2018

Sampling dark networks to locate people of interest

Wijegunawardana, Pivithuru; Ojha, Vatsal; Gera, Ralucca; Soundarajan, Sucheta

Monterey, California. Naval Postgraduate School

Wijegunawardana, Pivithuru, et al. "Sampling dark networks to locate people of interest." Social Network Analysis and Mining 8.1 (2018): 15.
http://hdl.handle.net/10945/63680

Social Network Analysis and Mining (2018) 8:15
https://doi.org/10.1007/s13278-018-0487-0

ORIGINAL ARTICLE

Sampling dark networks to locate people of interest

Pivithuru Wijegunawardana[1] · Vatsal Ojha[2] · Ralucca Gera[3] · Sucheta Soundarajan[1]

Received: 2 October 2017 / Revised: 16 December 2017 / Accepted: 24 January 2018 / Published online: 3 March 2018

Abstract
Dark networks, which describe networks with covert entities and connections such as those representing illegal activities, are of great interest to intelligence analysts. However, before studying such a network, one must first collect appropriate network data. Collecting accurate network data in such a setting is a challenging task, as data collectors will make inferences, which may be incorrect, based on available intelligence data, which may itself be misleading. In this paper, we consider the problem of how to effectively sample dark networks, in which sampling queries may return incorrect information, with the specific goal of locating people of interest. We present RedLearn and RedLearnRS, two algorithms for crawling dark networks with the goal of maximizing the identification of nodes of interest, given a limited sampling budget. RedLearn assumes that a query on a node can accurately return whether a node represents a person of interest, while RedLearnRS dispenses with that assumption. We consider realistic error scenarios, which describe how individuals in a dark network may attempt to conceal their connections. We evaluate and present results on several real-world networks, including dark networks, as well as various synthetic dark network structures proposed in the criminology literature. Our analysis shows that RedLearn and RedLearnRS meet or outperform other sampling strategies.

Keywords Sampling · Lying scenarios · Nodes of interest · Dark networks

1 Introduction

Complex network analysis has emerged as a key component of a wide variety of scientific fields, but the quality of the network analysis depends on the quality of the underlying data. In certain applications, such as those involving analysis of so-called dark networks representing illegal, covert, or undisclosed activities (Raab and Milward 2003), individuals within the network may purposefully conceal their network data with the goal of misleading the data collector or analyst.

The purpose of this work is to develop algorithms for sampling dark networks with the intention of locating as many "persons of interest" (POI) as possible in the network given a limited query budget. Here, a POI is a node possessing a certain attribute (e.g., individuals involved in a particular criminal action). We assume that we begin with knowledge of one POI in the network, with the rest of the network unobserved (both in terms of topology as well as node attributes).

A major complicating factor in this problem is that due to the covert nature of the networks being studied, one cannot expect the observed information to be reliable. When exploring a dark network, nodes may deliberately provide misinformation about themselves and the network structure (e.g., lie to an investigator), or the analyst herself may need to draw conclusions from multiple sources of information, with possible errors. This is in contrast to traditional sampling algorithms, which generally assume that the information observed is correct at the time of the query. There are thus major challenges in designing algorithms to sample dark networks: first and foremost, in order to select the next query,

✉ Pivithuru Wijegunawardana
ppwijegu@syr.edu

Vatsal Ojha
vojha@andrew.cmu.edu

Ralucca Gera
rgera@nps.edu

Sucheta Soundarajan
susounda@syr.edu

[1] Department of Electrical Engineering and Computer Science, Syracuse University, Syracuse, NY, USA

[2] Science and Humanities Scholars Program, Carnegie Mellon University, Pittsburgh, PA, USA

[3] Department of Applied Mathematics, Naval Postgraduate School, Monterey, CA, USA

one must draw inferences about the network structure from inaccurate data. Even for already-observed nodes, when predicting whether a node is or is not a POI, one can only use information observed so far; but because this information may be incorrect, one may come to a false conclusion, which in turn affects predictions for other nodes, potentially leading to a cascade of errors.

To our knowledge, we are the first to consider the problem of sampling with the goal of identifying nodes of interest in a setting with misinformation.

As an example, one of the networks we consider contains data depicting relationships among terrorists in the "Noordin Top" network in Indonesia. In this network, edges represent that two nodes belong to the same organization, attended the same school or training, have a kinship relationship, and so on.[1] In such a network, the POIs may be those individuals who have an attribute of interest to the analyst, such as involvement in a terrorist attack.

In this paper, querying a node refers to collecting data about the node, node's neighbors, whether these neighbors are POIs and determining whether the queried node is a POI itself. For example, querying a node can refer to a data analyst obtaining information about the node and determining whether it is a POI. The role of the sampling algorithm is to suggest a query on the node that is most likely to be a POI.

We consider two sampling settings, corresponding to different types of data collection processes. In the first setting, when a node is selected for query , then (1) the data collector accurately discovers whether or not that node is a POI, and (2) the data collector, possibly inaccurately, determines the identities of node's neighbors and whether those neighbors are POIs. In the second setting, we remove the assumption that the data collector can accurately discover a node's status as a POI: Instead, the query may misreport whether or not the node itself is a POI. In both settings, there may be errors in determining the queried node's neighbors and whether those neighbors are POIs: for example, a POI node may encode communications with other POI nodes, or try to make these communications look innocent. The first setting corresponds to cases in which it is easy to determine whether a queried node is of interest (e.g., comparing fingerprints to those found at a crime scene). The second setting corresponds to cases in which the determination of whether a node is a POI is difficult and may be done incorrectly (for example, by an analyst aggregating several sources of conflicting information).

We present two sampling algorithms, RedLearn and RedLearnRS. RedLearn operates under the assumption that the analyst can accurately determine the queried node's status. For the second sampling setting, we introduce RedLearnRS, a *re-sampling* algorithm that may repeatedly query the same node in order to update the accuracy of the information. We show that in cases where the POIs exhibit homophily (i.e., are likely to be connected to other POIs), a simple algorithm of choosing the node with the most POI neighbors works well. However, in the more realistic scenario where POIs hide their connections with other POIs, RedLearn and RedLearnRS show outstanding performance, improving over the best baseline algorithm by up to 340%.

This paper is organized as follows. In Sect. 2, we formally introduce the two problem settings we consider. In Sect. 3, we discuss related research work to our problem. Section 4 presents RedLearn and RedLearnRS algorithms to find nodes of interest. Sections 5 and 6 provide details of the datasets we use and experimental setup we use to evaluate monitor placement algorithms, respectively. In Sect. 7, we present performance of RedLearn and RedLearnRS algorithms compared to baseline POI finding algorithms.

2 Problem definition

We assume that there is an unobserved, undirected, unweighted graph $G = (V, E)$, in which each node $v \in V$ has a color attribute, $c_v \in \{red, blue\}$. Red nodes represent the persons of interest (POIs) and blue nodes represent all others.

We begin with the realistic assumption that a red node was identified in G (e.g., arrested while committing a crime). We are given a budget b of queries, where each query is conducted by analyzing data corresponding to a node. We call this process placing a *monitor* on a node.[2] In each step, we query an observed node v, regardless of its color, and the monitor reports (1) the suspected color of the node (s_v), (2) the neighbors of the node ($N(v)$), and (3) the color e_u of each of node v's reported neighbors u. As monitors represent humans performing data analysis, some or all of this information may be reported incorrectly by the monitor, due to analyst error or judgment.

There are four possible types of errors when retrieving node color and network structure information using monitors: (1) A *node color error*, where the suspected color of the node is not the true color ($s_v \neq c_v$) (2) an *edge existence error*, where the monitor fails to report some neighbors of the node, (3) an *edge nonexistence error*, where the monitor reports false edges between nodes and (4) a *neighbor relationship color error*, where the reported colors of the

[1] The data were collected by Roberts and Everton (2011) and compiled into a network by Gera et al. (2017).

[2] This terminology was motivated by an application in which hardware devices known as monitors are placed on computers to observe incoming and outgoing traffic.

queried node's neighbors are different from their true colors ($e_u \neq c_u$).

For example, placing a monitor on a suspected criminal node could represent an analyst conducting an investigation to determine whether this person is a criminal (node color). The investigation process includes looking through his email and phone contacts to observe his connections (neighbor identities), analyzing his emails and other messages with these contacts to determine whether these neighbors are themselves criminals (neighbor relationship colors). Naturally, there may be errors in this process due to nodes trying to hide/fabricate their connections, criminals using code words in their communications, general noisy communications among all nodes, and analysis errors that can occur.[3]

Suppose that after exhausting the monitor budget b, we have a set of monitored nodes $V_m = V_r \cup V_b$. Here, $V_r = \{v \in V_m : s_v = red\}$, the set of monitored nodes that are *suspected* to be red, and $V_b = \{v \in V_m : s_v = blue\}$, the set of monitored nodes that are *suspected* to be blue. The goal of the analysis is to maximize the true set of red nodes detected, namely $|\{v \in V_r : c_v = red\}|$.

We consider two problem settings depending on whether the analyst can conclusively determine someone is a criminal or not.

2.1 Problem Setting 1: node color reliable

In Problem Setting 1, we assume that a monitor accurately determines the color of the queried node. Therefore, $V_r = \{v \in V_m : s_v = red\} = \{v \in V_m : c_v = red\}$; all nodes that are suspected to be red are indeed red-colored nodes.

For example, suppose that there is an investigation to find a set of computers that are affected by some virus. The analyst has a hardware device which can detect whether a computer is affected or not. Whenever the analyst finds a potential-affected computer, she can place this device on the machine and determine the true status of the machine. In this example, placing the hardware equipment is equivalent to placing a monitor. Because the analyst has limited hardware equipment, she needs to carefully decide which computer to monitor next so that V_r is maximized.

2.2 Problem Setting 2: node color misreported

In Problem Setting 2, we consider the setting where a monitor may misreport the color of the monitored node. For example, suppose there is an investigation to find terrorists who were involved in a specific attack. The analyst has to determine whether someone is a terrorist solely based on his contacts, messages, and whereabouts. In the absence of conclusive evidence to prove that a person is a terrorist, the analyst has to make a judgment based on available information. This can lead to identifying a terrorist as not guilty if there is not enough evidence, and conversely may lead to identifying an innocent individual as guilty.

In this problem setting, placing a single monitor on a node may not reveal the true color of a node. In this case, we allow for repeated monitor placement to re-query previously monitored nodes. In the example above, repeated monitor placement would allow the analyst to gather more information about the criminal by considering other types of communication channels, spending more time and resources on the analysis which can lead to better judgment.

3 Related work

3.1 Criminal network analysis

A dark network is a social network representing illegal and covert activities, whose members are actively trying to conceal network information even at the expense of efficiency (Baker and Faulkner 1993; Raab and Milward 2003). Because dark networks are deceptive by nature, there are many errors involved in collecting and analyzing data from these networks.

Criminal networks are a common example of dark network (Baker and Faulkner 1993). Some researchers have used social network analysis techniques such as identifying key players, community detection, and network visualization techniques to identify leaders of criminal networks, identify subgroups in criminal networks, and visualize criminal networks, respectively (Chen et al. 2004; Lu et al. 2010; Koschade 2006). Some other researchers have considered using social network analysis to disrupt criminal networks (Sparrow (1991); Schwartz and Rouselle (2009)). These analysis techniques generally assume that the complete network structure information is available.

In this work, we introduce a methodology to identify POIs in dark networks while examining various error scenarios based on the terrorist actively trying conceal their true status and connections. Uncovering these networks amid concealed and misinformation require advanced network sampling techniques.

3.2 Network sampling algorithms

In general, network sampling algorithms can be divided into two categories: downsampling algorithms and crawling-based sampling algorithms. Downsampling algorithms begin with knowledge of the entire graph, but due to computational issues (e.g., time or space), must find an appropriately sized

[3] We consider realistic types of errors to represent analysis errors; these are described in Sect. 6.5.

subgraph that is representative of the larger graph (Leskovec and Faloutsos (2006)). Crawling-based algorithms on the other hand start with knowledge of few nodes and explore the network through querying observed nodes. In our problem, we use crawling to explore the network structure by placing monitors.

There are a multitude of sampling techniques for crawling-based network exploration, including random walks (Asztalos and Toroczkai 2010; Hughes 1995; Noh and Rieger 2004), biased random walks (Fronczak and Fronczak (2009)), and walks combined with reversible Markov chains (Aldous and Fill (2002)), Bayesian methods (Friedman and Koller 2003), or standard exhaustive search algorithms like depth-first or breadth-first searches, such as Adamic et al. (2001), Biernacki and Waldorf (1981), Bliss et al. (2014), Davis et al. (2016), and Leskovec and Faloutsos (2006). However, these methods do not use all discovered information, such as node attributes of the discovered part of network, and are not designed for sampling networks in which queries return inaccurate information.

Various researchers have considered the problem of sampling for specific goals. For example, Avrachenkov et al. (2014) present an algorithm to sample the node with the highest estimated unobserved degree. Hanneke and Xing (2009) and Maiya and Berger-Wolf (2010) examine online sampling for centrality measures. Macskassy and Provost (2005) develop a guilt-by-association method to identify suspicious individuals in a partially known network.

Some researchers have considered the problem of sampling to locate targeted nodes. Bnaya et al. (2013) present a volatile multi-arm bandit-based algorithm to crawl for targeted nodes. Their approach tries to find a balance between exploring the network and exploiting targeted nodes at the time of crawling. However, this work does not address potential inaccuracies in the sampling process (and additionally, using an explore–exploit method for sampling dark networks would present significant ethical problems, as it would require investigating non-suspects for the sake of exploration). Stern et al. (2013) present the TONIC problem for finding targeted nodes and propose a method for finding nodes that can lead to the targeted nodes. Unlike our problem setting, these works do not consider the case where queried nodes can provide information about their neighbors, and also do not account for possibly inaccurate information and errors. Gera et al. (2017) consider identifying a target community from partial information, by taking a different approach one what part of the network has been discovered. They assumed that the information of one layer (one type of relationship) of network was known and tried to identify a group in the whole network.

3.3 Node classification algorithms

A natural approach for the problem posed in this paper is to use a node classification algorithm to predict whether observed nodes are red or blue. Traditional data classification algorithms assume that data instances are independent from each other. In contrast, node classification algorithms study the problem, how to conduct a classification in network or relational data. These algorithms work with partially labeled network data to predict labels for unlabeled data instances. Node classification algorithms generally achieve better performance by exploiting relational information between classification data instances.

There are many applications of node classification algorithms. Xiang et al. (2010) showed applications of collective classification in classifying gender, political views, religious views, and relationship status using Facebook data. Zheleva and Getoor (2009) showed that privacy in social media sites can be exploited with a few public accounts using collective classification. Burfoot et al. (2011) used node classification algorithms to improve performance of sentiment classification in congressional floor debates. Carvalho and Cohen (2005) showed that collective classification approach can be successfully used to classify whether an email has an action request based on sequence of emails in a thread.

There are two main categories of node classification algorithms (Bhagat et al. 2011). The first category uses a local classifier trained using known labels and network structure information to predict node labels. Iterative classification algorithm (ICA) (Neville and Jensen 2000) and structured logistic regression model are such examples, and both use local information and links (Lu and Getoor 2003). Gallagher et al. (2008) proposed to use dummy edges to allow information flow in networks and looking at second neighborhood of a node instead of first neighbors to improve ICA performance.

The second type of node classification algorithms is the label propagation-based algorithms (Zhu et al. 2003; Lin and Cohen 2010). These algorithms use random walks to learn a global labeling function. Label propagation-based algorithms do not require neighborhood-based features to predict labels. They tend to explore the inherent link structure. Both types of algorithms assume some type of association between nodes with same labels.

