.1 Rare category analysis

RCD refers to the problem of identifying the initial examples from under-represented minority classes in an imbalanced data set. Lots of techniques have been developed for solving the problem of RCD in the past decade. Pelleg and Moore (2004) proposed a mixture model-based algorithm, which is the first attempt in this area. In He and Carbonell (2007) and He et al. (2008), the authors developed an innovative method to detect rare categories via unsupervised local-density-differential sampling strategy. Dasgupta and Hsu (2008) presented an active learning scheme via exploiting the cluster structure in data sets. In He et al. (2010), the authors introduced a novel problem called rare category characterization, which not only detects but also characterizes the rare categories, and proposed an optimization framework to explore the compactness of rare categories. More recently, in Liu et al. (2014), two prior-free methods were proposed in order to address the RCD problem without any prior knowledge. In Zhou et al. (2015a), the authors proposed a framework named MUVIR, which could leverage existing RCD models on each single view and estimate the overall probability of each example belonging to the minority classes. However, all of the preceding works focus on the static data sets, and few works have been proposed to address the problem of RCD under dynamic settings.

2.2 Outlier detection on streaming data

With the improvement of hardware technology on data collection, many applications require efficient mechanisms to process the outlier detection on streaming data (Gupta et al. 2014). Tons of algorithms have been proposed in the past decade. Yamanishi and Takeuchi (2002) and Yamanishi et al. (2004) presented an online discounting

learning algorithm to incrementally update a probabilistic mixture model and capture outliers in data streams. In Aggarwal and Philip (2010), the authors proposed online clustering methods, which maintained a dynamic clustering model to identify outliers under dynamic settings. Instead of only updating parameters of the prediction model, dynamic Bayesian network (Hill et al. 2007), a modifiable model, was proposed to detect anomalies from environmental sensor data. Different from regular data streams, distributed data streams are collected from distributed sensors over time. Bettencourt et al. (2007) and Franke and Gertz (2008) studied the problem of outlier detection on multiple types of distributed data streams, such as air temperature sensor network data, water pollution sensor network data and wind sensor network data. Different from outlier detection, rare category detection assumes that the anomalies belong to multiple distinct classes, in the sense that the within-class similarities are much larger than the between-class similarities. In this paper, we aim to discover these rare categories over a series of time-evolving graphs.

2.3 Graph based anomaly detection

In the literature, there are abundant works focusing on anomaly detection in static graphs. Basically, all of the existing works study two types of static graphs: plain static graphs and attributed static graphs. Plain graph assumes the only information we have is the structure of graph. This category of anomaly detection methods aims to exploit the structure of graphs and mine the unrepresentative pattern of anomalies, e.g., global graph structure methods (Kang et al. 2010; Henderson et al. 2010); local graph structure methods (Akoglu et al. 2010; Kang et al. 2011; Gupte and Eliassi-Rad 2012). Attributed graph assumes both the structure and the coherence of attributes are given. Müller et al. (2013) and Gao et al. (2010) proposed node outlier ranking methods on static attributed graphs. Yagada Davis et al. (2011) characterized anomalies by discrediting the numerical attributes into "outlier score". In Sricharan and Das (2014), the authors proposed a fast algorithm which could detect the node relationships for localizing anomalous changes in time-evolving graphs.

More recently, an increasing number of research has been conducted under dynamic graph settings. For examples, in Leskovec et al. (2005), the authors analyzed the properties of the time evolution of real graphs and proposed a "forest fire" graph-generative model; Backstrom et al. (2006) studied the problem of community evolution and developed a novel method to measure the movement of individuals among communities; in Kumar et al. (2010), the authors focused on the difficulties of conversation dynamics and proposed a simple mathematical model in order to generate basic conversation structures; in Berlingerio et al. (2012) and Koutra et al. (2011), the authors proposed several graph similarity measurements to detect the discontinuity in dynamic social networks. Besides, to reduce the time complexity, in Tong et al. (2008), the authors proposed a fast proximity tracking method for dynamic graphs; in Koutra et al. (2012), the authors used tensor decomposition techniques to efficiently obtain the "scores" for anomalies on dynamic graphs; in Fan et al. (2013), the authors proposed a new graph-pattern matching algorithm, which can avoid cubic-time computation; Akoglu et al. (2014) raised a divide-and-conquer framework, which could find the k-nearest-

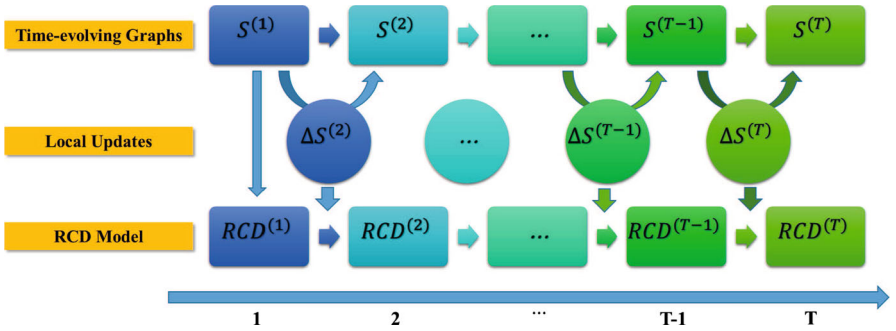

Fig. 1 Incremental rare category detection

neighbors efficiently on high volume of time-evolving graphs. BIRD approach (Zhou et al. 2015b) provided a fast updating method for the challenging problem of RCD on time-evolving graphs. In this paper, we propose several fast-updating RCD methods which could incrementally update the models based on local changes on time-evolving graphs. This paper extends our previous work (Zhou et al. 2015b) substantially by providing the detailed algorithm, theoretical justification and the comprehensive empirical evaluations on real-world time-evolving graph data sets, which are not presented in the previous version.

3 Incremental rare category detection

In this section, we introduce the proposed framework of incremental RCD. Our methods exploit the time-evolving nature of dynamic graphs and update the RCD model incrementally based on the local updates from time to time. To the best of our knowledge, existing RCD methods are all designed for static data sets, while we target a more challenging setting, in which the data is presented as time-evolving graphs. Notice that we allow the support regions of the majority and minority classes to overlap with each other in the feature space, which makes our algorithm widely applicable to a variety of real-world problems.

3.1 Notation

Suppose we are given a series of time-evolving graphs $\{S^{(1)}, \ldots, S^{(T)}\}$, which are shown in Fig. 1. For any time step $t = 1, \ldots, T$, the vertices in $S^{(t)}$ are identical and only edges change over time. We assume $y_i^{(t)} = 1$ corresponds to the majority class with prior $p_1^{(t)}$, and the remaining classes are the minority classes with priors $p_c^{(t)}$ at time step t. We use $\Delta S^{(t)}$ to denote the new edges and updated weights that appear at time step t. Specifically, we have $\Delta S^{(t)} = S^{(t)} - S^{(t-1)}$.

In the following part of this paper, we use the convention in Matlab to represent matrix elements, e.g., $S^{(t)}(i, j)$ is the element at ith row and the jth column of matrix $S^{(t)}$, and $S^{(t)}(:, j)$ is the jth column of matrix $S^{(t)}$, etc. The main symbols we used in this paper are listed in Table 1.

Table 1 Symbols

Symbol	Description
n	Number of nodes
$m^{(t)}$	Number of updated edges
x_i	ith nodes in data set
t	Time step
C	Number of classes
$p_c^{(t)}$	Proportion of classes c
α	Constraint parameter
I	Identity matrix
$S^{(t)}$	$n \times n$ original aggregated adjacency matrix at time t
$\Delta S^{(t)}$	$n \times n$ updating matrix for $S^{(t-1)}$
$M^{(t)}$	Normalized $n \times n$ aggregated adjacency matrix at time t
$\Delta M^{(t)}$	$n \times n$ updating matrix for $M^{(t-1)}$
$NN^{(t)}$	$n \times n$ neighbor information matrix at time step t
$A^{(t)}$	$n \times n$ global similarity matrix at time step t

3.2 Static rare category detection

In static RCD, we repeatedly select examples to be labeled by the oracle until all the minority classes in a static data set are discovered. One approach for static RCD is to make use of the manifold structure for identifying rare category examples. In He et al. (2008), authors developed a graph-based RCD method named GRADE. In GRADE algorithm, they first construct a pair-wise similarity matrix W' and its corresponding diagonal matrix D, whose elements are the row sums of W'. Then, they calculate the normalized matrix W as follows.

$$W = D^{-1/2} W' D^{-1/2}$$

Based on the normalized pair-wise similarity matrix W, they construct a global similarity matrix A as follows.

$$A = (I_{n \times n} - \alpha W)^{-1} \tag{1}$$

where α is a small enough positive discounting constant in the range of $(0, 1)$. By constructing the global similarity matrix, the changes of local density would become sharper near the boundary of the minority classes. Based on this intuition, GRADE could identify minority classes with much fewer queries than random sampling. However, the time complexity of calculating the global similarity matrix and finding each example's (K)th nearest neighbor is $O(n^3 + K \cdot n^2)$, which is not efficient enough for time-evolving RCD applications.