In our problem setting, we have incorporated node colors, potential colors of neighbors as well as link structure to identify POIs. Label propagation-based algorithms only explore the link structure, and so are not capable of using neighbor color information given by monitored nodes. ICA is the most widely used algorithm among local classifier-based node classification algorithms. ICA uses a traditional classification algorithm as the local classifier, which is trained using the labeled nodes in the network. The unlabeled nodes are labeled according to the predictions from this local classifier. ICA then iteratively recalculates features for nodes, using

these predicted labels, and predicts labels again. This process continues until the predicted labels converge. In Sect. 7.1, we present how ICA performs in finding POIs.

4 Proposed method

We propose algorithms RedLearn and RedLearnRS for Problem Settings 1 and 2, respectively. In Sect. 4.1, we present RedLearn, a novel re-sampling-based algorithm to find POIs, where the node colors reported by the queries are accurate. In Sect. 4.2, we present RedLearnRS, an extension of RedLearn for the more general Problem Setting 2, where the node colors returned by queries can be wrong. These two algorithms use features that include node color information, estimated neighbor color information, as well as network topology. Both of these algorithms operate under possible edge existence errors and neighbor color errors.

4.1 RedLearn: a learning-based monitor placement algorithm

We introduce RedLearn, a learning-based sampling algorithm for finding nodes of interest under Problem Setting 1 (i.e., the reported node color is the true node color). RedLearn uses nodes' structural features and stated attributes to predict labels for unmonitored nodes, as described in Table 1.

To understand the use of network structural features in predicting node labels, consider the following: Intuitively, if the original network displays node color homophily, the probability of node v being a red node is higher if it has many red neighbors. However, if the network displays low or even anti-homophily, using this measure will naturally result in poor performance.

For example, some monitor placement algorithms, such as selecting the node that was reported as red by the greatest number of monitors, depend on information retrieved from neighbors. The performance of such monitor placement criteria thus heavily depends on neighbor color errors and edge existence errors.

To overcome these dependencies, the proposed RedLearn algorithm models this as a two-class classification problem, but rather than predicting whether a node v is red or blue, we instead predict $P(c_v = R)$.

Features Table 1 describes the set of features used by RedLearn. There are two types of features: (*a*) network structure-based features (1, 2, 3, 4, 5) and (*b*) neighbor-reported color-based features (6, 7, 8, 9, 10, 11).

Network structure-based features are used to learn the patterns of connections between red nodes (e.g., homophily vs. anti-homophily), as previous work shows that algorithms perform differently based on the network studied (Wijegunawardana et al. 2017). Neighbor-reported color-based features are intended to learn the relationship between what a monitor reports about its neighbors' colors and the true colors of those neighbors. The second-hop features (4, 5) are used to better estimate the red nodes that do not show homophily (Gallagher et al. 2008).

Inferred probability of being red We formulate four different probabilities to measure how likely it is that an unmonitored node is red, based on the neighbor color estimates given by differently colored monitors. We use the observed neighbor color error information to calculate these probabilities (i.e., once we monitor a node and confirm its color, we can determine whether monitored neighbors of this node have misreported its color). These four probabilities provide us information about whether there is a general pattern of neighbor color misreporting behavior. For example, we may learn whether monitors placed on red-colored nodes tend to misreport neighbor colors more frequently. In Eq. 1, we find the proportion of truly red nodes, when other red nodes have estimated them to be red.

Table 1 Classification features used by RedLearn for a node v with neighbor set $N(v)$

	Feature	Description
(1)	Number of red neighbors	$\lvert\{u \in N(v) \lvert s_u = R\}\rvert$
(2)	Number of blue neighbors	$\lvert\{u \in N(v) \lvert s_u = B\}\rvert$
(3)	Number of red triangles if v is red	$\lvert\{u, w \in N(v) \lvert \quad (u \in N(w)) \wedge s_u = s_w = R\}\rvert$
(4)	Number of second-hop red neighbors	$\lvert\{w \in N(N(u)) \lvert s_w = R\}\rvert$
(5)	Number of second-hop blue neighbors	$\lvert\{w \in N(N(u)) \lvert s_w = B\}\rvert$
(6)	Number of neighbors estimate red	$\lvert\{u \in N(v) \lvert (e_{uv} = R)\}\rvert$
(6)	Number of neighbors estimate blue	$\lvert\{u \in N(v) \lvert (e_{uv} = B)\}\rvert$
(7)	Number of red neighbors estimate red	$\lvert\{u \in N(v) \lvert (e_{uv} = R) \wedge s_u = R\}\rvert$
(8)	Number of red neighbors estimate blue	$\lvert\{u \in N(v) \lvert (e_{uv} = B) \wedge s_u = R\}\rvert$
(9)	Number of blue neighbors estimate red	$\lvert\{u \in N(v) \lvert (e_{uv} = R) \wedge s_u = B\}\rvert$
(10)	Number of blue neighbors estimate blue	$\lvert\{u \in N(v) \lvert (e_{uv} = B) \wedge s_u = B\}\rvert$
(11)	Inferred probability of being red	$P^I(s_v = R)$

Here, c_v is the true color of v, s_v is the suspected color of v, and e_{uv} is u's estimate of v's color

Next, we present how we compute the probability of a node v being red/blue given the choices of that neighbor u is either red or blue.

$$P(s_v = R|(s_u = R) \wedge (e_{uv} = R)) = \frac{|\{u, v \in V_m|(s_u = R) \wedge (e_{uv} = R) \wedge (s_v = R)\}|}{|\{u, v \in V_m|(s_u = R) \wedge (e_{uv} = R)\}|},$$

$$P(s_v = R|(s_u = R) \wedge (e_{uv} = B)) = \frac{|\{u, v \in V_m|(s_u = R) \wedge (e_{uv} = B) \wedge (s_v = R)\}|}{|\{u, v \in V_m|(s_u = R) \wedge (e_{uv} = B)\}|},$$

$$P(s_v = R|(s_u = B) \wedge (e_{uv} = R)) = \frac{|\{u, v \in V_m|(s_u = B) \wedge (e_{uv} = R) \wedge (s_v = R)\}|}{|\{u, v \in V_m|(s_u = B) \wedge (e_{uv} = R)\}|},$$

$$P(s_v = R|(s_u = B) \wedge (e_{uv} = B)) = \frac{|\{u, v \in V_m|(s_u = B) \wedge (e_{uv} = B) \wedge (s_v = R)\}|}{|\{u, v \in V_m|(s_u = R) \wedge (e_{uv} = R)\}|}.$$

For each unmonitored node v, we infer the probability that it is red $P^I(s_v = R)$, based on colors of its monitored neighbors and color estimates these monitored neighbors have given about v, using Eq. 1:

$$P^I(s_v = R) = \frac{\sum_{u \in N(v)} P(s_v = R|(s_u = R/B) \wedge (e_{uv} = R/B))}{|N(v)|}. \quad (1)$$

Training data Suppose that we have placed n monitors so far. The training set then consists of the (correctly) reported colors of the n monitored nodes along with their respective feature values. We use each of the n networks obtained after placing each monitor to train the learning model, to determine where to place the $(n+1)^{st}$ monitor.

Classification algorithm RedLearn determines $P(c_v = R)$ for each unmonitored node v. Because it predicts a probability rather than a binary label, RedLearn uses a logistic regression classifier. Furthermore, because the learning model must be updated frequently, this classifier gives an added advantage of faster training.

Placing the next monitor (prediction) Given the placement of n monitors and deciding to place the $(n+1)^{st}$ monitor, RedLearn calculates a feature vectors for each unmonitored node and applies the classifier to these feature vectors, giving the probability that each unmonitored node is red. RedLearn selects the node with the highest probability for placing the next monitor. Algorithm 1 summarizes RedLearn.

4.2 RedLearn Re-sampling (RedLearnRS) algorithm

For the second problem setting, we introduce RedLearnRS by adapting RedLearn. In this problem setting, we assume that a monitor may be incorrect when reporting the color of the monitored node. For example, this setting corresponds to criminal investigations, where an analyst needs to make a judgment as to whether an individual is a criminal based on available information. RedLearnRS assumes that such judgment errors occur primarily on red nodes, who may report to the investigator that they are actually innocent blue nodes, while blue nodes are unlikely to claim that they are actually red.

Due to the potential for node color errors, RedLearnRS is a *re-sampling* algorithm, which may place monitors on already-monitored blue nodes in order to better identify their true colors. Re-sampling an already-monitored node indicates that the analyst spends more resources looking at other types of communication channels or obtaining additional evidence, which in turn can provide a different judgment of the node's color.

RedLearnRS uses a two-step cycle. In the first step, RedLearnRS identifies monitored nodes with likely node color errors, which are considered for re-query. Additionally, RedLearnRS removes these nodes from the training data to help establish better ground truth labels for training data. The re-sampling step is viewed as an outlier detection problem: Because only red nodes may report node color errors, all nodes with node color errors are among the set of suspected blue nodes and can be viewed as outliers within this set. Identified outliers are added back to prediction set, which now contains all observed unmonitored nodes and re-sampled monitored nodes.

The second step of RedLearnRS is similar to RedLearn. Once the outlier blue nodes are identified and moved to prediction set, the learning algorithm is applied to predict the probability of each node in the prediction set being red. The node with the highest probability of being red is selected as

Algorithm 1 Learning based monitor placement

```
procedure LEARNING(start, budget)
    S ← Graph
    S.add(start), S.add(N(start))                    ▷ Starting node and neighbors
    while budget>0 do
        V_m ← list of monitored nodes in S
        TrainingData ← feature vectors for u ∈ V_m
        Train classifier using TrainingData
        V_u ← list of not yet monitored nodes in S
        for v ∈ V_u do
            Get feature vector for v
            P(c_v = R) ← predict v's probability of red using learning model
        Choose node v with maximum P(c_v = R) from V_u
        budget ← (budget − 1)
        Use v as next monitor
```

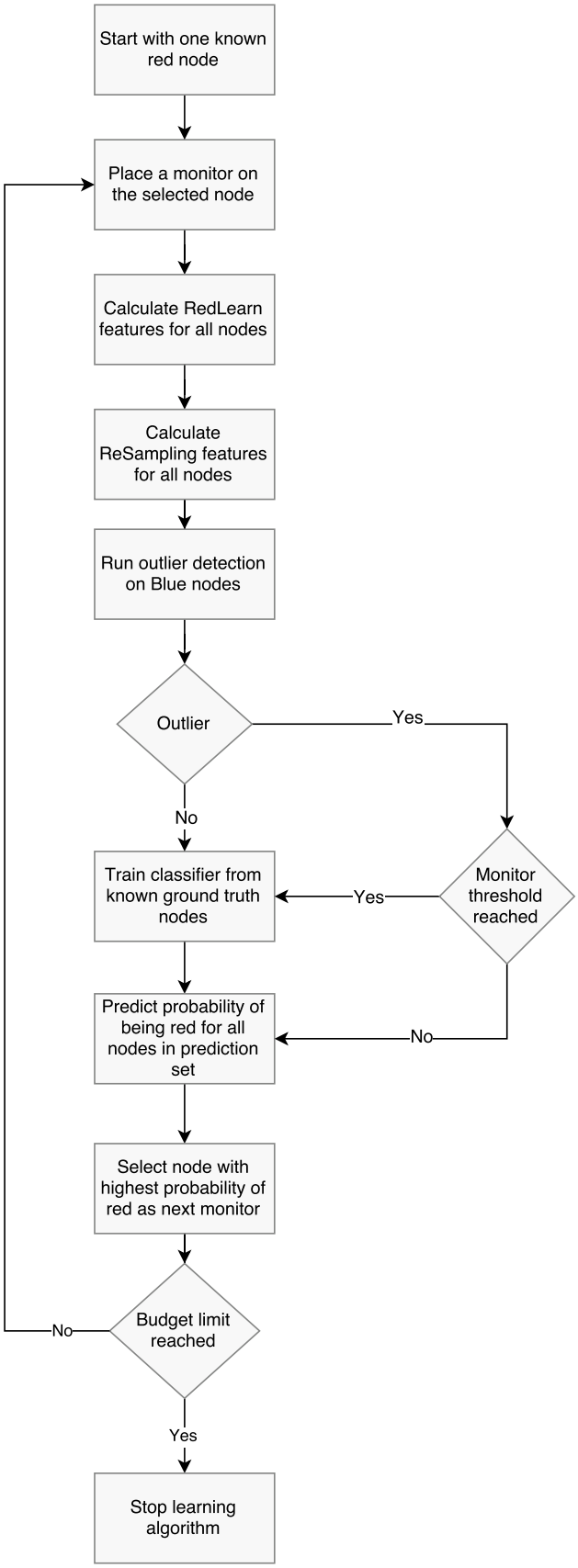

Fig. 1 RedLearnRS algorithm flow. This is a re-sampling-based algorithm to identify nodes of interest in a network where node colors reported by monitors may contain errors

the next node to be monitored. This node may be either an observed unmonitored node or a previously monitored outlier node. RedLearnRS's flow is depicted in Fig. 1.

New Features Table 2 shows the new features introduced to the learning algorithm to identify node color errors. The first feature is the number of used monitors. The more times a node has been monitored, while still being reported as blue, the more likely it is that it is actually blue. The next two features quantify whether the monitor placed on this node tends to report neighbor color errors more often as an indication concealing information.

Since RedLearnRS selects nodes among the set of reported blue nodes to re-monitor, there is a potential risk of monitoring innocent blue nodes repeatedly. Re-sampling does not necessarily mean that the monitored node will be arrested and interrogated, but rather indicates that an analyst takes further steps to identify true color of a node. However, we need to take measures to prevent infringing privacy of innocent nodes. In this regard, RedLearnRS limits the number of monitors that can be placed on a single node to prevent excessively monitoring the same node.

Repeated Monitor Threshold One can view this limit using optimal stopping criteria, which use prior probability information to calculate success or failure of placing another monitor.[4] However, in our case, we would need to repeatedly monitor blue nodes to estimate the priors required to calculate the optimal stopping point. Instead, RedLearnRS uses a simple numerical method to threshold the number of monitors placed on the same node as shown in Eq. 2.

$$\text{Monitor Threshold} = \max_{u \in V_r}(m_u + 1), \tag{2}$$

In Eq. 2, m_u represents the number of monitors placed on node u. Equation 2 increases the monitor threshold by one whenever the algorithm exceeds the current threshold number of monitors required to notice a node color change. If node color errors are less likely, we spend fewer monitors on each node, and vice versa if node errors are more common. We set the initial monitor threshold to 3 to avoid scenarios when all monitored red nodes may misreport their color in first round.

[4] The optimal stopping problem studies when to take an action in order to maximize some reward. The optimal stopping problem is applicable to many disciplines including stock trading (Tsitsiklis and Roy 1999), oil drilling (Benkherouf and Bather 1988), and determining when to stop a random walk (Novikov and Shiryaev 2005). The solutions to the optimal stopping problem generally make assumptions about prior probability distribution of success and failure. Since we have a limited budget of monitors, calculating these priors is not feasible.