3.3 Dynamic rare category detection

In this subsection, we introduce two fast-updating incremental RCD algorithms (*SIRD* and *BIRD*) to deal with the RCD problem on time-evolving graphs. Both methods greatly reduce the computation cost for updating the global similarity matrix and finding each node's Kth nearest neighbor. Similar to static rare category detection, we target the challenging case where the minority classes are not separable from the majority classes.

3.3.1 Single update

We first consider the simplest case: only one self-loop edge (a, a) changes at time step t. In other words, there is only one non-zero element (a, a) in $\triangle S^{(t)}$. Similar to He et al. (2008), we use $M^{(t)}$ to denote the normalized aggregated adjacency matrix, which is defined as follows.

$$M^{(t)} = (D^{(t)})^{-1/2} S^{(t)} (D^{(t)})^{-1/2} \tag{2}$$

Besides, let $\Delta M^{(t)}$ denote the updating matrix for $M^{(t)}$, such as $\Delta M^{(t)} = M^{(t)} - M^{(t-1)}$. Clearly, there is also only one non-zero element existing in $\Delta M^{(t)}$. Hence, $\Delta M^{(t)}$ could be easily decomposed into the product of two column vectors uv^T, where u and v are two column vectors with only one non-zero element. To address this problem, we first introduce Theorem 1 to update the global similarity matrix $A^{(t)}$ more efficiently.

Theorem 1 *The global similarity matrix $A^{(t)}$ at time step t can be exactly updated from global similarity matrix $A^{(t-1)}$ at the previous time step $t-1$ by the following equation:*

$$A^{(t)} = A^{(t-1)} + \alpha \frac{A^{(t-1)} u v^T A^{(t-1)}}{I + v^T A^{(t-1)} u}$$

where u and v^T are the two vectors decomposed from updating matrix $\Delta M^{(t)}$

Proof Suppose there is only one edge updated at time step t, and we have $\Delta M^{(t)} = uv^T$. Thus, Eq. (1) could be rewritten as follows.

$$\begin{aligned} A^{(t)} &= \left(I - \alpha M^{(t)}\right)^{-1} \\ &= \left(I - \alpha M^{(t-1)} - \alpha \Delta M^{(t)}\right)^{-1} \\ &= \left(I - \alpha M^{(t-1)} - \alpha u v^T\right)^{-1} \end{aligned} \tag{3}$$

By applying the Sherman–Morrison Lemma (Sherman and Morrison 1950), we have

$$A^{(t)} = A^{(t-1)} + \alpha \frac{A^{(t-1)} u v^T A^{(t-1)}}{I + v^T A^{(t-1)} u} \tag{4}$$

Hence, the global similarity matrix $A^{(t)}$ in our Algorithm 1 could be exactly updated at each time step. □

In Theorem 1, we can see column vectors u and v are essential for updating the global similarity matrix $A^{(t)}$. To reduce the computational complexity, in Algorithm 1, we use an approximate method to calculate the two column vectors u and v. The details are described as follows. We first assume that the updated edges at time step t have little impact on the row sum of adjacency matrix $S^{(t)}$ when the number of updated edges is extremely smaller than the total number of edges. Thus, we have

$$D^{(t)} \cong D^{(t-1)}$$

To normalize aggregated adjacency matrix of $S^{(t)}$ and $S^{(t-1)}$, we have

$$M^{(t)} = \left(D^{(t)}\right)^{-1/2} S^{(t)} \left(D^{(t)}\right)^{-1/2} \tag{5}$$

$$M^{(t-1)} = \left(D^{(t)}\right)^{-1/2} S^{(t-1)} \left(D^{(t-1)}\right)^{-1/2} \tag{6}$$

By Eqs. (5, 6), we have

$$\Delta M^{(t)} = \left(D^{(t-1)}\right)^{-1/2} \Delta S^{(t)} \left(D^{(t-1)}\right)^{-1/2} \tag{7}$$

As $\Delta M^{(t)} = uv^T$, we could easily assign $u = D(:, a)^{-1/2}$ and $v = \Delta S^{(t)}(a, b)D(:, b)^{-1/2}$.

Besides, as the time complexity of constructing a new neighbor information matrix $NN^{(t)}$ is $O(K^{(t)} \cdot n^2)$, we introduce Theorem 2 to efficiently update $NN^{(t)}$.

Theorem 2 *Suppose there is only one self loop edge* (a, a) *being updated at time step* t. *If it satisfies the condition that* $\frac{\alpha}{I+v^T A^{(t-1)}u} \leq \frac{\delta_i^{(t-1)}}{A_{i,a}^{(t-1)}\phi_a}$, *the first* $K^{(t)}$ *elements in* $NN^{(t)}(i, :)$ *are the same as* $NN^{(t-1)}(i, :)$.

Proof Based on Theorem 1, we have

$$\begin{aligned} A^{(t)} &= A^{(t-1)} + \alpha \frac{A^{(t-1)} u v^T A^{(t-1)}}{I + v^T A^{(t-1)} u} \\ &= A^{(t-1)} + \alpha \frac{A^{(t-1)} \Delta M^{(t)} A^{(t-1)}}{I + v^T A^{(t-1)} u} \end{aligned} \tag{8}$$

Since u and v are column vectors that contain only one non-zero element, then $I + v^T A^{(t-1)} u$ is a constant value, which means it is just a scalar and will not change the order of elements in $NN^{(t)}$.

From Eq. (8) we also have the updating rule for each element (i, j) in $A^{(t)}$

$$A_{i,j}^{(t)} = A_{i,j}^{(t-1)} + \beta A_{i,a}^{(t-1)} A_{a,j}^{(t-1)} \tag{9}$$

where $\beta = \frac{\alpha}{I+v^T A^{(t-1)}u}$ is also a constant.

Let $\delta_i^{(t-1)} = min_{j=1}^{K^{(t)}} \left\{NN^{(t-1)}(i, j) - NN^{(t-1)}(i, j+1)\right\}$ denote the smallest adjacent difference among the first $K^{(t)}$ elements in the ith row of $NN^{(t-1)}$, and $\phi_a = NN^{t-1}(a, 1)$ denote the largest element in row a. Intuitively, as long as the largest value of $\beta A_{i,a}^{(t-1)} A_{a,j}^{(t-1)}$ is smaller than the smallest adjacent gap between any of the first $K^{(t)}$ nodes in the ith row of $NN^{(t)}$, we can claim that the order of these sorted $K^{(t)}$ nodes will not change. Therefore, based on Eq. (9), if the condition satisfies

$$\frac{\alpha}{I + v^T A^{(t-1)} u} \leq \frac{\delta_i^{(t-1)}}{A_{i,a}^{(t-1)} \phi_a} \tag{10}$$

we can claim that the first $K^{(t)}$ elements in $NN^{(t)}(i, :)$ will not change. □

Based on Theorem 2, we can identify the rows of $NN^{(t)}$, in which the order of the $K^{(t)}$ largest elements will not change. Thus, we only need to update the disordered rows in $NN^{(t)}$.

The single-updated incremental RCD algorithm (*SIRD*) is shown in Algorithm 1. In Step 1 to Step 2, we first initialize the diagonal matrix D and neighbor information matrix $NN^{(1)}$ at time step 1. In Step 4, let $K^{(t)}$ represent the number of nodes in the largest minority class at time step t. Then, from Step 5 to Step 6, we update the global similarity matrix at each time step. Step 7 to Step 9 updates the rows in $NN^{(t)}$, of which the $K^{(t)}$ largest elements are changed. Step 11–20 is the query process. First of all, we calculate the class specific a^c at Step 13, which is the largest global similarity to the $k_c^{(th)}$ nearest neighbor. Then, in Step 14, we count the number of its neighbors whose global similarity is larger than or equal to a^c, and let n_i^c denote the counts for each node x_i. In Step 16, we calculate the score of each node x_i, which represents the change of local density. At last, we select the nodes with the largest score and let them be labeled by oracle. The query process only terminates as long as all the minority classes are discovered.