Table 2 Additional features used by RedLearnRS for a node v with neighbor set $N(v)$

	Feature	Description
(1)	Number of monitors placed	m_v
(2)	Number of blue nodes lied about	$\lvert\{u \in N(v) \lvert (e_{vu} = R) \wedge s_u = B\}\rvert$
(3)	Number of red nodes lied about	$\lvert\{u \in N(v) \lvert (e_{vu} = B) \wedge s_u = R\}\rvert$

Here, s_u represents the suspected color of u and e_{vu} represents the node color that a monitor on node v reports u to be

Table 3 Summary of datasets and node attributes we have used to assign node colors

Dataset	Nodes	Edges	Attribute	Red Nodes	No. of Red Nodes	Color Assort.
NoordinTop	139	1042	Communi.	Type1	9	0.23
				Type2	11	0.62
				Type3	9	0.11
				Type4	18	0.63
				Type5	5	0.06
Pokec	26059	241514	Age	Age=28-32	1725	0.11
			Height	Height 160cm	1663	0.06
Amherst46	2235	90954	Year	Year=2008	380	0.58
				Year=2009	377	0.85
				Year=2007	363	0.34
			Major	Major=99	200	0.04
				Major=100	174	0.03
				Major=114	161	0.03
Colgate88	3482	155043	Year	Year=2008	641	0.63
				Year=2009	641	0.86
				Year=2007	589	0.47
			Major	Major=277	283	0.04
				Major=247	279	0.03
				Major=249	253	0.05

5 Datasets

5.1 Noordin Top network

The first network studied is the Noordin Top dataset, a real terrorist network with 139 nodes and 1042 edges ("Noordin Top" is the name of the leader of this network) (Gera et al. 2017). Its edges depict several types of relationships, including familial relationships, communications, and co-attendance at meetings.[5] In this network, every node is a terrorist, and we are interested in identifying those individuals that communicate using a certain medium. We consider 5 versions of this network. In NoordinComs1, the POI (red nodes) are those communicating using a general computer medium; in NoordinComs2, the red nodes are those who communicate using print media; in NoordinComs3, the red nodes are those who communicate using support materials; in NoordinComs4, the red nodes are those who communicate using unknown media; and in NoordinComs5, the red nodes are those who communicate using video (Table 3).

5.2 Pokec network

The Pokec network is part of a Slovakian online social network.[6] The nodes in the network are users of the social network and edges depict friendship relations. Each node has some number of associated user attributes (e.g., age, region, gender, interests, and height). We use a sample of this network containing all nodes in the region "kosicky kraj, michalovce" and edges among them. This sampled network contains 26, 220 nodes and 241, 600 edges. Although this is not a dark network dataset, we include it for purposes of performing more comprehensive experiments and to

[5] Obtained from https://sites.google.com/site/sfeverton18/research/appendix-1.

[6] Obtained from http://snap.stanford.edu/data/.

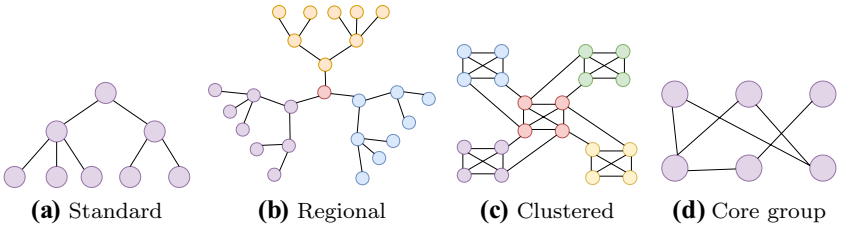

Fig. 2 Criminal network typologies described by Le (2012). **a** Standard hierarchy with a chain of commands. **b** Regional hierarchy with multiple chains of commands and a central controlling body. **c** Clustered hierarchy with tight knit groups. **d** Core group hierarchy, which doesn't exhibit clear structure

evaluate how monitor placement algorithms perform on larger networks.

We assign node colors based on two different node attributes (Table 3): *age* (a node with age in the range 28–32 is marked red, and blue otherwise, giving 1736 red nodes) and *height* (a user of height less than 160 *cm* is marked red, giving 1668 red nodes).

5.3 Facebook100 networks

Facebook100 is a collection of early Facebook networks, from when Facebook was only available to college students, and each college had its own network.[7] This is a collection of attributed social networks, containing node attributes such as gender, student major, year of matriculation, and dormitory of residence. In our experiments, we use select Major and Year attributes to assign node colors. Major has low attribute assortivity (homophily), whereas Year has high assortivity in these networks, and so we can evaluate the performance of monitor placement algorithms in different attribute homophily scenarios.

For each attribute, the top three most frequent (non-null) attribute values are selected to define red–blue nodes. For example, if we consider the year attribute, one set of red nodes would be all nodes that joined in year 2008 (while all others are blue). This results in six different red node and blue node assignments for each Facebook100 network. We have considered two Facebook100 college networks for our analysis resulting 12 different red-–blue node assignments altogether. Table 3 shows selected attribute values and number of red nodes in each network.

5.4 Synthetic terrorist networks

To simulate dark networks embedded into civilian networks, we generate synthetic terrorist network structures corresponding to the four criminal network typologies described by Le (2012), which we then embed into larger real social networks. We label the nodes in the criminal networks as red, and the nodes in the larger social networks as blue. Through experiments on these networks, we gain a better understanding of how the various monitor placement algorithms perform when locating individuals from different structural categories of criminal networks. Le describes these typologies in informal terms, but they can be directly mapped to concrete network structures. The network structures that we consider and the processes that we used to generate them are described below:

- **Standard:** Standard criminal networks follow the chain of command and can be modeled using a tree structure as shown in Fig. 2a.

  Network Generation We generate this network as a random power-law tree with power-law exponent 3[8].
- **Regional:** Regional criminal networks contain regional cells, each of which has a clear chain of command, with all regional leaders reporting to a central node. This network structure is similar to a collection of trees, where all root nodes are connected to the same central node. This network structure is shown in Fig. 2b.

  Network Generation We generate this network as a set of four random power-law trees. We create a single central node and connect that node to a randomly selected root node from each of the regional trees.
- **Clustered:** Clustered criminal networks contain multiple tightly knitted groups of criminals, with a central coordination body. An example clustered network structure is shown in Fig. 2c.

  Network Generation We generate this network as a set of five clusters of nodes, where each cluster is generated using a power-law cluster generating algorithm (Holme

[7] Obtained from https://archive.org/download/oxford-2005-facebook-matrix.

[8] Available at https://networkx.github.io/documentation/networkx-1.10/reference/generated/networkx.generators.random_graphs.random_powerlaw_tree.html#networkx.generators.random_graphs.random_powerlaw_tree.

and Kim 2002)[9]. A node is connected to 80% of the same cluster. We create a triangle with probability 0.8 whenever a new node is added. Nodes from non-coordinating clusters are connected to the coordinating cluster nodes randomly with 0.05 probability.

– **Core group:** Core group networks are unstructured, with a loose, flat hierarchy. An example of core group structure is shown in Fig. 2d.

Network Generation We use an Erdős–Rényi random network with edge probability 0.1 to generate this structure (Erdos and Rényi 1960)[10].

To simulate a criminal network hidden within a larger social network, we embed these synthetic networks into the Facebook100-Amherst41 network. We set the size of the generated criminal network to be equal to 111, which is 5% of the nodes in the base social network. The number of edges in the synthetic networks varies based on the network typology, as described above. We use the network embedding process proposed by Yan (2013), which randomly connects nodes in the criminal network to nodes in the social network. This process first selects a degree d for each node in criminal network, where d is drawn from the original degree distribution of the larger network, and then connects this node to d randomly chosen nodes from the larger social network.

6 Experimental setup

In this section, we describe our experimental setup. To evaluate the performance of RedLearn and RedLearnRS, we compare them to several meaningful baseline monitor placement strategies, over a variety of network settings. In Sect. 6.1, we describe these baseline monitor placement strategies.

To perform a comprehensive evaluation of how the monitor placement algorithms perform across different error settings, we consider three categories of errors: (1) Node color errors: how monitors on red nodes report incorrect node colors, (2) Edge existence errors: how monitors on red nodes fail to report connections to other red nodes, and (3) Neighbor color errors: how node monitors may misreport neighbors' colors. These error behaviors are modeled by a variety of "error scenarios," which we describe in, respectively, Sects. 6.3, 6.5, and 6.2. Finally, we describe our experimental setup in Sect. 6.6.

Using these experiments, we answer the following questions:

– How do the various monitor placement algorithms perform when red nodes conceal some or all of their connections to other red nodes?
– How do the various monitor placement algorithms perform when incorrect noisy edges are present?
– How do the various monitor placement algorithms perform when the monitors misreport nodes' true color or wrongly estimate the colors of their neighbors?
– How do the various monitor placement algorithms perform on different criminal network typologies?
– Do RedLearn and RedLearnRS consistently, across a variety of misreporting scenarios, outperform the baseline monitor placement algorithms at the task of locating as many red nodes as possible?

6.1 Baseline monitor placement algorithms

We now describe several baseline monitor placement algorithms.

In Problem Setting 1, these baseline algorithms consider all observed but unmonitored nodes to select the next node to monitor. In Problem Setting 2, when monitors on red nodes may report wrong node color, baseline algorithms consider all blue nodes for repeated monitor placement. The node with the highest score among all blue nodes and unmonitored observed nodes is selected as the next node to monitor. The same repeated monitor thresholding mechanism is used as in Sect. 4.2.

Smart Random Sampling (SR) In each step, the smart random placement algorithm places a monitor on a random node, modeling chance.

Red Score (RS) The red score algorithm is guided by the colors reported by neighbors of a node. If a node v reports its neighbor u as red, the score associated with node u is increased by one, making it more suspicious. This algorithm selects the node with highest red score to place the next monitor. For this method, the red score is highly impacted by the accuracy of information given by the neighboring node. Additionally, due to its use of both red and blue node information, this algorithm uses the most amount of information as compared to the other baseline algorithms.

Most Red Say Red (MRSR) The MRSR algorithm places a monitor on the node with the greatest number of red neighbors who estimate it as a red node. It does not factor in blue node information and is dependent solely on the accuracy of the information given by neighboring red nodes. Blue nodes are essentially useless in this algorithm, mimicking

[9] Availabe at https://networkx.github.io/documentation/networkx-1.10/reference/generated/networkx.generators.random_graphs.powerlaw_cluster_graph.html#networkx.generators.random_graphs.powerlaw_cluster_graph.

[10] Available at https://networkx.github.io/documentation/networkx-1.10/reference/generated/networkx.generators.random_graphs.erdos_renyi_graph.html#networkx.generators.random_graphs.erdos_renyi_graph.

the reality when they might not know who the POIs are. This placement algorithm would result in a red node with no red neighbors being impossible to discover except by chance.

Most Red Neighbors (MRN) The MRN placement algorithm places a monitor on the node with the most known red neighbors. As such, it is highly dependent on network's homophily. Similar to the MRSR algorithm, blue neighbors are unimportant in determining the likelihood of a given node being red.

6.2 Node color errors

In the case of monitors misreporting the neighbors' colors (representing, e.g., a guilty individual shielding himself from detection), we allow a red node to report itself as blue with some predefined probability, drawn from a normal distribution, $\mathcal{N}(0.5, 0.125)$. We conducted experiments in which blue nodes have a small probability of reporting themselves as red, and obtained similar results. For the sake of brevity, we do not include these results.

6.3 Edge existence errors

In a dark network, red nodes actively try to hide their presence as well as the existence of some or all connections with other red nodes (for example, instead of using their normal cell phone to make calls to other red nodes, a red node might use a burner phone for such calls). To account for this, we consider versions of the datasets where some connections between red nodes are hidden uniformly at random. We have evaluated performance of monitor placement strategies considering how robust an algorithm is to hiding red connections. Note that this type of network presents a much more challenging setting, as one cannot simply rely on homophily to find red nodes.

6.4 Edge nonexistence errors

Any network that has been created using the data collected from node interactions and connections is prone to noisy edges that are not necessarily a part of the actual social network. We test the performance of proposed algorithms against noisy edges by adding random edges between disconnected node pairs in the original network.

There is another type of edge nonexistence errors, where criminals fabricate new edges to deceive POI detection algorithms. Michalak et al. (2017) presents how criminals can add new edges to attack network centrality measures such that leaders of criminal organizations are not the highest central nodes while keeping influence in the network. This work is not in scope for this paper.

6.5 Neighbor color errors

We assume the existence of a hierarchy among the nodes, and that nodes are more likely to conceal interactions with those above them in the hierarchy. (In the Noordin Top and synthetic terrorist networks, this hierarchy is known; in other networks, we infer it using node degree.) We assume that the red nodes are fully aware of the hierarchy, but blue nodes may or may not be aware of it, depending on the error scenario.

Consider nodes u and v, where $v \in N(u)$. The probability that a monitor on node u estimates v's color incorrectly is given by $P(e_{uv} \neq c_v)$, which depends on:

- The color of u (c_u) and color of v (c_v),
- The hierarchical position of u (L_u) relative to the position of v (L_v),
- The inherent investigation error E_u based on available information at node u.

We simulate the effort that a red node u would conceal its interactions with another criminal node v as $\frac{L_v}{L_u}$: i.e., criminals are more likely to conceal interactions with higher-ranking neighbors due to possible consequences of leaking information.

Case 1: Suppose u is a red node. The neighbor color error of a red neighbor $v \in N(u)$ is determined based on Eq. 3.

$$P(e_{uv} \neq c_v | c_v = red) = \min\left\{E_u * \frac{L_v}{L_u}, 1\right\}. \tag{3}$$

Equation 4 defines the probability u will estimate the wrong color of a blue node. This depends on inherent estimation error, E_u

$$P(e_{uv} \neq c_v | c_v = blue) = E_u. \tag{4}$$

Case 2: Suppose that uis a blue node. Whether or not u is even aware of red nodes depends largely on the domain, and in particular, whether the blue nodes are part of the same organization as the red nodes (blue and red nodes are all part of the dark network, and red nodes represent a subset of interest), or if the blue nodes represent individuals who are not part of the same organization as the red nodes (e.g., the red nodes represent dark nodes in a sea of blue node civilians).

- Neighbor color error type 1 (All nodes aware of red nodes).

  Here, $P(e_{uv} \neq c_v | c_u = blue, c_v = red)$ is determined using Eq. 3, since blue nodes know about red nodes and their hierarchy. Additionally, monitors which are placed on blue nodes may misreport colors of blue nodes depending on inherent analysts' error as mentioned in Eq. 4.

– Neighbor color error type 2 (only red nodes aware of other red nodes).

  In this case, it is not possible to distinguish whether a neighbor is red/blue by looking at their emails, messages, etc., because blue nodes don't know that their red neighbors are red. Here, blue nodes will simply report that all their neighbors are blue. Because of this, $P(e_{uv} \neq c_v | c_u = blue, c_v = blue) = 0$ and $P(e_{uv} \neq c_v | u = blue, v = red) = 1$.

In all cases, if $P(u\ lie\ v)$ is greater than 1, it is rounded down to 1.

6.6 Experimental settings

Because our error types are probabilistic and the inherent neighbor color error E_u of each node u may also change, for each network and lying scenario, we perform 25 runs of each monitor placement algorithm. For a fair comparison across different monitor placement strategies, we run each monitor placement algorithm with the same settings (including the same red starting node).

Monitor Budget In each run, we consider budgets of up to half the number of nodes in the network. We do this to illustrate the behavior of the algorithm over both small and large budgets, even though in practice it is unlikely that fully half of the nodes can be queried.

Training Learning Algorithms The Noordin Top network is small, and so we retrain RedLearn and RedLearnRS after each monitor is placed. The Pokec, Facebook, and Synthetic networks are larger, so for the sake of efficiency, we train the learning model once per every 20 monitors placed.

Node color errors A monitored red node misreports its color with some probability drawn from a normal distribution, $\mathcal{N}(0.5, 0.125)$. We consider this to be an error in the node color determination mechanism. Therefore, all monitored red nodes misreport their colors with the same probability as described in Sect. 6.2.

Edge existence errors In a given network, we consider that only red nodes try to conceal their connections with other red nodes with a fixed probability. We present results when red nodes hide their connections with probability 0 (original network), 0.5 (hiding half of red edges), and 1.0 (hide all red connections).

Edge nonexistence errors We evaluate monitor placement algorithm performance in the presence of noise (incorrect edges) by adding edges with a probability of 0.01, between any two disconnected nodes in the original network.

Neighbor color errors In these experiments, the inherent neighbor color error at each node is drawn from a normal distribution, $E \sim \mathcal{N}(0.5, 0.125)$. For the Noordin Top network, ground truth hierarchy scores are known, as shown in Table 4. In the Pokec and Facebook100 networks, we set the hierarchy score to be the degree of the node.[11] For the synthetic networks, hierarchy assignments are as follows: In the standard and regional typologies, the hierarchy score of a node is equal to the number of descendant nodes that it has; and in the other typologies, the hierarchy score of a node is equal to its degree. Given some error type, a monitored node u, misreports neighbor v's color with probability $P(e_{uv} \neq c_v)$ as mentioned in Sect. 6.5.

Table 4 Noordin Top network hierarchy assignment

Role	Hierarchy score	No. of nodes
Strategist	5	10
Commander; religious leader	4	23
Trainer/instructor; bomb maker; facilitator; propagandist; recruiter	3	33
Bomber/fighter; suicide bomber; courier; recon/surveillance	2	33
Unknown	1	40

7 Results and analysis

This section is organized as follows: Because node classification algorithms appear to be a natural approach to this problem, we begin by presenting a comparison of RedLearn against the ICA node classification algorithm in Sect. 7.1. Next, we consider Problem Setting 1, in which monitors accurately report the color of the occupied node. We compare RedLearn to the baseline monitor strategies discussed in Sect. 6.1. We examine how the performances of these monitor placement algorithms are affected by (1) red nodes concealing their connections with other red nodes with probability 0, 0.5, and 1.0, (2) the neighbor color error type used by the nodes, and (3) the monitor placement budget.

We next move to Problem Setting 2, where monitors may misreport the color of the occupied node. In Sect. 7.3, we evaluate the performance of RedLearnRS as compared to baseline algorithms.

In Sect. 7.4, we evaluate performance of monitor placement algorithms when nodes report incorrect edges. We then conduct a feature importance analysis in Sect. 7.5 to determine the most important features for RedLearn and RedLearnRS under different neighbor color error types and edge existence errors. Finally, in Sect. 7.6, we evaluate RedLearnRS on the various dark network typologies discussed in Sect. 5.4

[11] Results were similar for different types of centrality, including eigenvector and betweenness centrality.

7.1 Node classification algorithms

In this section, we compare performance of the node classification algorithm (ICA) to RedLearn. Figure 3 compares the performance of ICA to RedLearn on several networks, under Problem Setting 1, for the two neighbor color error settings. In both algorithms, we used logistic regression as the base classifier with the same set of features. RedLearn outperforms ICA in both error types in all networks we have considered. A similar trend is followed even when we introduce edge existence errors by concealing edges among red nodes.

ICA performs poorly because the training is performed on a sampled network, and complete information about network topology is necessary to provide accurate predictions. Prediction errors are propagated throughout the predictions in ICA, because ICA uses predicted labels to calculate features. Even though ICA performs well in node classification problems when complete information is present, it exhibits poor performance in partial information scenario.

On the other hand, RedLearn uses both node colors that are reported by the monitors and connections in making accurate predictions about node colors. RedLearn does not give priority to network's topology while making the predictions, whereas ICA heavily depends on topology to calculate neighborhood-based features.

7.2 Problem Setting 1: node color reliable

In this section, we present results for the monitor placement algorithms for Problem Setting 1. Figure 4 shows the performance of RedLearn on the NoordinComs4 network for (a) when no edge existence errors are present, (b) red edges are concealed with probability 0.5, and (c) all red edges are concealed. When all edges between red nodes are available, the problem becomes easy, and the simple algorithm of monitoring the node with the most red neighbors (MRN) is best (because the red nodes exist in a near-clique). However, note that in both lying scenarios, even when all red edges are revealed, RedLearn is second to MRN.

However, we see that as edges between red nodes are increasingly concealed, the MRN performance worsens. In such cases, RedLearn performs much better than all comparison methods: It is able to learn the patterns of reporting and structural characteristics of red nodes, achieving the highest performance.

We present the average performance of monitor placement algorithms on different attributed social networks in Figs. 5 and 6. Figure 5 shows monitor placement algorithm performances for neighbor color error type 1, and Figure 6 shows average performance of neighbor color error type 2.

The NoordinTop and Facebook100-Year networks have very high node color homophily, and so all monitor placement algorithms perform well compared to random placement. However, in Facebook100-Major and Pokec networks, node color homophily is very low, and we observe overall lower performance of the various strategies. MRN and RedLearn perform equally well when there are no edge existence errors, and all red edges are correctly reported, but as before, when red edges are concealed, RedLearn is generally the best.

We see similar patterns across all networks: Edges between red nodes help select the node with the most red neighbors, and MRN outperforms all algorithms closely followed by RedLearn; but when these edges are concealed, RedLearn is the clear winner.

7.3 Problem Setting 2: node colors misreported

In this section, we evaluate the performance of monitor placement algorithms, including RedLearnRS, when node colors are misreported by monitors.

Figure 7 shows that the results on the NoordinComs4 network follow a similar pattern as in Problem Setting 1: When reported edges are reliable, MRN performs well, with RedLearnRS close behind; as we introduce edge existence errors by hiding edges among red nodes, RedLearnRS becomes the clear winner.

All monitor placement strategies are affected by node color errors, because they must expend greater amounts of budget to repeatedly monitor the same node. Thus, the performance of the baseline algorithms is reduced in this problem setting, since they do not employ a mechanism to find possible node color inaccuracies. In placing the next monitor, they have to consider all believed-blue nodes and unmonitored nodes. In contrast, RedLearnRS identifies possible node color errors among blue nodes using the outlier detection algorithm and only considers these error nodes and unmonitored nodes when placing the next monitor. RedLearnRS performs similarly across the different neighbor color error types considered.

We summarize the average performance of monitor placement algorithms across all networks in Figs. 8 (NCE1) and 9 (NCE2). In all networks where attribute assortivity (homophily) is low (Pokec, Facebook100-Major networks), RedLearnRS out performs all other monitor placement strategies. When attributes show homophily, RedLearnRS performs quite similar to the best monitor placement algorithm (MRN).

Our conclusion is that as we conceal red edges, RedLearnRS is the best performing monitor placement algorithm. While it comes in second in a few cases in which all the original edges between red nodes are known, it closely follows the performance of MRN, and it outperforms it in some cases. Also, the situation of all the existing edges being known is not realistic, so the performance of RedLearnRS

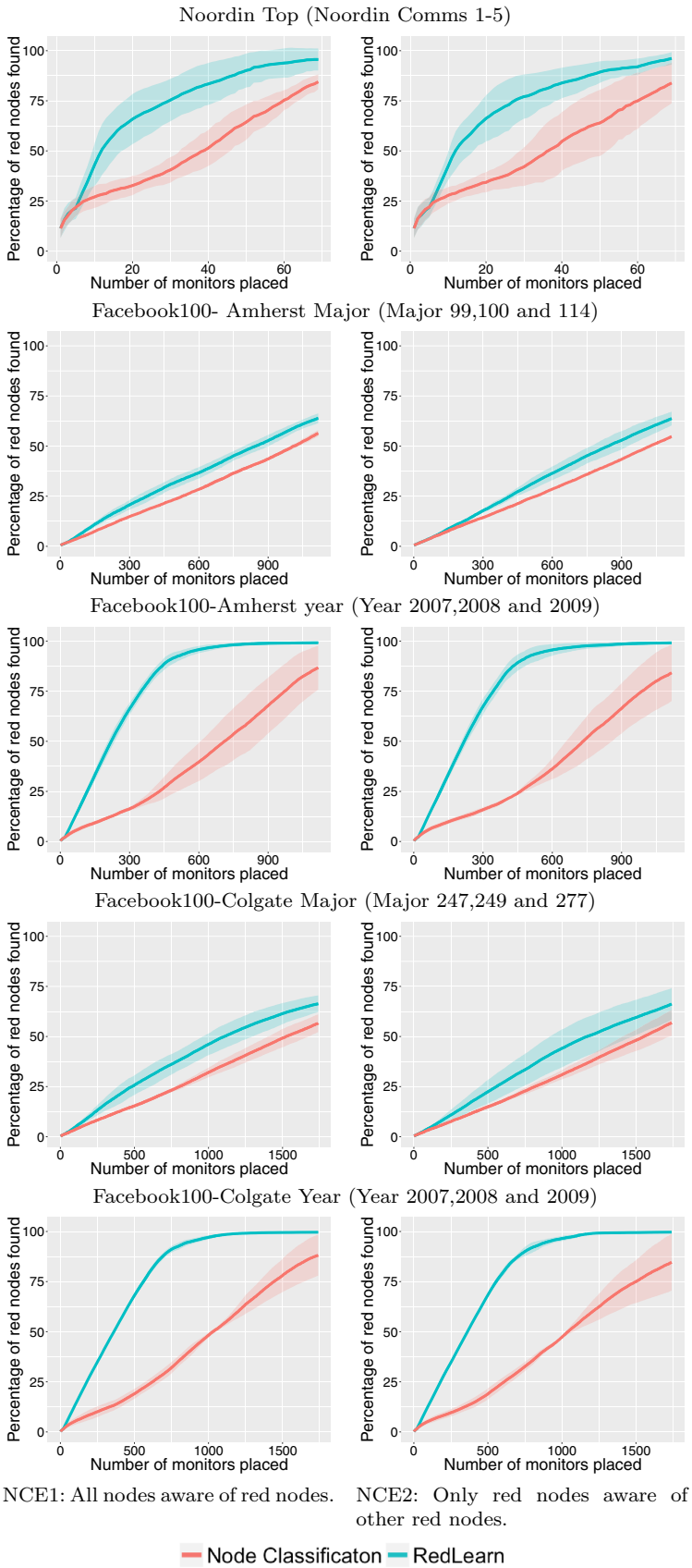

Fig. 3 Comparison of average performance of ICA and RedLearn on the NoordinTop terrorist network and Facebook100 networks. RedLearn outperforms ICA in all networks for both neighbor color error types considered. Noordin Top (Noordin Comms 1–5). Facebook100-Amherst Major (Major 99,100, and 114). Facebook100-Amherst Year (Year 2007, 2008, and 2009). Facebook100-Colgate Major (Major 247, 249, and 277). Facebook100-Colgate Year (Year 2007, 2008, and 2009). NCE1: All nodes aware of red nodes. NCE2: Only red nodes aware of other red nodes

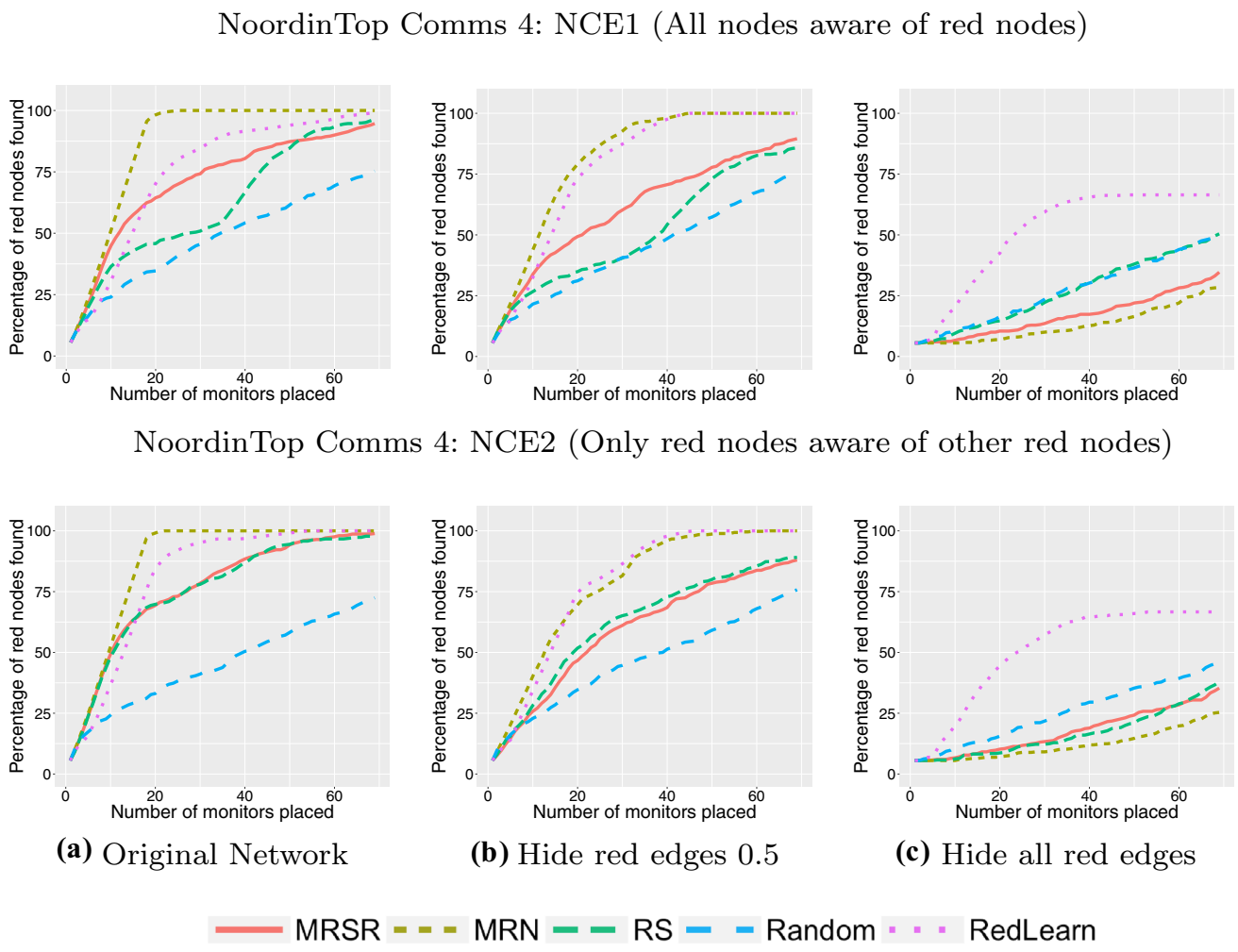

Fig. 4 Comparison of monitor placement algorithms on the NoordinComs4 network when reported node colors are reliable (Problem Setting 1). MostRedNeighbors (MRN) performs well when there are no edge existence errors. RedLearn performs consistently well across all edge existence error scenarios. NoordinTop Comms 4: NCE1 (All nodes aware of red nodes). NoordinTop Comms 4: NCE2 (Only red nodes aware of other red nodes). **a** Original Network. **b** Hide red edges 0.5. **c** Hide all red edges

in the cases when some or all of these edges are missing is most relevant to dark networks.

7.4 Edge nonexistence error analysis

In this section, we evaluate performance of monitor placement algorithms when the monitors report incorrect edges. This case is the most realistic one, incorporating noise that is normally present in real data collection. We included random edges between disconnected nodes in the original network with probability 0.01 to evaluate this phenomenon.

Figures 10 and 11 present average results for NoordinTop networks. The performance of all monitor placement algorithms is not drastically affected by the incorrect edges that were added to the network.