The efficiency of the updating process for Algorithm 1 is given by the following lemma.

Lemma 1 *The computational cost of the updating process at each time step in Algorithm 1 is* $O(n^2 + l \cdot K^{(t)} \cdot n)$.

Proof As described before, the computational cost for normalization and decomposition process is O(n). Then, in Step 6, compared to the straightforward computation, i.e., $A^{(t-1)} = (I - \alpha M^{(t)})^{-1}$, we reduce the time complexity from $O(n^3)$ to $O(n^2)$ by avoiding the matrix inverse computation. Furthermore, from Step 7 to Step 9, we simplify the resorting process by only updating the rows, in which the top $K^{(t)}$ elements are disordered. Suppose l is the total number of rows in $NN^{(t)}$, which does not satisfy Eq. (10), then the computational cost is reduced from $O\left(K^{(t)} \cdot n^2\right)$ to $O\left(l \cdot K^{(t)} \cdot n\right)$. By leveraging each part, the computational cost of the updating process is $O\left(n^2 + l \cdot K^{(t)} \cdot n\right)$. □

ALGORITHM 1: *SIRD* Algorithm

Input: $M^{(1)}, A^{(1)}, \Delta S^{(2)}, \ldots, \Delta S^{(T)}, p_c^{(t)}, \alpha$.
Output: The set I of labeled nodes
1: Construct the $n \times n$ diagonal matrix D, where $D_{ii} = \sum_{(j=1)}^{n} S^{(1)}, i = 1, \ldots, n$.
2: Sort row i of $A^{(1)}$ decreasingly and save into $NN^{(1)}(i,:)$, where $i = 1, \ldots, n$.
3: **for** t=2:T **do**
4: Let $K^{(t)} = \max_{c=2}^{C} n \times p_c^{(t)}$.
5: Let column vector $u = D(:,a)^{-1/2}$, and column vector $v = \Delta S^{(t)}(a,a)D(:,a)^{-1/2}$, where $\Delta S^{(t)}(a,a)$ is the non-zero element in $\Delta S^{(t)}$.
6: Update the global similarity matrix as follows.

$$A^{(t)} = A^{(t-1)} + \alpha \frac{A^{(t-1)} u v^T A^{(t-1)}}{I + v^T A^{(t-1)} u}$$

7: **for** i=1:n **do**
8: Based on Theorem 2, identify whether the first $K^{(t)}$ elements of $NN^{(t)}$ (i,:) are changed. If true, update the first $K^{(t)}$ elements in $NN^{(t)}(i,:)$; otherwise, let $NN^{(t)}(i,:) = NN^{(t-1)}(i,:)$.
9: **end for**
10: **end for**
11: **for** c = 2:C **do**
12: Let $k_c = n \times p_c^{(T)}$
13: Find the first k_c elements in each row of $NN^{(T)}$. Set a^c to be the largest value of them.
14: Let $KNN^c(x_i, a^c) = \{x | NN^{(T)}(i,j) > a^c\}$, and $n_i^c = |KNN^c|$, where $i = 1, \ldots, n$ and $j = 1, \ldots, n$.
15: **for** index = 1: n **do**
16: For each node x_i has been labeled y_i, if $A^{(T)} > a^{y_i}$, $score_j = -\infty$; else, let $score_i = \max_{A^{(T)}(i,j) > \frac{a^c}{index}} (n_i^c - n_j^c)$
17: Select the nodes x with the largest score to labeling oracle.
18: If the label of x is exact class c, break; else, mark the class that x belongs to as discovered.
19: **end for**
20: **end for**

3.3.2 Batch update

In most real world applications, we always observe that a batch of edges change at the same period. Specifically, the updated aggregated adjacency matrix $\Delta M^{(t)}$ may have more than one non-zero element. Hence, $\Delta M^{(t)}$ cannot be decomposed into two column vectors, and Theorem 2 could not be applied in this condition. In this part, we introduce Theorem 3 to help us update the neighbor information matrix $NN^{(t)}$ when a batch of edges are changed.

Theorem 3 *Suppose there are m edges* $\{(a^1, b^1), \ldots, (a^m, b^m)\}$ *being updated at time step t. The first* $K^{(t)}$ *elements in* $NN^{(t)}(i,:)$ *are the same as* $NN^{(t-1)}(i,:)$*, if it satisfies the condition that*

$$\frac{\alpha}{I + V^T A^{(t-1)} U} \leq \min_{i=1,\ldots,m} \{T_i\}$$

where $T_i = \min \left\{ \frac{\delta_i^{(t-1)}}{A_{i,a^i}^{(t-1)} \phi_{b^i}}, \frac{\delta_i^{(t-1)}}{A_{i,b^i}^{(t-1)} \phi_{a^i}} \right\}$.

Proof Since the aggregated adjacency matrix $M^{(t)}$ is a symmetric matrix, then, each element (a, b), where $a \neq b$, has a symmetrical element (b, a) in $M^{(t)}$.

When the two edges (a, b) and (b, a) are updated at time step t, we have $\Delta M^{(t)} = \Delta M_1^{(t)} + \Delta M_2^{(t)}$, where $\Delta M_1^{(t)}$ has only one non-zero element (a, b), and $\Delta M_2^{(t)}$ has only one non-zero element (b, a). Similar to Eq. (8), we have an approximate updating rule as follows.

$$A^{(t)} \cong A^{(t-1)} + \alpha \frac{A^{(t-1)} \Delta M_1^{(t)} A^{(t-1)}}{I + (v^{(1)})^T A^{(t-1)} u^{(1)}} + \alpha \frac{A^{(t-1)} \Delta M_2^{(t)} A^{(t-1)}}{I + (u^{(1)})^T A^{(t-1)} v^{(1)}} \tag{11}$$

where $\Delta M_1^{(t)} = u^{(1)}(v^{(1)})^T$, $\Delta M_2^{(t)} = v^{(1)}(u^{(1)})^T$ and $u^{(1)}$, $v^{(1)}$ are two column vectors.

Besides, we also have

$$A^{(t)} = A^{(t-1)} + \beta \left(A^{(t-1)} \Delta M_1^{(t)} A^{(t-1)} + A^{(t-1)} \Delta M_2^{(t)} A^{(t-1)} \right)$$

where $\beta = \frac{\alpha}{I + (v^{(1)})^T A^{(t-1)} u^{(1)}}$, and β is a constant.

Therefore, $A_{i,j}^{(t)} = A_{i,j}^{(t-1)} + \beta A_{i,a}^{(t-1)} A_{b,j}^{(t-1)} + \beta A_{i,b}^{(t-1)} A_{a,j}^{(t-1)}$.

Based on Theorem 2, we can claim that the largest $K^{(t)}$ elements in $NN^{(t)}(i, :)$ will not change, if it satisfies

$$\frac{\alpha}{I + V^T A^{(t-1)} U} \leq T_1 \tag{12}$$

where $T_1 = \min \left\{ \frac{\delta_i^{(t-1)}}{A_{i,a^1}^{(t-1)} \phi_{b^1}}, \frac{\delta_i^{(t-1)}}{A_{i,b^1}^{(t-1)} \phi_{a^1}} \right\}$.

Similarly, when there are $m^{(t)}$ pairs of edges being updated at time step t, we can claim that if it satisfies

$$\frac{\alpha}{I + V^T A^{(t-1)} U} \leq \min_{m=1}^{m^{(t)}} \{T_m\} \tag{13}$$

where $T_m = \min \left\{ \frac{\delta_i^{(t-1)}}{A_{i,a^c}^{(t-1)} \phi_{b^c}}, \frac{\delta_i^{(t-1)}}{A_{i,b^c}^{(t-1)} \phi_{a^c}} \right\}$, then the first $K^{(t)}$ elements in $NN^{(t)}(i, :)$ will not change. □

The Batch-update Incremental RCD (*BIRD*) algorithm is shown in Algorithm 2. Step 1 and Step 2 are the initialization step. Step 3 to Step 12 updates the global similarity matrix $A^{(t)}$ and the neighbor information matrix $NN^{(t)}$. Different from Algorithm 1, Step 5 to Step 8 iteratively updates the global similarity matrix $A^{(t)}$ based on $m^{(t)}$ changed edges. Another difference is that, in Step 10, T is the minimum value of the thresholds calculated from $m^{(t)}$ updated edges. At last, Step 13 to Step 20 is the query process, which is the same as what we have described in Algorithm 1.