We observe a similar pattern of results for all networks and all error scenarios we have considered in this paper.

7.5 Feature importance analysis

In this section, we determine the most important features for good performance for RedLearn and RedLearnRS. We look at how feature importance changes when red nodes do or do not show homophily, as well as when node colors are misreported. We examine the NoordinTop terrorist networks and Facebook100-Amherst Major networks for our analysis, since these networks show high and low color assortivity, respectively.

We have divided features in to four different categories to better explain feature importance: (1) Neighbor based: Number of blue/red neighbors (2) Tight knit groups: Number

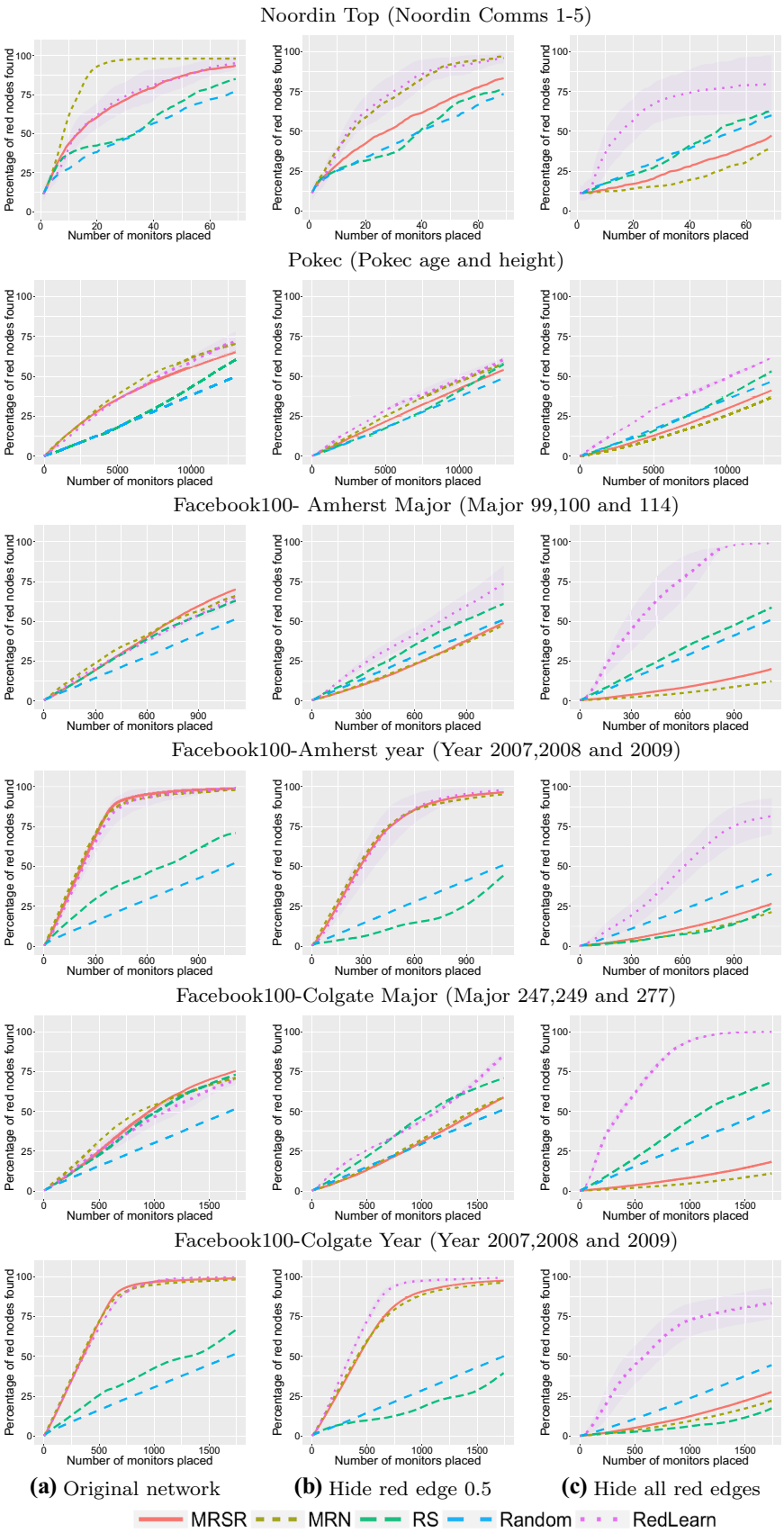

Fig. 5 Monitor placement algorithm average performance when reported node color is reliable (Problem Setting 1). Neighbor color error type 1 (NCE1: All nodes aware of red nodes) is considered here. MostRedNeighbors (MRN) and RedLearn algorithms perform similarly when there are no edge existence errors. As we introduce these errors,RedLearn becomes the clear winner across all monitor placement algorithms. Noordin Top (Noordin Comms 1–5). Pokec (Pokec age and height). Facebook100-Amherst Major (Major 99, 100, and 114). Facebook100-Amherst Year (Year 2007, 2008, and 2009). Facebook100-Colgate Major (Major 247, 249, and 277). Facebook100-Colgate Year (Year 2007, 2008, and 2009). **a** Original network. **b** Hide red edge 0.5. **c** Hide all red edges

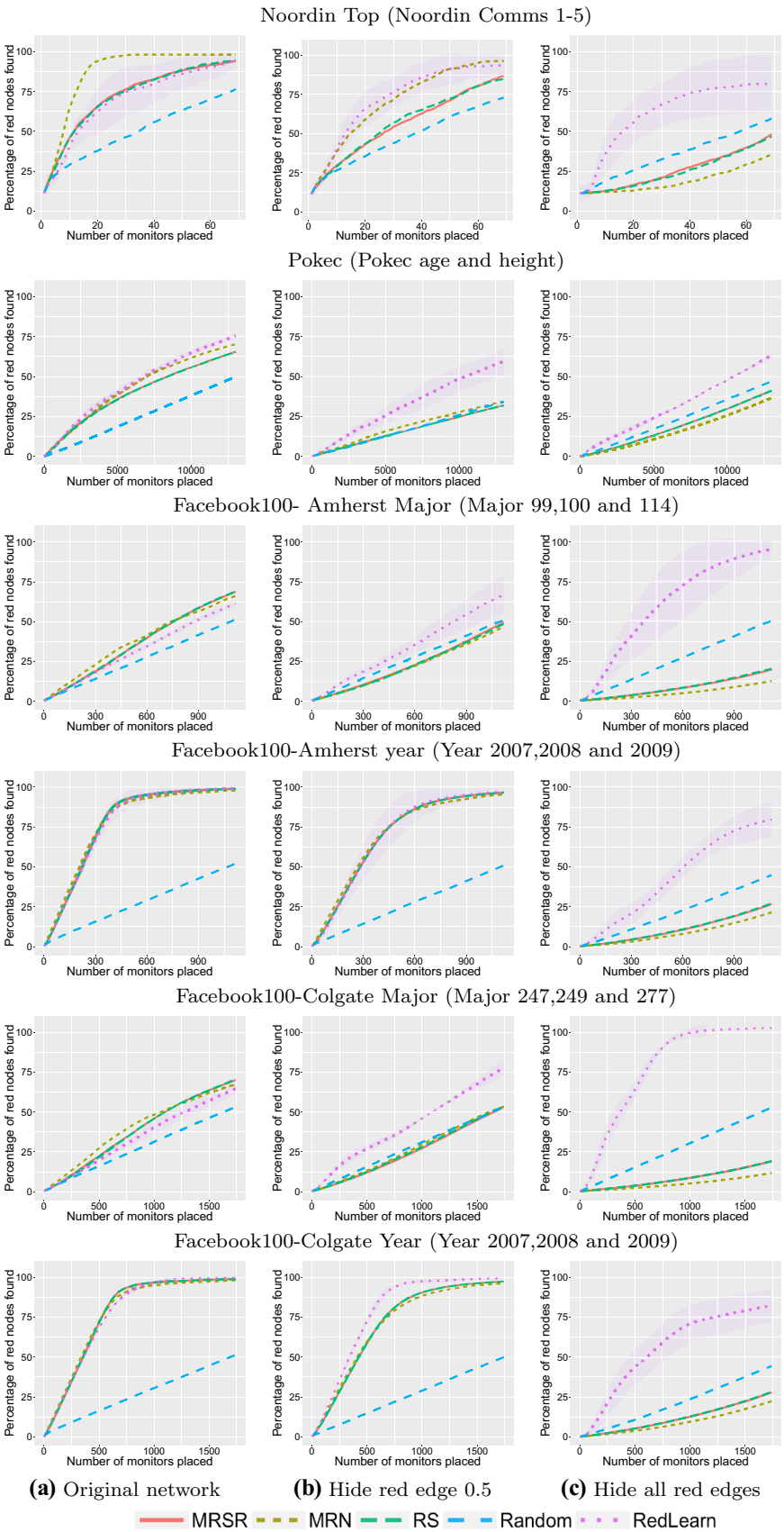

Fig. 6 Monitor placement algorithm average performance when reported node colors are reliable (Problem Setting 1). Neighbor color error type 2 (NCE2: Only red nodes aware of other red nodes) is considered here. RedLearn has matched the performance of MostRedNeighbors (MRN), in most networks when there are no edge existence errors. As red nodes hide their connections with other red nodes, RedLearn becomes the clear winner. Noordin Top (Noordin Comms 1–5). Pokec (Pokec age and height). Facebook100-Amherst Major (Major 99, 100, and 114). Facebook100-Amherst Year (Year 2007, 2008, and 2009). Facebook100-Colgate Major (Major 247, 249, and 277). Facebook100-Colgate Year (Year 2007, 2008, and 2009). **a** Original network. **b** Hide red edge 0.5. **c** Hide all red edges

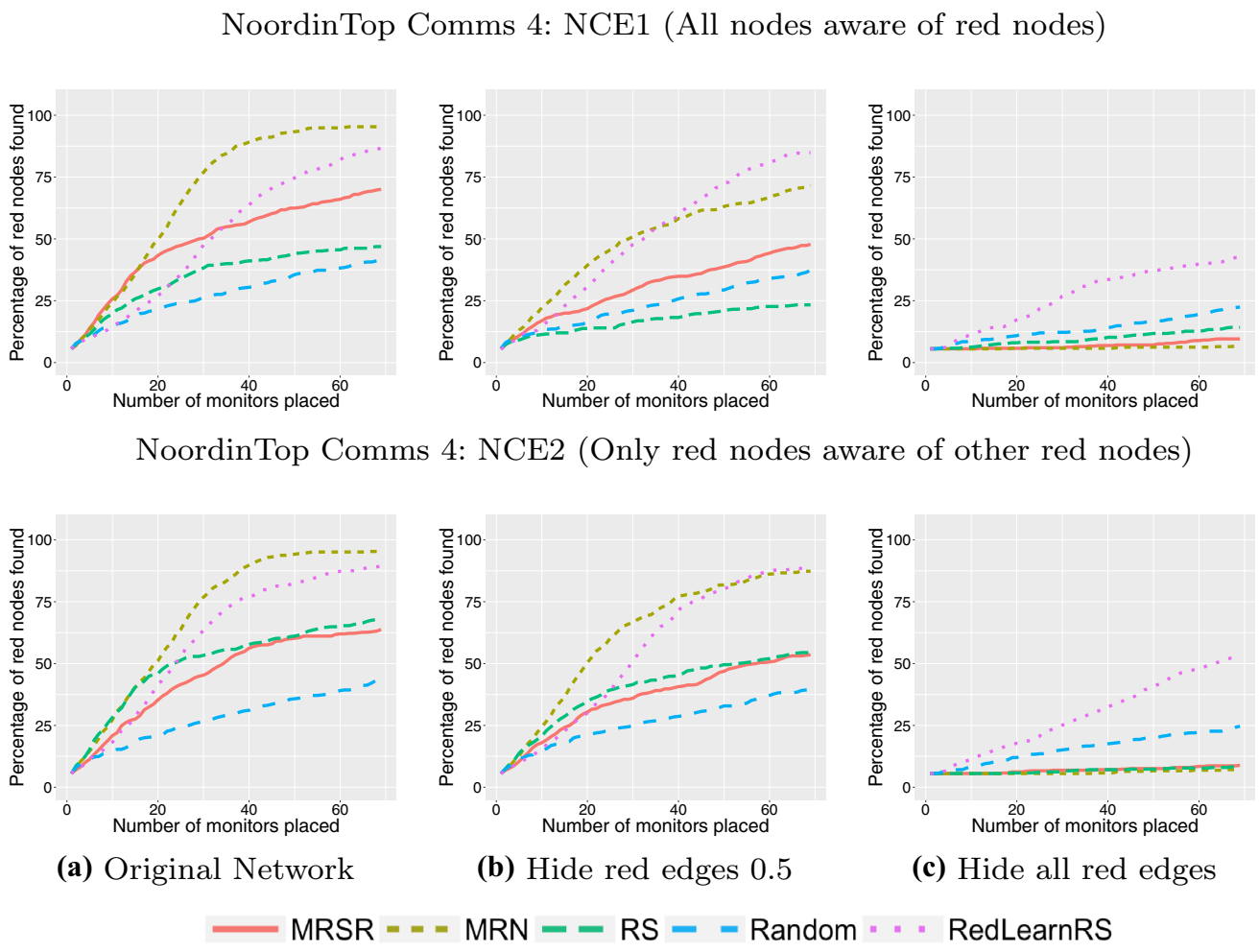

Fig. 7 Comparison of monitor placement algorithms on the NoordinComs4 network when monitors misreport node colors (Problem Setting 2). Even though MostRedNeighbors (MRN) performs well in the original network, its performance worsens when edge existence errors are introduced. RedLearn performs consistently well across all neighbor color error types and edge existence error scenarios. NoordinTop Comms 4: NCE1 (All nodes aware of red nodes). NoordinTop Comms 4: NCE2 (Only red nodes aware of other red nodes). **a** Original network. **b** Hide red edges 0.5. **c** Hide all red edges

of red triangles, (3) Neighbor estimates based: Number of red/blue neighbors estimate red/blue, red score and inferred probability of red, and (4) Second-hop neighbors: Second-hop red/blue neighbors. When node colors are misreported (Problem Setting 2), we consider an additional repeated monitor category, the number of monitors placed that blue/red nodes lied about.

We evaluate feature importance using absolute values of the coefficients of each feature at the logistic regression decision boundary. The average feature importance of each category is reported.

Problem Setting 1: node color reliable

Figure 12 presents average feature importance scores for NoordinTop terrorist networks and Facebook100-Amherst Major networks. Tight knit groups are the most important feature group for RedLearn in NoordinTop networks. This agrees with our intuition, as in NoordinTop networks red nodes are mutually adjacent. Neighbor estimates do not have a high importance in NoordinTop networks, since red nodes can be easily identified using neighborhood features. In Facebook100-Amherst Major networks neighbor color estimates, neighbors and tight knit groups are equally important since red nodes are much difficult to distinguish from blue nodes.

Feature importance changes as we introduce different types of errors into the problem. When we introduce edge existence type of errors (by hiding edges between red nodes), the neighbor feature category improves importance in both network types. This is primarily due to RedLearn learning that red nodes tend to have fewer red neighbors, and it obtains outstanding performance when we remove edges among red nodes. Other neighborhood-based monitor

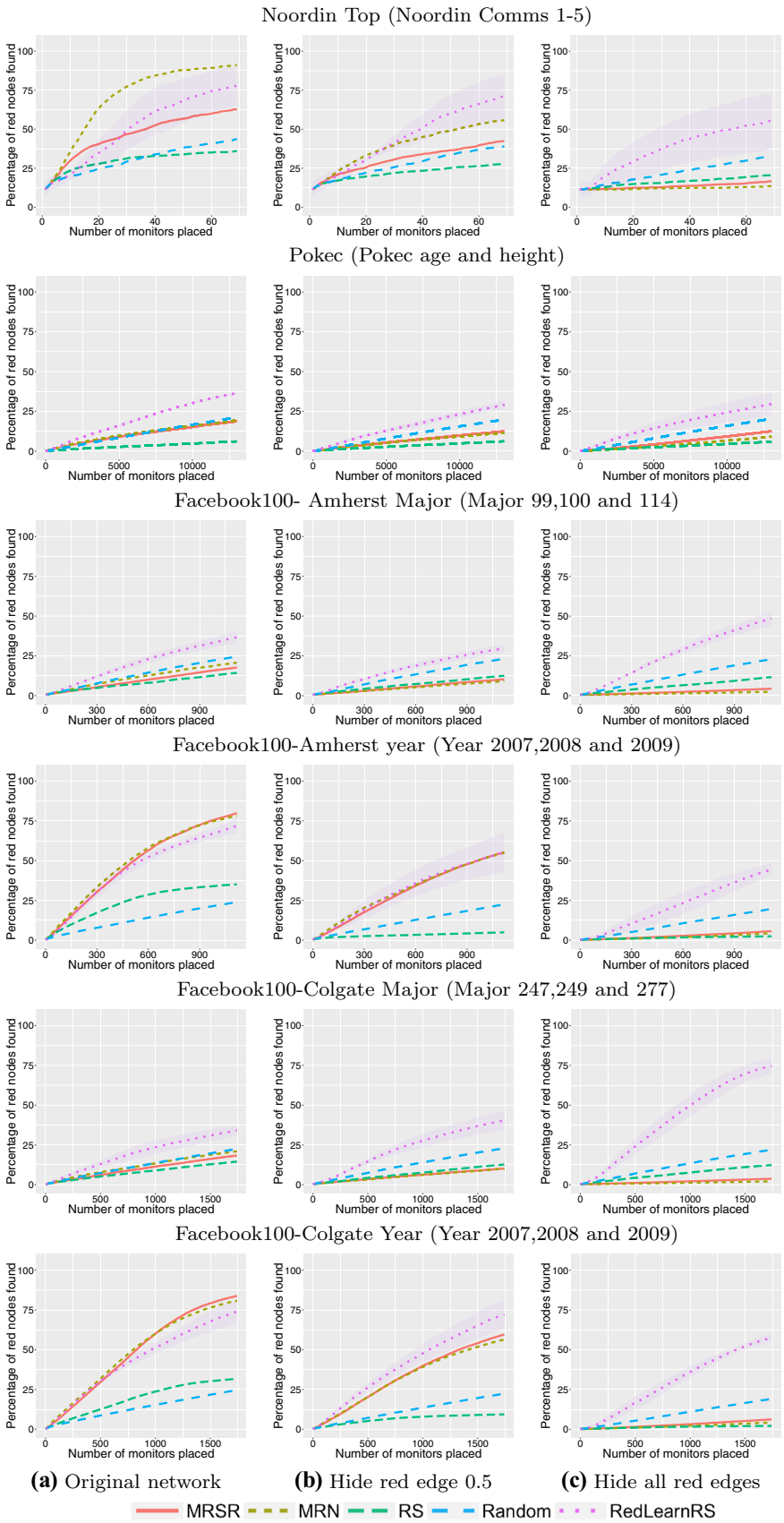

Fig. 8 Monitor placement algorithm average performance when node colors are misreported by monitors (Problem Setting 2), under neighbor color error type 1 (NCE1: All nodes are aware of red nodes). MostRedNeighbors (MRN) performs well when there are no edge existence errors, but worsens as we introduce such errors. RedLearn performs consistently well across all edge existence error scenarios. Noordin Top (Noordin Comms 1–5). Pokec (Pokec age and height). Facebook100-Amherst Major (Major 99, 100, and 114). Facebook100-Amherst Year (Year 2007, 2008, and 2009). Facebook100-Colgate Major (Major 247, 249, and 277). Facebook100-Colgate Year (Year 2007, 2008, and 2009). **a** Original network. **b** Hide red edge 0.5. **c** Hide all red edges

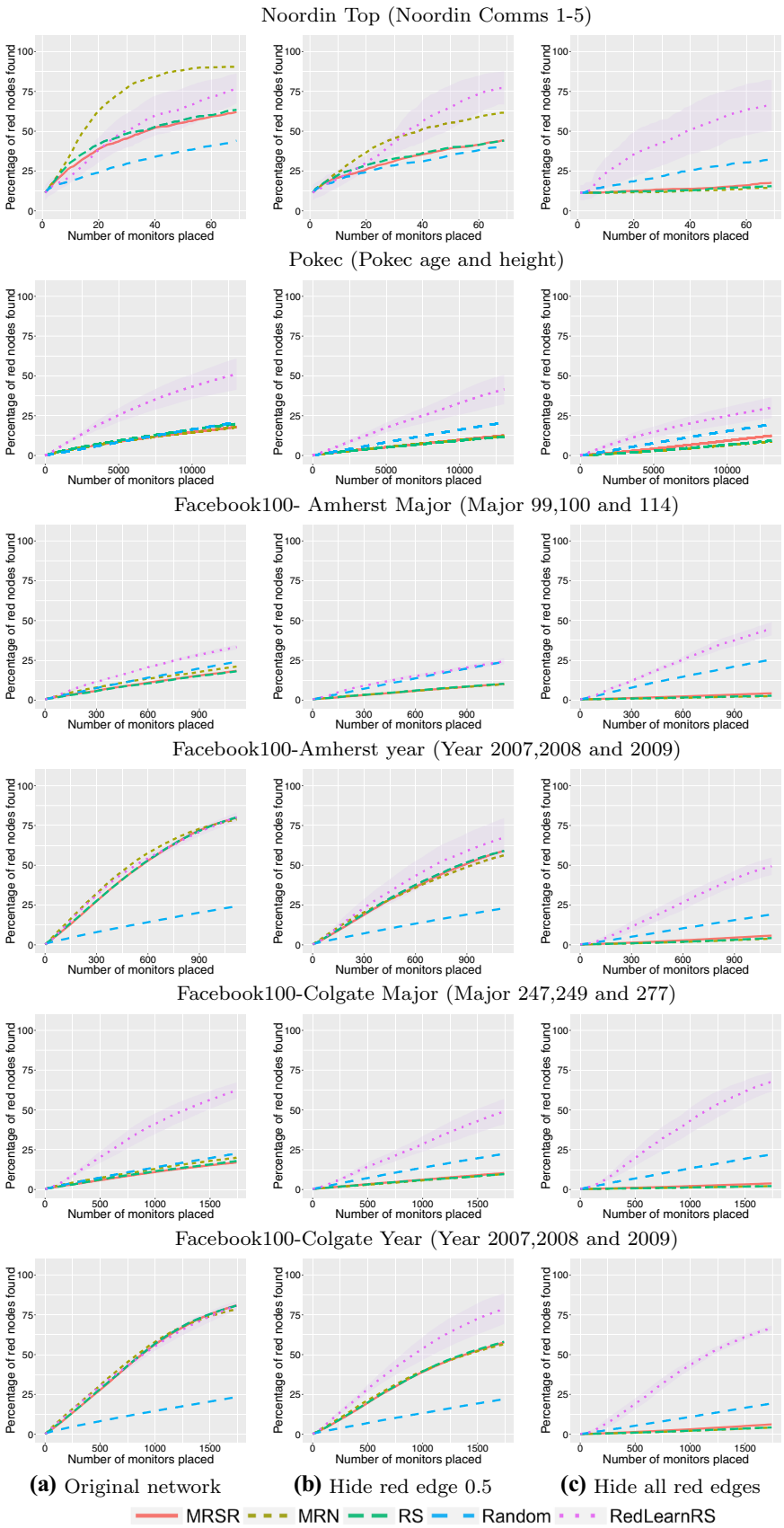

Fig. 9 Average performance of monitor placement algorithms when node colors are misreported (Problem Setting 2), under neighbor color error type 2 (NCE2: Only red nodes are aware of other red nodes). RedLearn performs consistently well across all edge existence error scenarios, when other monitor placement algorithms are affected heavily by concealing red edges. Noordin Top (Noordin Comms 1–5). Pokec (Pokec age and height). Facebook100-Amherst Major (Major 99, 100, and 114). Facebook100-Amherst Year (Year 2007, 2008, and 2009). Facebook100-Colgate Major (Major 247, 249, and 277). Facebook100-Colgate Year (Year 2007, 2008, and 2009). **a** Original network. **b** Hide red edge 0.5. **c** Hide all red edges

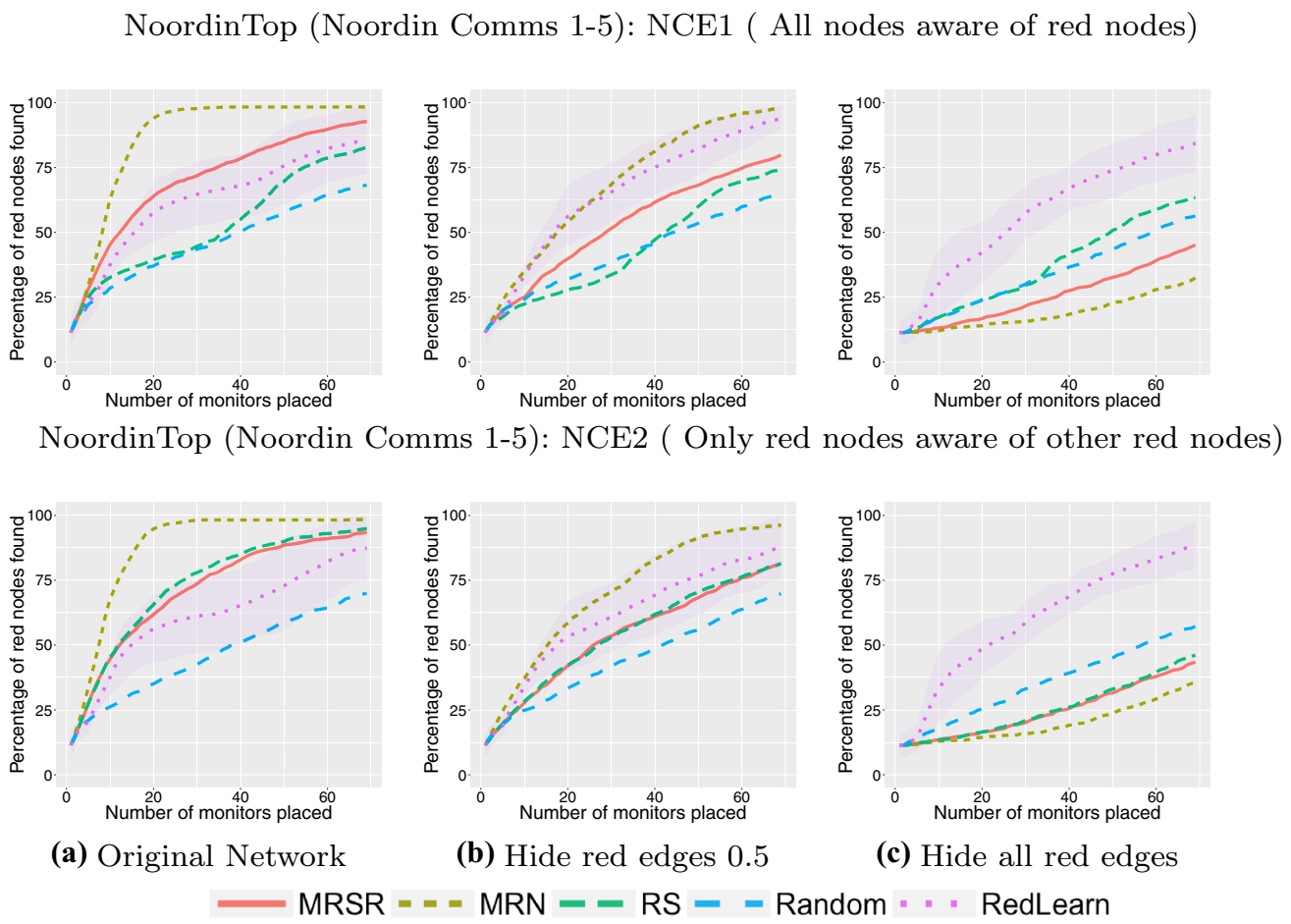

Fig. 10 Average performance of monitor placement algorithms on the NoordinTop networks when monitors report incorrect edges. Problem Setting 1: Node colors are reliable is considered here. Performance of monitor placement algorithms is not heavily affected by noisy edges. NoordinTop (Noordin Comms 1–5): NCE1 (All nodes aware of red nodes). NoordinTop (Noordin Comms 1–5): NCE2 (Only red nodes aware of other red nodes). **a** Original network. **b** Hide red edges 0.5. **c** Hide all red edges

placement algorithms are not capable of learning this phenomenon. Feature importance scores remain quite similar across different neighbor color error types considered.

Problem Setting 2: node color misreported

Figure 13 shows average feature importance scores for RedLearnRS. We add the repeated monitor feature category, which was introduced to determine whether to place a monitor on an already-monitored node. This feature category has high importance in both networks considered. Other feature categories have similar importance scores as Problem Setting 1 as seen in Fig. 12.

7.6 Network structure analysis

In this section, we discuss how monitor placement algorithms perform for several real network structures observed in terrorist networks. Recall that we create these networks by embedding different synthetically created terrorist network hierarchies into the Facebook100-Amherst social network, as discussed in Sect. 5.4.

Figure 14 depicts the performance of monitor placement algorithms when node colors reported by monitors are reliable (Problem Setting 1). We see that in standard and regional hierarchies, RedLearn outperforms the other monitor placement strategies, and the MRN method performs poorly. This occurs because standard and regional hierarchies have a chain of command and follows a tree structure, so each node will be connected to very limited number of red nodes. In such a structure, RedLearn is able to learn the neighborhood characteristics of a red node.

In neighbor color error type 2, the performance is affected by only red nodes' knowing colors of other red nodes. Because each red node is connected to a very limited number of other red nodes, neighbor color estimates do not reveal much information.