The efficiency of batch-edges updating in Algorithm 2 is proved by the following lemma.

Lemma 2 *In Algorithm 2, the computational cost of the updating process at each time step is* $O(m^{(t)}n^2 + l \cdot K^{(t)} \cdot n)$.

Proof Different from Algorithm 1, in Algorithm 2, we have $m^{(t)}$ updated edges at time step t. We need to update the global similarity matrix $A^{(t)}$ for $m^{(t)}$ times. Thus, the computation cost of updating the global similarity matrix is $O(m^{(t)}n^2)$. Let l be the number of rows in $NN^{(t)}$, which do not satisfy Eq. (13). For updating these rows in $NN^{(t)}$, the computational complexity is $O(l \cdot K^{(t)} \cdot n)$. Thus, in total, the computation cost of updating process at each time step is $O(m^{(t)}n^2 + l \cdot K^{(t)} \cdot n)$. □

ALGORITHM 2: *BIRD* algorithm

Input: $M^{(1)}, A^{(1)}, \Delta S^{(2)}, \ldots, \Delta S^{(T)}, p_c^{(t)}, \alpha$.
Output: The set I of labeled nodes
1: Construct the $n \times n$ diagonal matrix D, where $D_{ii} = \sum_{(j=1)}^{n} S^{(1)}, i = 1, \ldots, n$.
2: Sort row i of $A^{(1)}$ decreasingly and save into $NN^{(1)}(i, :)$, where $i = 1, \ldots, n$.
3: **for** t=2:T **do**
4: Let $K^{(t)} = \max_{l=c}^{C} n \times p_c^{(t)}$.
5: **for** m = 1: $m^{(t)}$ **do**
6: Let column vector $u = D(:, a^m)^{-1/2}$, and column vector $v = \Delta S^{(t)}(a^m, b^m)D(:, b^m)^{-1/2}$, where $\Delta S^{(t)}(a^m, b^m)$ is the non-zero element in $\Delta S^{(t)}$.
7: Update the global similarity matrix as follows.

$$A^{(t)} = A^{(t-1)} + \alpha \frac{A^{(t-1)}uv^T A^{(t-1)}}{I + v^T A^{(t-1)}u}$$

8: **end for**
9: **for** i=1:n **do**
10: Based on Theorem 3, identify whether the first $K^{(t)}$ elements of $NN^{(t)}$ (i,:) are changed. If true, update the first $K^{(t)}$ elements in $NN^{(t)}(i, :)$; otherwise, let $NN^{(t)}(i, :) = NN^{(t-1)}(i, :)$.
11: **end for**
12: **end for**
13: **while** not all the classes have been discovered **do**
14: Calculate n_i for each node, where $i = 1, \ldots, n$.
15: **for** index = 1: n **do**
16: For each node x_i has been labeled y_i, if $A^{(T)} > a$, $score_j = -\infty$; else, let $score_i = \max_{A^{(T)}(i,j) > \frac{a}{index}} (n_i - n_j)$
17: Select the nodes x with the largest score to labeling oracle.
18: Mark the class that x belongs to as discovered.
19: **end for**
20: **end while**

3.4 *BIRD* with less information

In many applications, it may be difficult to obtain the exact priors of all the minority classes. In this subsection, we introduce *BIRD*-LI, a modified version of *BIRD*, which requires only an upper bound prior $p^{(t)}$ for all the minority classes existing at time step t. To be specific, *BIRD*-LI calculates $NN^{(1)}$ and diagonal matrix D at the first time step, which is the same as *BIRD*. Then, the global similarity matrix $A^{(t)}$ and the neighbor information matrix $NN^{(t)}$ could be updated from the first time step to the

time step T. The only difference between *BIRD* and *BIRD*-LI is that the size of the minority class $K^{(t)}$ is calculated based on an estimated upper bound prior instead of the exact ones for all the minority classes. After the updating process, *BIRD*-LI calculates an overall score for the minority classes and selects the nodes with the largest overall score to be labeled by the oracle.

BIRD-LI is described in Algorithm 3. It works as follows: Step 1 to Step 2 is the initial process for calculating $NN^{(1)}$ and the diagonal matrix D at the first time step. Step 3 to Step 12 aims to update the global similarity matrix $A^{(T)}$ and the neighbor information matrix $NN^{(T)}$ from time step 1 to time step T, which is the same as *BIRD*. The while loop from Step 13 to Step 20 is the query process. We calculate an overall score for the minority classes and select the nodes with the largest overall score to be labeled by the oracle. *BIRD*-LI only terminates the loop until all the classes are discovered.

ALGORITHM 3: *BIRD*-LI algorithm

Input: $M^{(1)}, A^{(1)}, \Delta S^{(2)}, \ldots, \Delta S^{(T)}, p^{(t)}, \alpha$.
Output: The set I of labeled nodes and the L of their labels
1: Construct the $n \times n$ diagonal matrix D, where $D_{ii} = \sum_{(j=1)}^{n} S^{(1)}, i = 1, \ldots, n$.
2: Sort row i of $A^{(1)}$ decreasingly and save into $NN^{(t)}(i, :)$, where $i = 1, \ldots, n$.
3: **for** t=2:T **do**
4: Let $K^{(t)} = n \times p^{(t)}$.
5: **for** m = 1: $m^{(t)}$ **do**
6: Let column vector $u = D(:, a^m)^{-1/2}$, and column vector $v = \Delta S^{(t)}(a^m, b^m) D(:, b^m)^{-1/2}$, where $\Delta S^{(t)}(a^m, b^m)$ is a non-zero element in $\Delta S^{(t)}$.
7: Update the global similarity matrix as follows.

$$A^{(t)} = A^{(t-1)} + \alpha \frac{A^{(t-1)} u v^T A^{(t-1)}}{I + v^T A^{(t-1)} u}$$

where u and v^T are the two vectors decomposed from normalized updating matrix $\Delta M^{(t)}$.
8: **end for**
9: **for** i=1:n **do**
10: Based on Equation 13, identify whether the first $K^{(t)}$ elements of $NN^{(t)}$ (i,:) are changed. If true, update the first $K^{(t)}$ elements in $NN^{(t)}(i, :)$; otherwise, let $NN^{(t)}(i, :) = NN^{(t-1)}(i, :)$.
11: **end for**
12: **end for**
13: **while** not all the classes have been discovered **do**
14: Calculate n_i for each node, where $i = 1, \ldots, n$
15: **for** index = 1: n **do**
16: For each node x_i has been labeled y_i, if $A^{(T)} > a$, $score_j = -\infty$; else, let $score_i = \max_{A^{(T)}(i,j) > \frac{a}{index}} (n_i - n_j)$
17: Select the nodes x with the largest score to labeling oracle.
18: Mark the class that x belongs to as discovered.
19: **end for**
20: **end while**

4 Query dynamics

In the previous section, we introduce two incremental RCD methods, i.e., *BIRD* and *SIRD*, which are used for identifying rare categories on time-evolving graphs. Taking

the advantage of *BIRD* and *SIRD*, we can efficiently update the initial RCD model at time step 0 to any future time step T. However, in many real word applications, we may not want to make queries to oracle at each time step or we may only be allowed with a limited number of queries. In these two cases, we introduce the following two open problems: (1) query locating (QL): how to find the optimal time step T to discover rare categories; (2) query distribution (QD): how to allocate limited number of queries into different time steps.

4.1 Query locating

First of all, we introduce the query locating problem. In real world applications, it could be the case that we are given a series of unlabeled time-evolving graphs $S^{(1)}, S^{(2)}, \ldots, S^{(T)}$, and we need to select an optimal time step T_{opt}, so that we can identify the minority classes with as less queries as possible (ALAP) and as early as possible (AEAP).

Before presenting our methods, let us introduce the two main factors that may affect the required number of queries in rare category detection. The first factor is $P(y = 2|x_i)$, which is the probability that example x_i belongs to the minority class given the features of x_i. A considerable number of works have already studied it before, such as MUVIR (Zhou et al. 2015a), GRADE (He et al. 2008) and NNDM (He and Carbonell 2007). Another factor is the density D_i at x_i, the definition of which is introduced in Theorem 4. When the density D_i at example x_i is high, it means there are many other examples close or similar to example x_i. Suppose there are two nodes x_i and x_j in graph G, where $P(y = 2|x_i) = P(y = 2|x_j)$ and $D_i > D_j$. Since the density at node x_i is larger than the density at node x_j, there is a higher probability that multiple classes are overlapped in the neighborhood of x_i. To some extent, higher density D_i implies higher probability of mis-classifying x_i. Thus, the value of $P(y = 2|x_i)$ is negatively correlated with the number of required queries, and the value of density D_i is positively correlated with the number of required labels. For the second factor, we introduce the following theorem to estimate local density based on the global similarity matrix constructed before.

Theorem 4 *For each example x_i, the density of x_i is positively correlated with $D_i^{(t)}$ at time step t, where $D_i^{(t)} = \Sigma_{j=1}^{n} A_{i,j}^{(t)}$, $i = 1, \ldots, n$.*

Proof Notice that $A^{(t)}(i, j)$ represents the global similarity between x_i and x_j. Thus, $D_i^{(t)} = \Sigma_{j=1}^{n} A_{i,j}^{(t)}$ is the aggregated global similarity between example x_i and all the existing examples on graph. If the density of example x_i is high, then it is always true that there are lots of examples which are similar or close to x_i. In other words, the density $D_i^{(t)}$ should be large. Similarly, when the density of x_i is low, the value of $D_i^{(t)}$ should be small. In conclusion, for any existing example x_i in the graph, its density is positively correlated with $D_i^{(t)}$. □

We let $score^{(t)} = P(y = 2|x_i^{(t)})$, which could be obtained using existing techniques such as MUVIR Zhou et al. (2015a) or GRADE He et al. (2008). Under this circumstance, we propose to assign the hardness of identifying the minority classes at

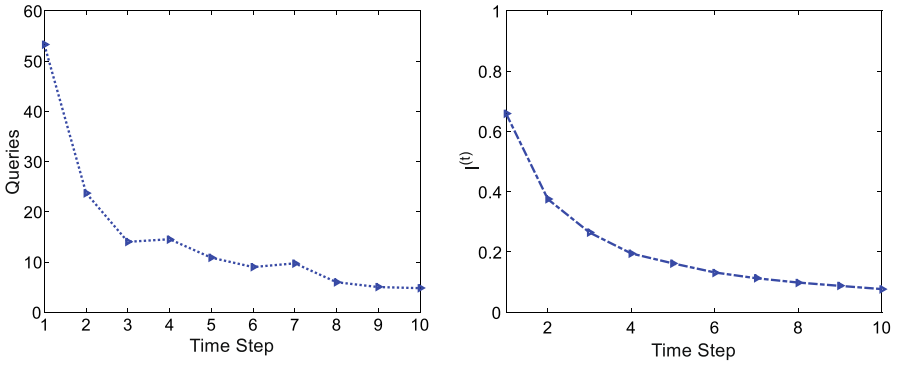

Fig. 2 Correlation

time step t as follows.

$$I^{(t)} = \left\{ k_c \max_{i=1,\ldots,k_c} \frac{score_i^{(t)}}{D_i^{(t)}} \right\}^{-1} \tag{14}$$

where k_c is the number of examples in the minority class c. In Fig. 2, the left figure shows the exact number of queries needed to identify rare categories from a series of time-evolving graphs. The right figure shows the value of $I^{(t)}$ calculated by Eq. (14). We can see these two curves are highly correlated.

Let $RS^{(t)}$ denote the number of required queries by random sampling at time step t. Simultaneously, let $C = \frac{RS^{(1)} - RS^{(T)}}{T}$. Intuitively, we could achieve optimal solution T_{opt}, when the difference between the "exact" saved number of queries and the estimated saved number of queries, i.e., $C * T_{opt}$, is maximized. The formulation is shown as follows.

$$\max_{t=1,\ldots,T} \frac{I^{(1)} - I^{(t)}}{I^{(1)} - I^{(T)}} \cdot \left(RS^{(1)} - RS^{(T)} \right) - C \cdot t \tag{15}$$

4.2 Query distribution

In this subsection, we discuss a more general problem: query distribution. In real-world applications, it could be the case that the updated graphs come as streams, and we need to allocate our query budget among multiple time steps. Hence, we need a method to allocate the queries properly among different time steps and enable us to find the minority class examples with the minimum query budget and time.

To further explore this problem, we generate a synthetic data set containing two classes, in which the initial proportion of the minority class equals to 0.1 %. We increase the proportion of the minority class by 1 % in each time step. In Fig. 3, each point $(Q; T)$ represents the minimum required budget Q for identifying the minority class by time step T, and the budget is evenly allocated from time step 1 to time step T. From this figure, we have 3 observations: (i) if we need to finish the task by time step 1, then the largest number of queries is required; (ii) if we only need to finish the

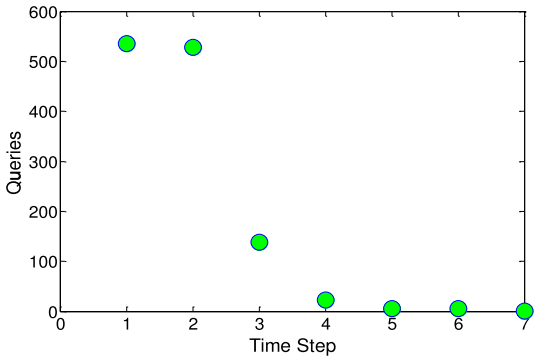

Fig. 3 Query allocation

task by the last time step, the smallest number of queries is required. (iii) the point at time step 3 is likely to hold a good trade-off, which has a relatively low querying number and early detection time.

To further study the query dynamics problem, we propose 5 potential strategies for the query distribution problem:

- *S1* Allocate all the budget at the first time step.
- *S2* Allocate all the budget at the last time step.
- *S3* Allocate all the budget into time step T_{opt}.
- *S4* Allocate the query budget evenly among different time steps.
- *S5* Allocate the query budget into different time steps following exponential distribution, such as $e^{-\alpha t}$, where $\alpha > 0$.

For query distribution problem, we propose Algorithm 4. Different from the query process of Algorithm 2, in Step 3, we need to apply a strategy to calculate the certain budget $B^{(t)}$ for time step t. If we have not found the minority class within $B^{(t)}$ at time step t, then we go to the next time step. The overall algorithm stops either when the minority class is discovered or when there is no budget to use.

We compare the performance of these five strategies with both synthetic data sets and real data sets in Sect. 5.

5 Experiments

The analysis in Sects. 3 and 4 shows the advantage of our model in RCD on time-evolving graphs. In this section, we aim to empirically verify the effectiveness and the efficiency of the proposed algorithms on both synthetic data sets and real data sets.

5.1 Data sets and setup

Six time-evolving graph data sets are used for testing our proposed algorithms. Among these 6 data sets, there is 1 synthetic data set, 3 semi-real data sets and 2 real data sets. In Table 2, we list several statistics of each data set.

ALGORITHM 4: Query distribution algorithm

Input: Strategy S,$M^{(1)}$,$A^{(1)}$,$NN^{(1)}$,$\Delta S^{(2)}, \ldots, \Delta S^{(T)}$,$p^{(t)}$,$\alpha$.
Output: The set I of labeled nodes and the L of their labels
for t = 1:T **do**
  Let $K^{(t)} = \max_{l=c}^{C} n \times p_l^{(t)}$.
  Calculate $B^{(t)}$ as given Strategy S.
  Calculate $NN^{(t)}$ as described in Algorithm 2.
  while not all the classes have been discovered **do**
    Find the $(K^{(t)})$th element in each row of $NN^{(t)}$. Set a^c to be the largest value of them.
    Let $KNN^c(x_i, a^c) = \{x | NN^{(T)}(i, j) > a^c\}$, and $n_i^c = |KNN^c|$, where $i = 1, \ldots, n$ and $j = 1, \ldots, n$.
    for index = 1: $B^{(t)}$ **do**
      For each node x_i has been labeled y_i, if $A^{(T)} > a^{y_i}$, $score_j = -\infty$; else, let $score_i = \max_{A^{(T)}(i,j) > \frac{a^c}{index}} \left(n_i^c - n_j^c\right)$
      Select the nodes x with the largest score to labeling oracle.
      If the label of x is exact class c, break; else, mark the class that x belongs to as discovered.
    end for
  end while
  If all the minority classes are discovered, break.
end for

Table 2 Statistics of different data sets

Name	Instance	Time steps	Number of classes
Synthetic data	5000	6	2
Abalone	4177	6	20
Adult	48,842	6	2
Statlog	58,000	6	6
Epinion	5665	16	24
Twitter	16,149	5	6

The synthetic data set contains 5000 instances, and we assume the proportion of the minority class is increasing over time. Hence, to generate the time-evolving graphs in later time steps, we let the proportion of a certain minority class increase by 1 % and simultaneously let the proportion of the majority class decrease by 1 % at each time step. Meanwhile, we generate additional 6 time-evolving graphs for 6 more time steps.