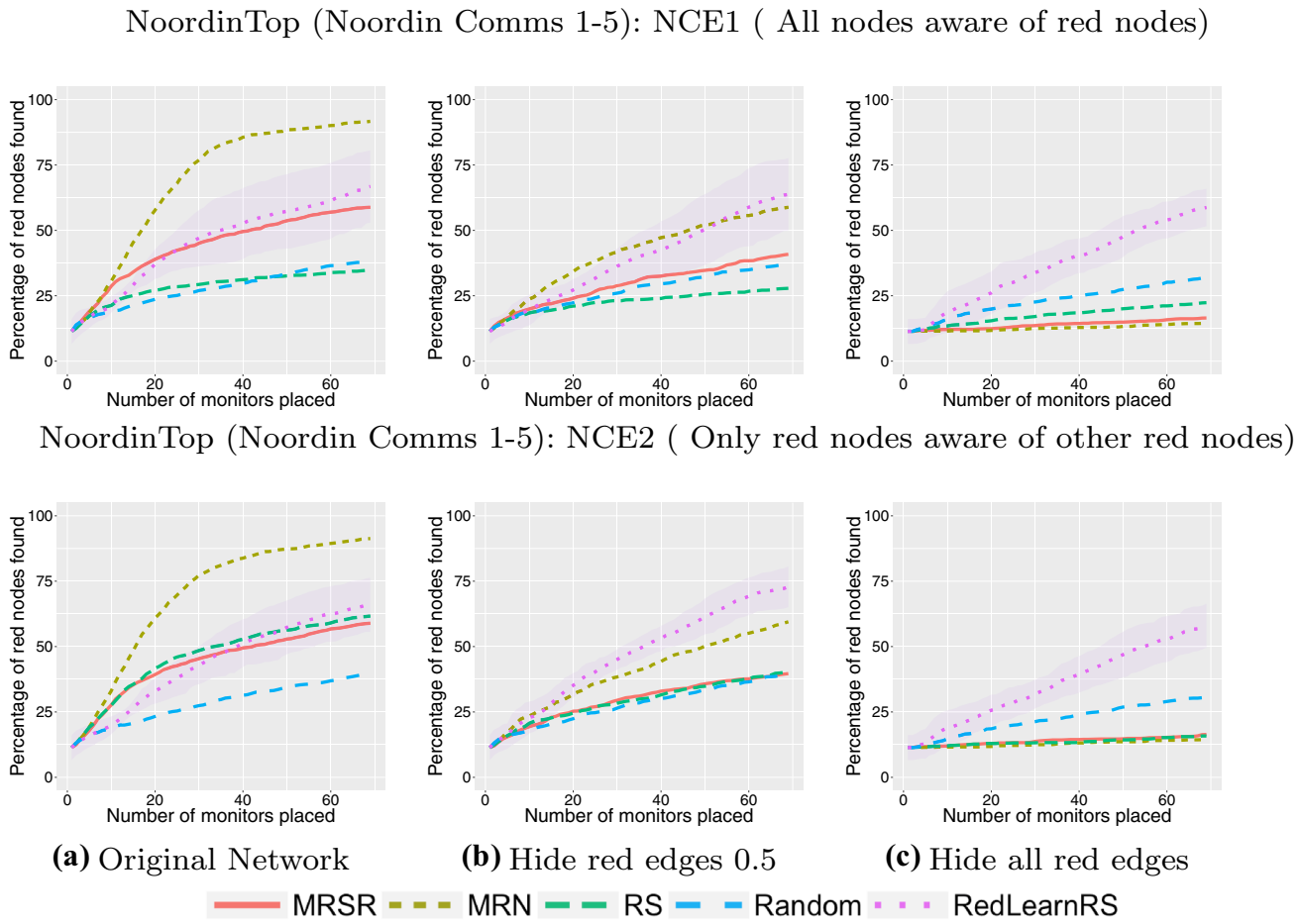

Fig. 11 Average performance of monitor placement algorithms on the NoordinTop networks when monitors report incorrect edge. Problem Setting 2: Node colors are misreported is considered here. Performance of monitor placement algorithms remains quite similar to when there are no noisy edges. NoordinTop (Noordin Comms 1–5): NCE1 (All nodes aware of red nodes). NoordinTop (Noordin Comms 1–5): NCE2 (Only red nodes aware of other red nodes). **a** Original network. **b** Hide red edges 0.5. **c** Hide all red edges

In clustered and core group hierarchies, most red neighbors (MRN) perform well because red nodes are connected to many other red nodes. Overall, RedLearn performs consistently well across all types of terrorist network structures.

Figure 15 shows monitor placement algorithm performance when the colors of monitored nodes may be misreported (Problem Setting 2). In this setting, RedLearnRS outperforms all monitor placement strategies in all network hierarchies and neighbor color error types. In the clustered hierarchy and core group hierarchy, MRN performs quite well, but it is not able to beat RedLearn. Overall, the baseline algorithms perform worse than random monitor placement in most network hierarchies, because they cannot identify blue nodes with color errors.

8 Discussion and conclusion

Members of dark networks conceal information by nature, and while these networks are deceptive and sparse, they are still structured. Based on these properties, we created and analyzed the results of several methods of sampling the networks to identify people of interest (POI or "red nodes"). We tested these methods on a small real terrorist network, larger social networks, and synthetic terrorist networks embedded into real social networks. As our monitors represent analysts that may misreport findings, we used several error scenario models, including node color errors, neighbor color errors, neighbor existence error, and neighbor nonexistence errors.

We created RedLearn, a learning-based method for locating POI in dark networks when reported node colors are reliable (Problem Setting 1). RedLearn uses features from

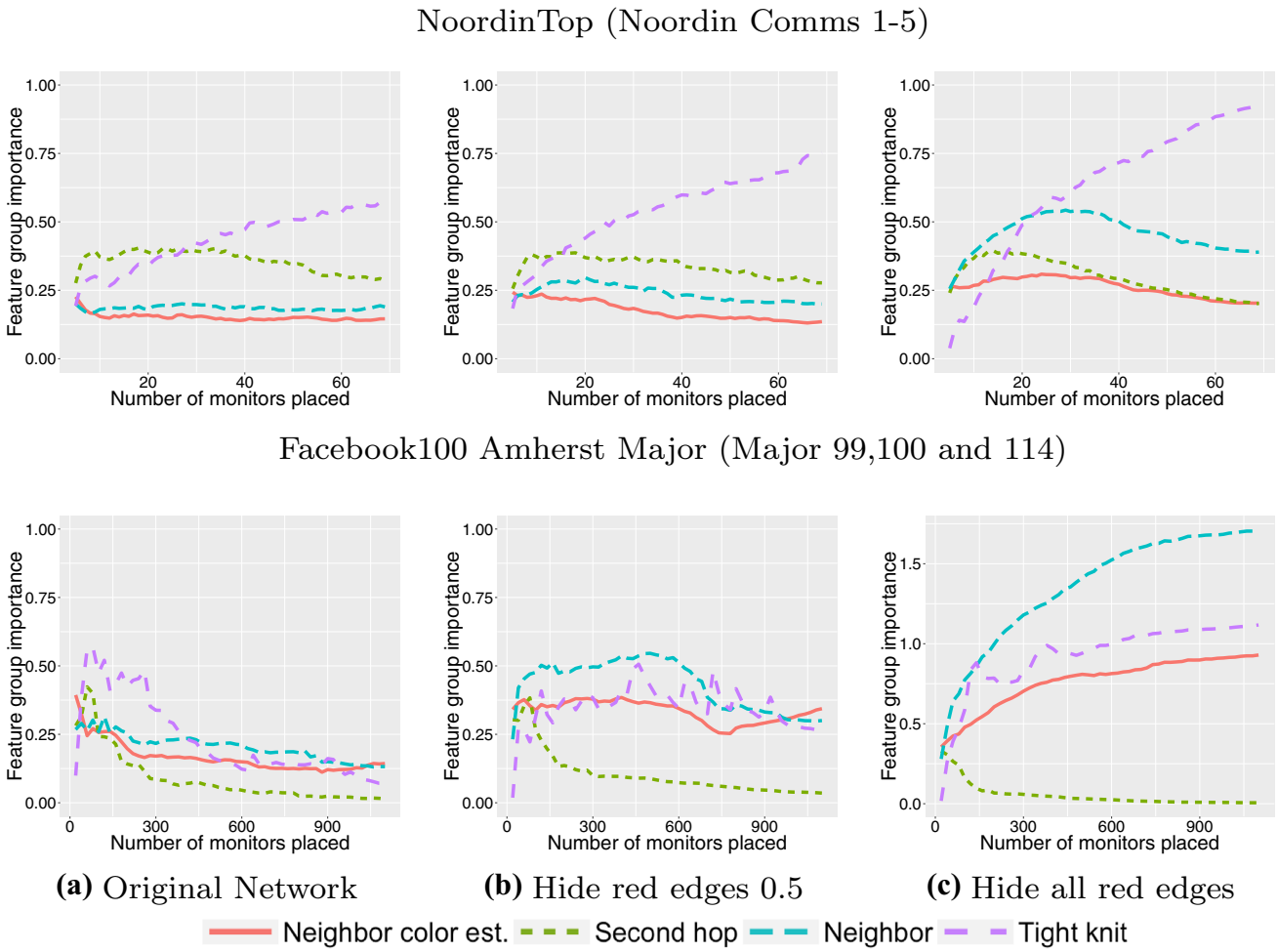

Fig. 12 Average feature importance scores of NoordinTop and Facebook100-Amherst Major networks when reported node colors are reliable (Problem Setting 1). Tight knit groups are the most important feature category in NoordinTop networks, since red nodes tend to be clustered together. The importance of the neighbor feature category increases when we hide red edges. NoordinTop (Noordin Comms 1–5). Facebook100-Amherst Major (Major 99, 100, and 114). **a** Original network. **b** Hide red edges 0.5. **c** Hide all red edges

the simpler baseline algorithms and learns how to identify red nodes in networks. Results show that RedLearn outperforms the other methods in cases where one cannot rely on nodes to reveal all their connections (edge existence errors). RedLearn performed well across the different neighbor color error types we considered.

We then introduced Problem Setting 2, where node colors may be reported with errors. In this setting, we used RedLearnRS, a re-sampling algorithm that extends RedLearn to identify POIs. All monitor placement algorithm performances were affected in this case since multiple monitors needed to be placed on a single node to uncover the true color; however, RedLearnRS outperformed other baseline monitor placement algorithms by successfully identifying POIs that were initially misreported.

RedLearnRS assumes that only red nodes will misreport their color. Therefore, it is necessary to re-monitor nodes that are reported to be blue to identify potential node color errors. This might lead to the risk of repeatedly monitoring innocent blue nodes. We introduced repeated monitor threshold in RedLearnRS to minimize number of times a blue node will be monitored. Our analysis shows that the RedLearnRS algorithm re-samples a blue node 1.61 times on average with a standard deviation of 1.2 compared to 1.68 average and 1.03 standard deviation of re-sampling a red node. This shows that RedLearnRS does not excessively monitor the same blue node. Another important fact is that when we refer to "monitoring" some node, it doesn't necessarily mean that the node will be arrested and questioned. It is rather an analyst will spend more time and resources determining the true color of a node.

In general, all monitor placement algorithms performed well in networks where red nodes tend to have direct connections with other red nodes (high color assortivity present in

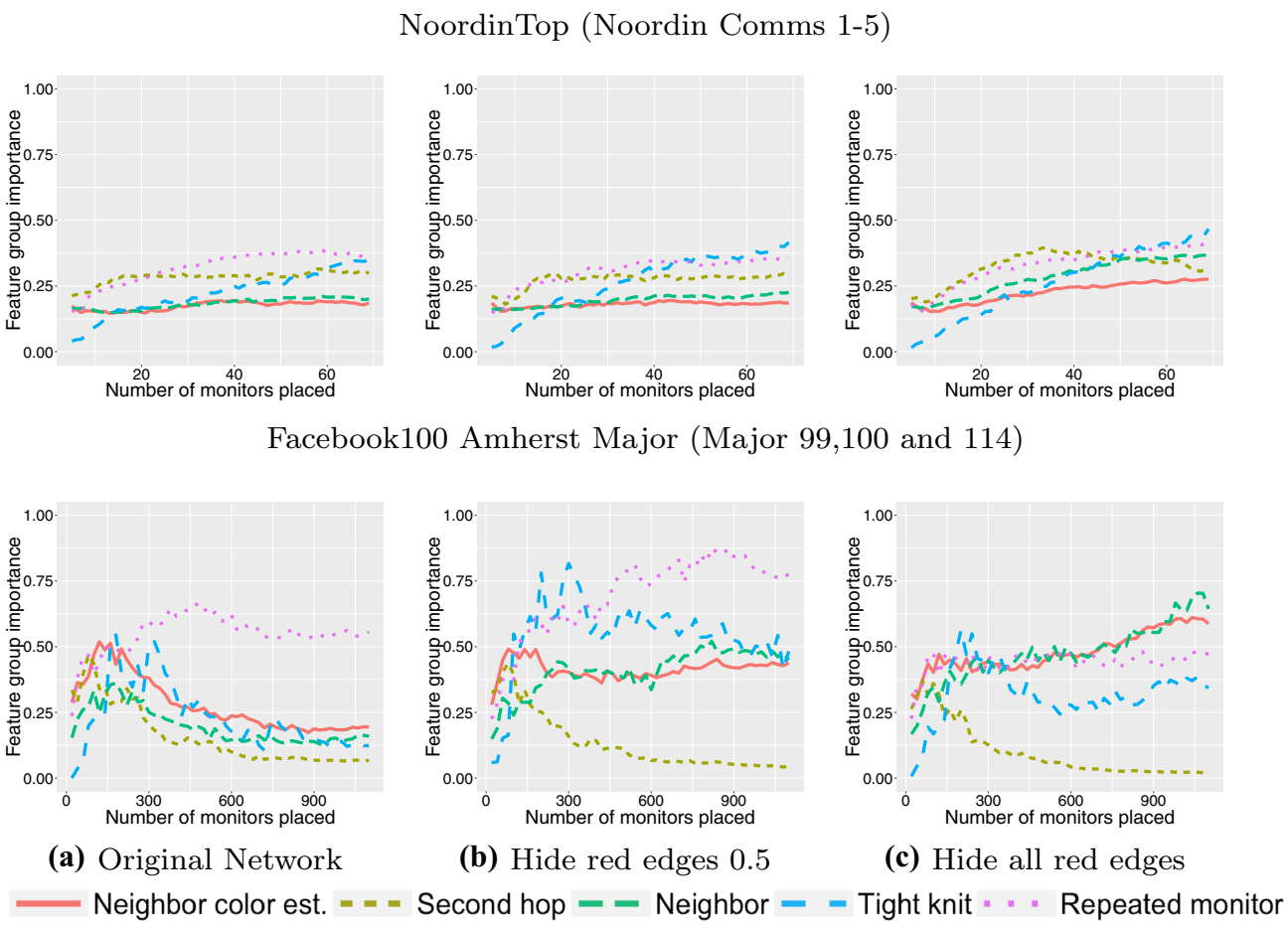

Fig. 13 Average feature importance scores of NoordinTop and Facebook100-Amherst Major networks when node colors may be misreported by monitors. Repeated monitor feature group has the highest importance in most networks since this feature category determines whether to place a monitor on an already-monitored node. NoordinTop (Noordin Comms 1–5). Facebook100-Amherst Major (Major 99, 100, and 114). **a** Original network. **b** Hide red edges 0.5. **c** Hide all red edges

Noordin Top, Facebook100-Year attribute); and vice versa when red node assortivity is low (Facebook100-Major attribute and Pokec networks). The MostRedNeighbors (MRN) monitor placement algorithm performed the best in highly color assortative networks since red nodes tend to have many red neighbors. The performance of MRN was drastically affected by edge existence errors . On the other hand, RedLearn performed well across all network types and error types. All monitor placement algorithms are not affected by small edge nonexistence errors

RedLearn and RedLearnRS use several network structure-based features and neighbor color estimate-based features to identify nodes of interest in social networks. We conducted a feature importance analysis to determine most important features in different network structures and problem settings. Tight knit groups were seen to be most important feature group when red node assortivity is high (NoordinTop networks). For other networks, neighbors, neighbor color estimates, and tight knit groups showed to be equally important. Repeated monitors feature category was the most important feature category in Problem Setting 2, when node colors are reported with errors.

We generated several criminal network hierarchies in accordance with the criminology literature and embedded these synthetic criminal network structures into Facebook100-Amherst network to determine which algorithms perform well across different criminal hierarchies. Our results showed that MRN monitor placement works well when criminals are clustered together in groups. In all other hierarchies, RedLearn and RedLearnRS outperform baseline monitor placement algorithms across two problem settings and error types we have considered.