The Abalone data set comes from a biology study. In this data set, we need to predict the age of abalone based on multiple features. The age varies from 1 to 29, which roughly forms a normal distribution. Specifically, there are very few examples lying in the two extreme sides of the distribution. We separate the Abalone data set into 5 classes, i.e., one majority class and 4 minority classes. The proportion of the majority class is 56.93 %, and the proportion of the smallest class is 0.4 %. Besides, we choose the minority class with the smallest prior to evolve over time.

The Adult data set comes from a demographic census, which aims to predict whether the income of people exceeds $50 K per year or not. In Adult data set, there are 48,842

examples containing one majority class and one minority class. The majority class is the population of income below \$50 K, and the minority class is the population of income above \$50 K. In this data set, around 24 % of examples belong to the minority class. Since we stand on the problem of the RCD, we keep the majority class the same and only take 500 examples from the minority class. In this way, we can generate 24 data sets with the minority priors of 1.3 %. Notice that all the experimental results for the Adult data set are calculated from the average values of the 24 sub-data sets.

The Statlog data set comes from a shuttle schedule database, which consists of 58,000 examples and 7 classes. However, we only include the 6 largest classes in our evaluation, because the smallest class only contains 13 examples. After this modification, the priors of the 5 minority classes vary from 0.04 to 15 %. Same as before, we incrementally increase the proportion of the smallest minority class by 1 % in each time step.

The Epinion data set is a collection of about 5665 instances and 10,382 features over 16 time steps crawled from Epinion.com. Epinions is a product review site, where users can share their reviews about products. Users themselves can also build trust networks to seek advice from others. In this data set, each product is an instance, and the features for each product are formed by the bag-of-words model upon its reviews. In addition, the smallest class in Epinion only consists 0.03 % vertices while the proportion of the largest class is 17.56 %.

The Twitter data set is crawled from Twitter streaming API based on a set of terrorism related keywords, such as shoot, kidnap, blast and etc.. We include 16,149 English-speaking twitter users from 6 countries and around 10 millions tweets from 4/25/2015 to 5/5/2015. Then, we extract 209 features based on users' profiles, sentiments analysis, topic model analysis and users' ego-network analysis. In this data set, there are 56 % of users from Turkey, 0.09 % from Syria, 0.3 % from Iraq, 1.3 % from Iran, 36 % from Saudi Arabia and 5.8 % from Yemen. We separate the users into 6 classes based on their nationalities and generate a time-evolving graph in each 2-day interval.

5.2 Performance evaluation

First of all, we demonstrate the effectiveness upon 1000 synthetic data sets and 3 semi-synthetic data sets. We generate 1000 synthetic data sets, and each of them contains 5,000 examples belonging to two classes. Besides, we initialize the priors of the minority classes as 1 % and increase these priors by 1 % at each time step. We also make use of 3 real data sets which meet the scenario of RCD. The details of these 3 real data sets are summarized in Table 2. Then, we synthesize additional 6 time-evolving graphs from time step 2 to time step 7. For these time-evolving graphs, we let the proportion of a certain minority class increase by 1 % and simultaneously let the proportion of the majority class decrease by 1 % at each time step. Fig. 4a shows the comparison results of 4 different methods: random sampling (RS), *BIRD*, *BIRD*-LI and GRADE. Notice that *BIRD* and *BIRD*-LI perform the query process upon the approximate aggregated adjacency matrix, while GRADE is performed on the exact adjacency matrix at each time step. Besides, we provide *BIRD*-LI with a much looser

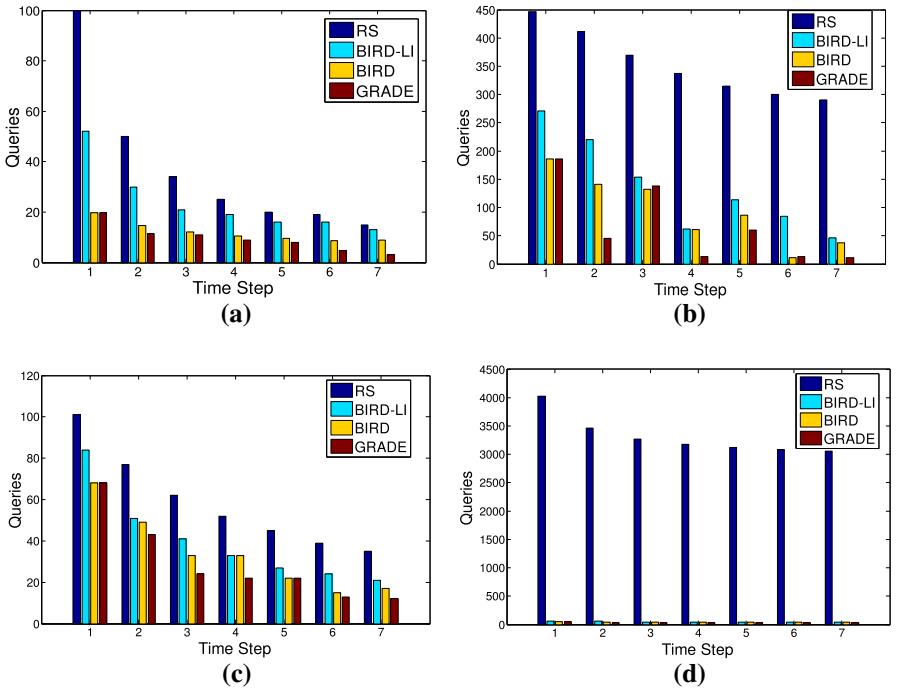

Fig. 4 Performance on synthetic and semi-synthetic data sets. **a** Synthetic data. **b** Abalone. **c** Adult. **d** Statlog

prior upper bound, e.g., we input 5 % as the upper bound instead of using the exact prior of 1 %. Then, we perform the same comparison experiments on 3 semi-synthetic data sets, which are shown in Fig. 4b–d. At last, we evaluate our algorithms on two real data sets in Figs. 5 and 6. Different from the previous cases, the proportions of the minority classes vary randomly instead of increasing over time. In general, we have the following observations: (i) both *BIRD* and *BIRD*-Li outperform random sampling in any conditions; (ii) all of these 4 methods perform better when the prior of minority class is getting larger; (iii) *BIRD* gives a comparable performance as GRADE does; (iv) *BIRD*-LI is quite robust and requires only a few more queries than BIRD does in most cases.

5.3 Efficiency of batch update

We run the experiments with Matlab 2014a on a workstation with CPU 3.5 GHz 4 processors, 256 GB memory and 2 T disk space. For both *BIRD* and GRADE, the most time-consuming step is updating the global similarity matrix $A^{(t)}$ and neighbor information matrix $NN^{(t)}$ at each time step. In this subsection, we report the running time of updating $A^{(t)}$ and $NN^{(t)}$ from an initial time step to the second time step. To better visualize the performance, we run the experiment on an increasing size of graph, i.e., from 500 examples in graph to 1000 examples in graph. For each certain

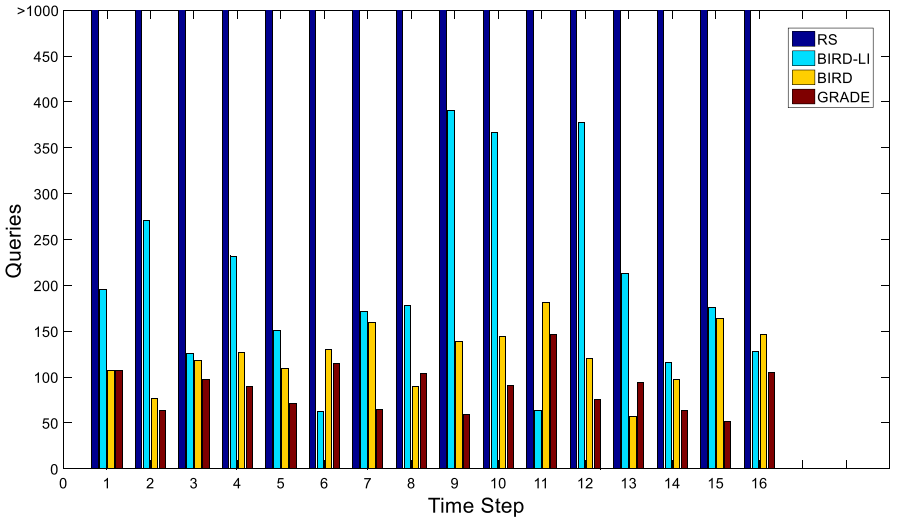

Fig. 5 Performance on epinion data set

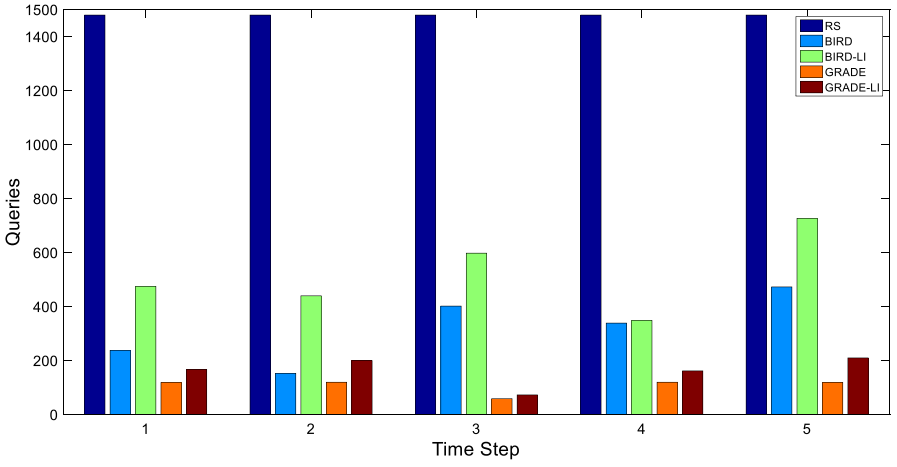

Fig. 6 Performance on twitter data set

size, we have 100 identical-setting data sets. Each point in Fig. 7 is computed based on the average value of the 100 data sets under identical settings. As we mentioned before, the computation cost of GRADE is $O(n^3)$, and our method only costs $O(n^2)$. From Fig. 7, we can see the difference of running time is gradually increasing over time. The difference is limited when the number of examples is 500. However, when the size of graph goes to 10,000, the running time of BIRD is 6.227 seconds, while the running time of GRADE is 41.41 seconds, which is 7 times of BIRD. Moreover, the difference would be even more significant when we run algorithms on a series of time steps.

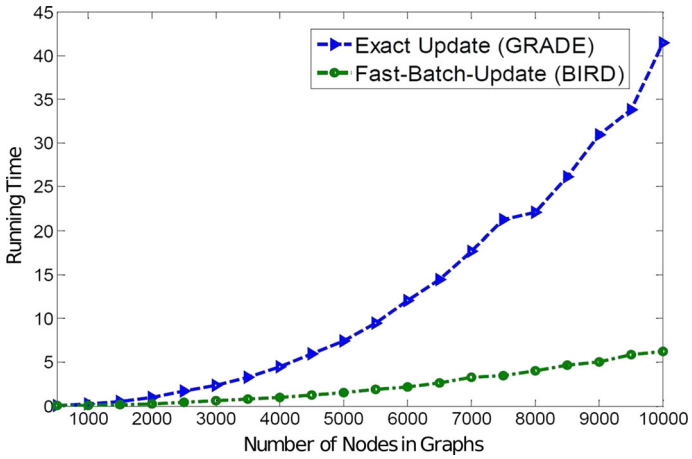

Fig. 7 Efficiency

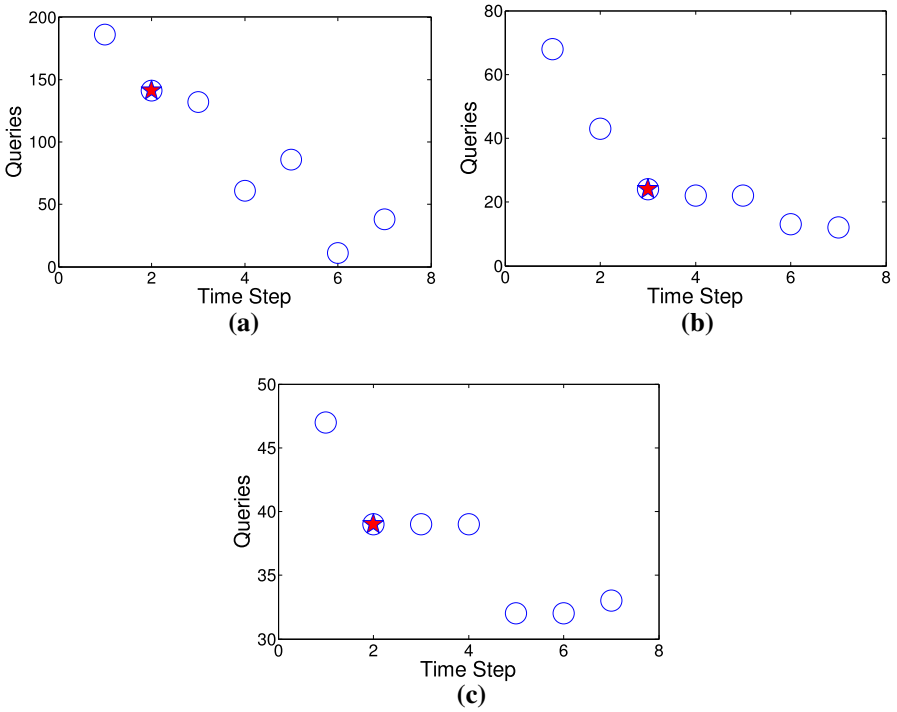

Fig. 8 Query locating. **a** Abalone. **b** Adult. **c** Statlog

5.4 Query dynamics

In this subsection, we show the performance of query locating and query distribution. In Fig. 8, we apply the query locating methods on 3 real data sets. As the proportion

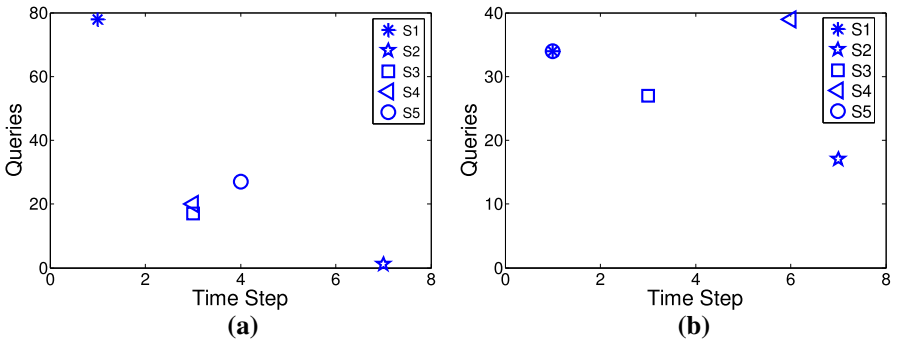

Fig. 9 Query distribution. **a** Synthetic data set. **b** Real data set (adult)

is increasing over time, the labeling request is decreasing in general. Besides, we also observe that T_{opt} is always located at the left bottom of each graph, which meets our ALAP and AEAP intuitions.

Furthermore, by applying Algorithm 4, we perform the results of 5 different strategies on one binary-class synthetic data set and one binary-class real data set, i.e., Adult. In both Fig. 9a, b, we observe that Strategy $S1$ is always located at the left top of the figure, which holds the time optimal; Strategy $S2$ is always located at the right bottom of the figure, which holds the budget optimal; Strategy $S3$ is always located at the left bottom of the figure, which considers both the time and the budget factors. All of these 3 observations are consistent with our intuitions.