For future work, one interesting direction to consider is the dynamicity of the network (both on the edge and node rate of birth and retirement). Another interesting future direction would be to consider this problem from a criminal's perspective to analyze how criminals can deceive these learning algorithms. This analysis can then be used

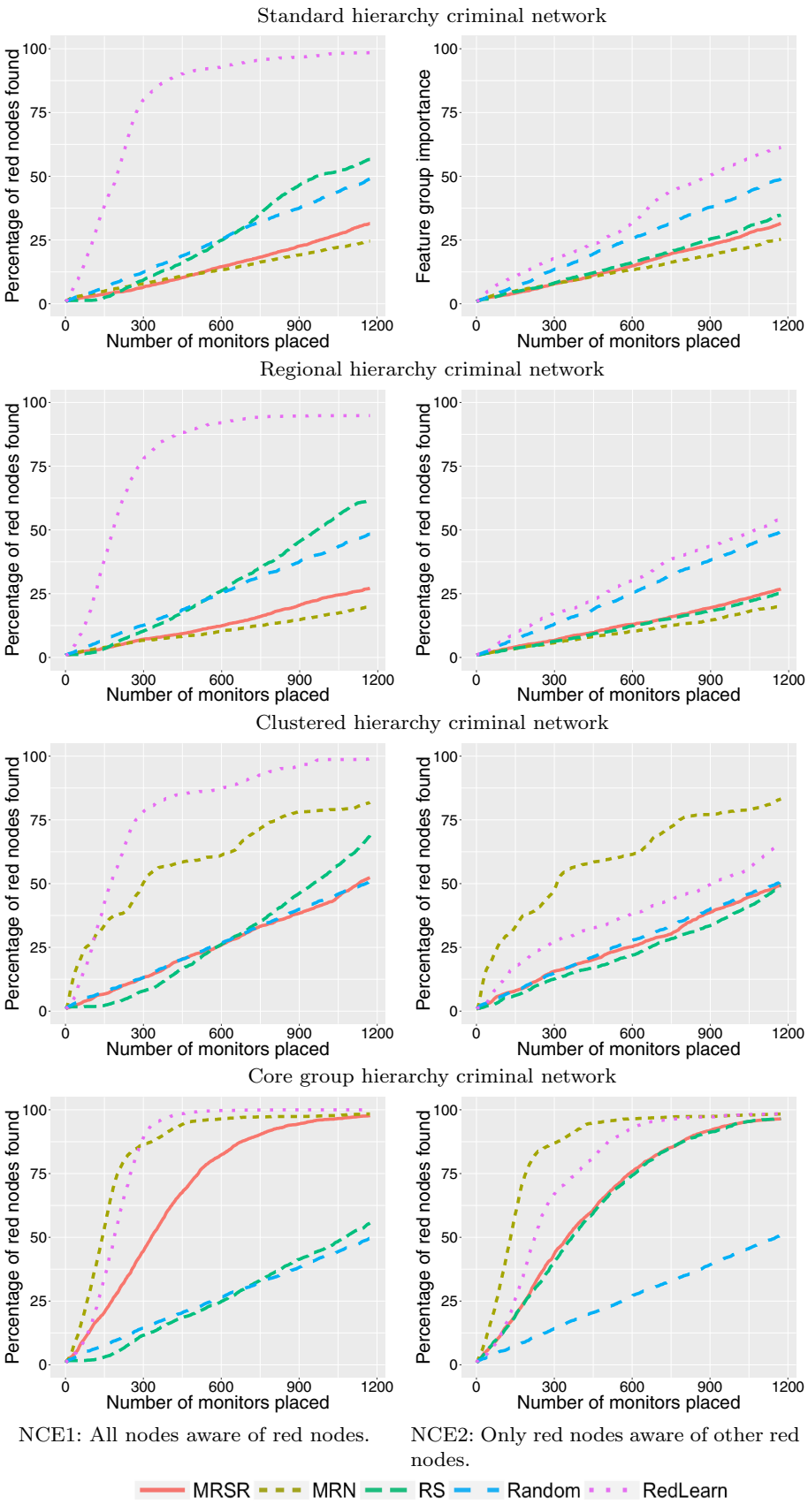

Fig. 14 Comparison of monitor placement algorithm performance for different criminal network hierarchies when reported node colors are reliable (Problem Setting 1). RedLearn performs the best in all network hierarchies in NCE 1. When red nodes are clustered and/or connected to lot of other red nodes (core group), MRN tends to perform well. Standard hierarchy criminal network. Regional hierarchy criminal network. Clustered hierarchy criminal network. Core group hierarchy criminal network. NCE1: All nodes aware of red nodes. NCE2: Only red nodes aware of other red nodes

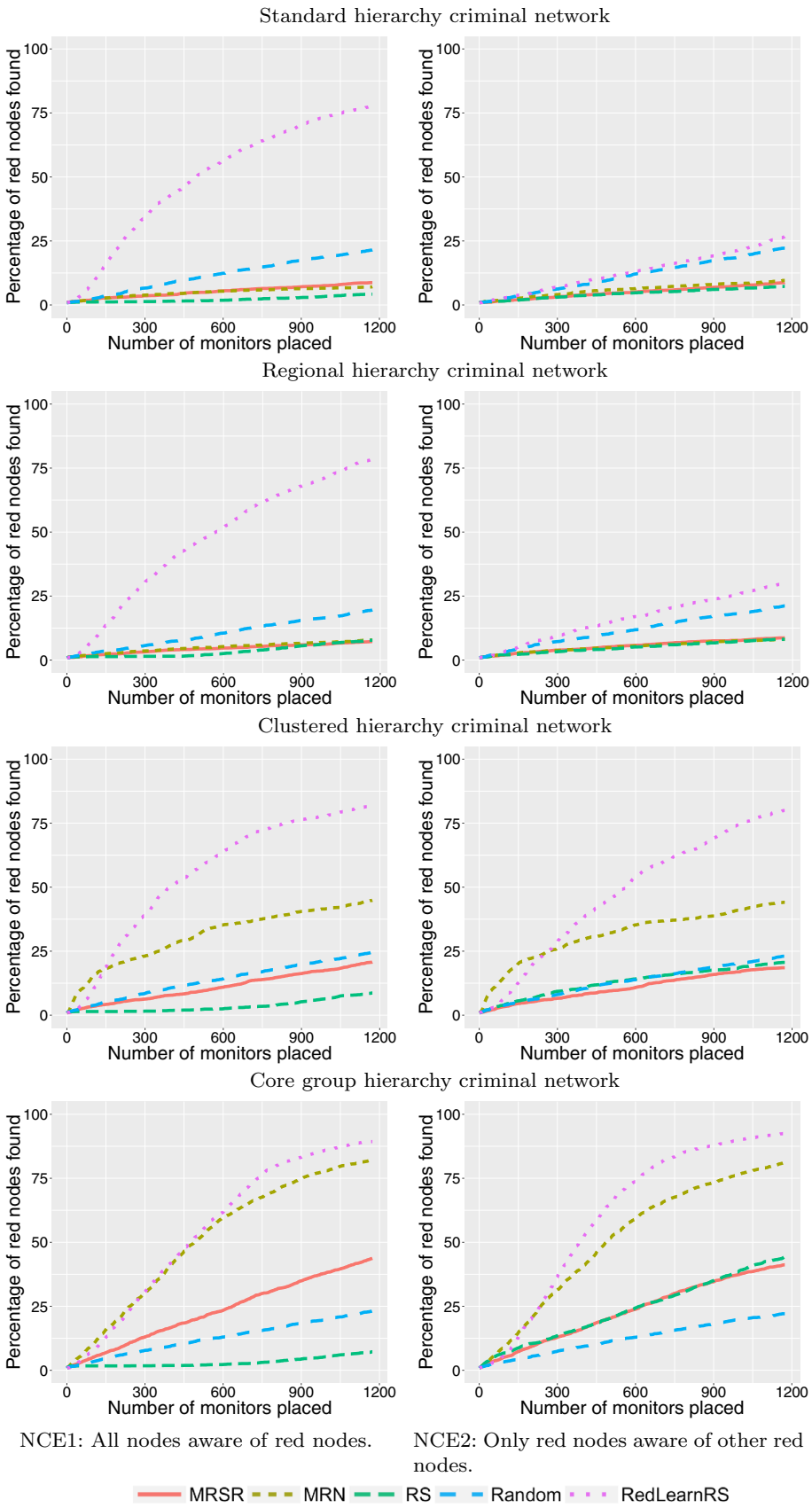

Fig. 15 Comparison of monitor placement algorithm performance for different criminal network hierarchies when node colors may be misreported by monitors (Problem Setting 2). RedLearnRS outperforms all other monitor placement strategies. Standard hierarchy criminal network. Regional hierarchy criminal network. Clustered hierarchy criminal network. Core group hierarchy criminal network. NCE1: All nodes aware of red nodes. NCE2: Only red nodes aware of other red nodes

to develop algorithms that are robust to deceptive criminal behavior.

Acknowledgements R. Gera thanks the DoD for partially sponsoring this work. This research was supported in part through computational resources provided by Syracuse University.

References

Adamic LA, Lukose RM, Puniyani AR, Huberman BA (2001) Search in power-law networks. Phys Rev E 64(4):046–135

Aldous D, Fill J (2002) Reversible markov chains and random walks on graphs. Berkeley

Asztalos A, Toroczkai Z (2010) Network discovery by generalized random walks. EPL (Europhys Lett) 92(5):50,008

Avrachenkov K, Basu P, Neglia G, Ribeiro B, Towsley D (2014) Pay few, influence most: Online myopic network covering. In: IEEE NetSciCom workshop

Baker WE, Faulkner RR (1993) The social organization of conspiracy: Illegal networks in the heavy electrical equipment industry. Am Sociol Rev 58(6):837–860

Benkherouf L, Bather J (1988) Oil exploration: sequential decisions in the face of uncertainty. J Appl Probab 25(3):529–543

Bhagat S, Cormode G, Muthukrishnan S (2011) Node classification in social networks. In: Social network data analytics, Springer, pp 115–148

Biernacki P, Waldorf D (1981) Snowball sampling: problems and techniques of chain referral sampling. Soc Methods Res 10(2):141–163

Bliss CA, Danforth CM, Dodds PS (2014) Estimation of global network statistics from incomplete data. PloS ONE 9(10):e108,471

Bnaya Z, Puzis R, Stern R, Felner A (2013) Social network search as a volatile multi-armed bandit problem. HUMAN 2(2):84

Burfoot C, Bird S, Baldwin T (2011) Collective classification of congressional floor-debate transcripts. In: Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-vol 1, Association for Computational Linguistics, pp 1506–1515

Carvalho VR, Cohen WW (2005) On the collective classification of email speech acts. In: Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, ACM, pp 345–352

Chen H, Chung W, Xu JJ, Wang G, Qin Y, Chau M (2004) Crime data mining: a general framework and some examples. Computer 37(4):50–56

Davis B, Gera R, Lazzaro G, Lim BY, Rye EC (2016) The marginal benefit of monitor placement on networks. In: Complex networks VII, Springer, pp 93–104

Erdos P, Rényi A (1960) On the evolution of random graphs. Publ Math Inst Hung Acad Sci 5(1):17–60

Friedman N, Koller D (2003) Being bayesian about network structure. a bayesian approach to structure discovery in bayesian networks. Mach Learn 50(1–2):95–125

Fronczak A, Fronczak P (2009) Biased random walks in complex networks: the role of local navigation rules. Phys Rev E 80(1):016–107

Gallagher B, Tong H, Eliassi-Rad T, Faloutsos C (2008) Using ghost edges for classification in sparsely labeled networks. In: Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, pp 256–264

Gera R, Miller R, MirandaLopez M, Warnke S, Saxena A (2017) Three is the answer: combining relationships to analyze multilayered terrorist networks. In: Advances in social networks analysis and mining (ASONAM), 2017 IEEE/ACM, IEEE

Hanneke S, Xing EP (2009) Network completing and survey sampling. In: AISTATS, pp 209–215

Holme P, Kim BJ (2002) Growing scale-free networks with tunable clustering. Phys Rev E 65(2):026–107

Hughes BD (1995) Random walks and random environments. Oxford, vol 2, 1995–1996

Koschade S (2006) A social network analysis of jemaah islamiyah: The applications to counterterrorism and intelligence. Stud Confl Terror 29(6):559–575

Le V (2012) Organised crime typologies: structure, activities and conditions. Int J Criminol Sociol 1:121–131

Leskovec J, Faloutsos C (2006) Sampling from large graphs. In: SIGKDD, ACM, pp 631–636

Lin F, Cohen WW (2010) Semi-supervised classification of network data using very few labels. In: 2010 International Conference on Advances in Social Networks Analysis and Mining (ASONAM), IEEE, pp 192–199

Lu Q, Getoor L (2003) Link-based classification. In: Proceedings of the 20th International Conference on Machine Learning (ICML-03), pp 496–503

Lu Y, Luo X, Polgar M, Cao Y (2010) Social network analysis of a criminal hacker community. J Comput Inf Syst 51(2):31–41

Macskassy SA, Provost F (2005) Suspicion scoring based on guilt-by-association, collective inference, and focused data access. In: International Conference on Intelligence Analysis

Maiya AS, Berger-Wolf TY (2010) Online sampling of high centrality individuals in social networks. In: PAKDD, pp 91–98

Michalak TP, Rahwan T, Wooldridge M (2017) Strategic social network analysis. In: AAAI, pp 4841–4845

Neville J, Jensen D (2000) Iterative classification in relational data. In: Proceedings of AAAI-2000 workshop on learning statistical models from relational data, pp 13–20

Noh JD, Rieger H (2004) Random walks on complex networks. Phys Rev Lett 92(11):118–701

Novikov AA, Shiryaev AN (2005) On an effective solution of the optimal stopping problem for random walks. Theory Probab Appl 49(2):344–354

Raab J, Milward HB (2003) Dark networks as problems. J Public Adm Res Theory 13(4):413–439

Roberts N, Everton S (2011) Terrorist data: Noordin top terrorist network. https://sites.google.com/site/sfeverton18/research/appendix-1

Schwartz DM, Rouselle TD (2009) Using social network analysis to target criminal networks. Trends Organ Crime 12(2):188–207

Sparrow MK (1991) The application of network analysis to criminal intelligence: an assessment of the prospects. Soc Netw 13(3):251–274

Stern RT, Samama L, Puzis R, Beja T, Bnaya Z, Felner A (2013) Tonic: Target oriented network intelligence collection for the social web. In: AAAI

Tsitsiklis JN, Van Roy B (1999) Optimal stopping of markov processes: Hilbert space theory, approximation algorithms, and an application to pricing high-dimensional financial derivatives. IEEE Trans Autom Control 44(10):1840–1851

Wijegunawardana P, Ojha V, Gera R, Soundarajan S (2017) Seeing red: locating people of interest in networks. In: Workshop on Complex Networks CompleNet, Springer, pp 141–150

Xiang R, Neville J, Rogati M (2010) Modeling relationship strength in online social networks. In: Proceedings of the 19th International Conference on World Wide Web, ACM, pp 981–990

Yan G (2013) Peri-watchdog: hunting for hidden botnets in the periphery of online social networks. Comput Netw 57(2):540–555

Zheleva E, Getoor L (2009) To join or not to join: the illusion of privacy in social networks with mixed public and private user profiles. In: Proceedings of the 18th International Conference on World Wide Web, ACM, pp 531–540

Zhu X, Ghahramani Z, Lafferty J et al (2003) Semi-supervised learning using gaussian fields and harmonic functions. ICML 3:912–919