Besides, we also find two interesting observations. The first one is that, in Fig. 9a, Strategy $S4$ performs slightly better than Strategy $S5$, while Strategy $S5$ outperforms Strategy $S4$ in Fig. 9b. The reason is as follows. Strategy $S5$ always allocates most of the budget at the earliest few time steps, which is why Strategy $S5$ could identify minority class examples at time step 1 in Fig. 9b. But, if Strategy $S5$ cannot discover the minority class at the beginning, it will finish the task later than Strategy $S4$, which is why $S5$ performs worse than $S4$ in Fig. 9a. Strategy $S4$ allocates its budget evenly among each time steps. However, when the task is relatively tough at the beginning few time steps and relatively easy at the end, Strategy $S4$ may fail. This is what is happening in Fig. 9b. Another interesting observation is that, in Fig. 9b, Strategy $S3$ only queries 27 examples at time step 3 for discovering the minority class, while Strategy $S4$ needs 39 labeling requests. Since the graph is evolving over time, Strategy $S4$ may automatically skip some minority-class examples when these examples are pretty similar to the previous labeled examples, which is the reason why Strategy $S4$ requires more queries.

6 Conclusion

In this paper, we mainly focus on the problem of efficiently and incrementally identifying under-represented rare category examples from time-evolving graphs. We propose two fast incremental updating algorithms, i.e., *BIRD* and *SIRD*, as well as a generalized version of *BIRD* named *BIRD*-LI to handle the problems where the exact priors

of the minority classes are unknown. Furthermore, based on our algorithms, we introduce five strategies to deal with the novel problem—query distribution. To the best of our knowledge, this is the first attempt in RCD under these dynamic settings. The comparison experiments with state-of-the-art methods demonstrate the effectiveness and the efficiency of the proposed algorithms.

Acknowledgements This work is supported by NSF research Grant IIS-1552654, and an IBM Faculty Award. The views and conclusions are those of the authors and should not be interpreted as representing the official policies of the funding agencies or the U.S. Government.

References

Aggarwal CC, Philip SY (2010) On clustering massive text and categorical data streams. Knowl Inf Syst 24(2):171–196

Akoglu L, McGlohon M, Faloutsos C (2010) Oddball: spotting anomalies in weighted graphs. In: Pacific-Asia conference on knowledge discovery and data mining, Springer, New York, pp 410–421

Akoglu L, Khandekar R, Kumar V, Parthasarathy S, Rajan D, Wu KL (2014) Fast nearest neighbor search on large time-evolving graphs. In: Joint European conference on machine learning and knowledge discovery in databases, Springer, New York, pp 17–33

Backstrom L, Huttenlocher D, Kleinberg J, Lan X (2006) Group formation in large social networks: membership, growth, and evolution. In: ACM SIGKDD international conference on knowledge discovery and data mining, ACM, New York, pp 44–54

Berlingerio M, Koutra D, Eliassi-Rad T, Faloutsos C (2012) Netsimile: a scalable approach to size-independent network similarity. In: arXiv preprint arXiv:1209.2684

Bettencourt LM, Hagberg AA, Larkey LB (2007) Separating the wheat from the chaff: practical anomaly detection schemes in ecological applications of distributed sensor networks. In: Distributed computing in sensor systems, Springer, New York, pp 223–239

Dasgupta S, Hsu D (2008) Hierarchical sampling for active learning. In: International conference on machine learning, ACM, New York, pp 208–215

Davis M, Liu W, Miller P, Redpath G (2011) Detecting anomalies in graphs with numeric labels. In: ACM international conference on information and knowledge management, ACM, New York, pp 1197–1202

Eberle W, Graves J, Holder L (2010) Insider threat detection using a graph-based approach. J Appl Secur Res 6(1):32–81

Fan W, Wang X, Wu Y (2013) Incremental graph pattern matching. ACM Trans Database Syst 38(3):18

Franke C, Gertz M (2008) Detection and exploration of outlier regions in sensor data streams. In: IEEE international conference on data mining workshops, IEEE, Los Alamitos, pp 375–384

Gao J, Liang F, Fan W, Wang C, Sun Y, Han J (2010) On community outliers and their efficient detection in information networks. In: ACM SIGKDD international conference on knowledge discovery and data mining, ACM, New York, pp 813–822

Gupta M, Gao J, Aggarwal C, Han J (2014) Outlier detection for temporal data. Synth Lect Data Min Knowl Discov 5(1):1–129

Gupte M, Eliassi-Rad T (2012) Measuring tie strength in implicit social networks. In: Annual ACM web science conference, ACM, New York, pp 109–118

He J, Carbonell JG (2007) Nearest-neighbor-based active learning for rare category detection. In: Advances in neural information processing systems, pp 633–640

He J, Liu Y, Lawrence R (2008) Graph-based rare category detection. In: IEEE international conference on data mining, IEEE, pp 833–838

He J, Tong H, Carbonell J (2010) Rare category characterization. In: IEEE international conference on data mining, IEEE, pp 226–235

Henderson K, Eliassi-Rad T, Faloutsos C, Akoglu L, Li L, Maruhashi K, Prakash BA, Tong H (2010) Metric forensics: a multi-level approach for mining volatile graphs. In: ACM SIGKDD international conference on knowledge discovery and data mining, ACM, New York, pp 163–172

Hill DJ, Minsker BS, Amir E (2007) Real-time bayesian anomaly detection for environmental sensor data. In: Congress-international association for hydraulic research, Citeseer, vol 32, p 503

Kang U, McGlohon M, Akoglu L, Faloutsos C (2010) Patterns on the connected components of terabyte-scale graphs. In: IEEE international conference on data mining, IEEE, pp 875–880
Kang U, Tsourakakis CE, Appel AP, Faloutsos C, Leskovec J (2011) Hadi: mining radii of large graphs. ACM Trans Knowl Discov Data 5(2):8
Koutra D, Ke TY, Kang U, Chau DHP, Pao HKK, Faloutsos C (2011) Unifying guilt-by-association approaches: theorems and fast algorithms. In: Joint European conference on machine learning and knowledge discovery in databases, Springer, New York, pp 245–260
Koutra D, Papalexakis EE, Faloutsos C (2012) Tensorsplat: spotting latent anomalies in time. In: Panhellenic conference on informatics, IEEE, pp 144–149
Kumar R, Mahdian M, McGlohon M (2010) Dynamics of conversations. In: ACM SIGKDD international conference on knowledge discovery and data mining, ACM, pp 553–562
Leskovec J, Kleinberg J, Faloutsos C (2005) Graphs over time: densification laws, shrinking diameters and possible explanations. In: ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, pp 177–187
Liu Z, Chiew K, He Q, Huang H, Huang B (2014) Prior-free rare category detection: more effective and efficient solutions. Expert Syst Appl 41(17):7691–7706
Müller E, Sánchez PI, Mülle Y, Böhm K (2013) Ranking outlier nodes in subspaces of attributed graphs. In: IEEE international conference on data engineering workshops, IEEE, pp 216–222
Pelleg D, Moore AW (2004) Active learning for anomaly and rare-category detection. In: Advances in neural information processing systems, pp 1073–1080
Phua C, Lee V, Smith K, Gayler R (2010) A comprehensive survey of data mining-based fraud detection research. arXiv preprint arXiv:10096119
Sherman J, Morrison WJ (1950) Adjustment of an inverse matrix corresponding to a change in one element of a given matrix. Annals Math Stat 21(1):124–127
Sricharan K, Das K (2014) Localizing anomalous changes in time-evolving graphs. In: ACM SIGMOD international conference on management of data, ACM, pp 1347–1358
Tong H, Papadimitriou S, Philip SY, Faloutsos C (2008) Proximity tracking on time-evolving bipartite graphs. In: SIAM international conference in data mining, pp 704–715
Yamanishi K, Takeuchi Ji (2002) A unifying framework for detecting outliers and change points from non-stationary time series data. In: ACM SIGKDD international conference on knowledge discovery and data mining, ACM, pp 676–681
Yamanishi K, Takeuchi JI, Williams G, Milne P (2004) On-line unsupervised outlier detection using finite mixtures with discounting learning algorithms. Data Min Knowl Discov 8(3):275–300
Zhou D, He J, Candan K, Davulcu H (2015a) Muvir: Multi-view rare category detection. In: International joint conference on artificial intelligence, pp 4098–4104
Zhou D, Wang K, Cao N, He J (2015b) Rare category detection on time-evolving graphs. In: IEEE international conference on data mining, IEEE, pp 1135–1140