Behavior-based Community Detection: Application to Host Assessment In Enterprise Information Networks

Cheng Cao*,†
Amazon Inc.
chengcao@amazon.com

Zhengzhang Chen*,#
NEC Laboratories America
zchen@nec-labs.com

James Caverlee
Texas A&M University
caverlee@cse.tamu.edu

Lu-An Tang
NEC Laboratories America
ltang@nec-labs.com

Chen Luo
Rice University
cl67@rice.edu

Zhichun Li
NEC Laboratories America
zhichun@nec-labs.com

ABSTRACT

Community detection in complex networks is a fundamental problem that attracts much attention across various disciplines. Previous studies have been mostly focusing on external connections between nodes (*i.e.*, topology structure) in the network whereas largely ignoring internal intricacies (*i.e.*, local behavior) of each node. A pair of nodes without any interaction can still share similar internal behaviors. For example, in an enterprise information network, compromised computers controlled by the same intruder often demonstrate similar abnormal behaviors even if they do not connect with each other. In this paper, we study the problem of community detection in enterprise information networks, where large-scale internal events and external events coexist on each host. The discovered host communities, capturing behavioral affinity, can benefit many comparative analysis tasks such as host anomaly assessment. In particular, we propose a novel community detection framework to identify behavior-based host communities in enterprise information networks, purely based on large-scale heterogeneous event data. We continue proposing an efficient method for assessing host's anomaly level by leveraging the detected host communities. Experimental results on enterprise networks demonstrate the effectiveness of our model.

CCS CONCEPTS

• **Information systems → Enterprise information systems**; **Data mining**; • **Security and privacy → Intrusion detection systems**; • **Theory of computation → Graph algorithms analysis**; • **Computing methodologies → Anomaly detection**; *Dimensionality reduction and manifold learning*;

KEYWORDS

Community detection; Anomaly Detection; Host assessment; Comparative analysis; Behavior modeling; Entity Embedding; Enterprise information network

ACM Reference Format:
Cheng Cao*,†, Zhengzhang Chen*,#, James Caverlee, Lu-An Tang, Chen Luo, and Zhichun Li. 2018. Behavior-based Community Detection: Application to Host Assessment In Enterprise Information Networks. In *The 27th ACM International Conference on Information and Knowledge Management (CIKM'18), October 22–26, 2018, Torino, Italy*. ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3269206.3272022

1 INTRODUCTION

Discovering community structures in complex networks has received wide attention in the past few years. The task of community detection is to find clusters of nodes that have tight affinities (either physical or virtual) *within* the same cluster and loose affinities *between* different clusters. Identifying communities in a complex system can shed light on the organization of the system and the functions of its individual members, which has become a fundamental problem in network science [10] with great impacts on the Web, recommendation systems, physical system, and online social media systems [1, 3, 4, 11, 28, 35, 36].

Naturally, the topology information of a network has inspired many community detection approaches that were derived from graph theory [3, 4, 8, 26]. These approaches, however, typically only consider the tightness of external links between nodes while ignoring the closeness between nodes in terms of their internal characteristics and properties. Network topology alone may fail to account for insightful partitions of a network. A modern example is the online social network, on which a common scenario is the family members from the real world "follow" each other, virtually forming a social community. In spite of their social connections, they can have very distinct profiles, interests, and daily activities, which hardly conclude they belong to the same community from their behavioral perspective.

Recently, several efforts began to combine network topology and node attribute for community detection [28, 33, 34, 37]. These approaches, however, require well-curated categorical attributes of each node as the input, which may not be available in many *enterprise information networks*.

Enterprise information networks are ubiquitous. A typical enterprise information network consists of a collection of *hosts* [9, 19].

*Both authors contributed equally.
†Work done during an internship at NEC Laboratories America.
#Corresponding author.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
CIKM'18, October 22–26, 2018, Torino, Italy
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-6014-2/18/10…$15.00
https://doi.org/10.1145/3269206.3272022

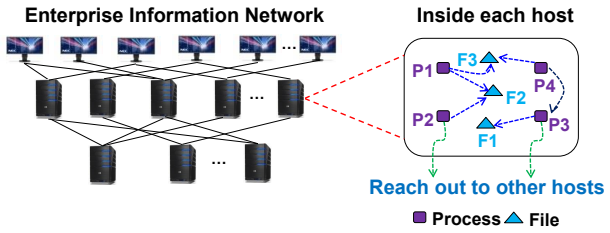


Figure 1: An example of enterprise information network. Nodes of enterprise system denote machines (hosts) and lines denote their communications via network-level events. Arrows inside the host denote the process-level events.

Hosts in the same community should share certain properties — some common cases include connectivity, shared roles, and IPs. The definition of community always depends on the problem domain, but those defined by traditional properties become less meaningful in an enterprise information network. For example, in most corporations, the network connectivity among all hosts is unimpeded. Also, the roles of hosts usually have already been known by the company. Thus, when it comes to host community in enterprise information network, we should take into account the unique characteristics of its data.

In a typical enterprise information network as shown in Fig. 1, hundreds of hosts incessantly generate operational data — usually called *events*. Such massive event data can be generally categorized into two types: *process-level events* and *network-level events*. Process-level events are locally generated on each host and reflect the internal behaviors of individual hosts (*e.g.*, a process *opens* a file or *forks* another process). Network-level events represent the external behaviors of interactions between different hosts, which can be viewed as globally connected pairs of hosts (*e.g.*, a process reaches out some other host). Together, these two types of events provide the opportunity for identifying communities in enterprise information networks, in the context of host behaviors. A key question is how to exploit both these types of host behaviors for community detection.

Unfortunately, we can not adopt those aforementioned approaches that combine network structure and node attributes for four key reasons. First, a host in an enterprise information network does not have well-curated attributes for our community detection task. Those basic attributes of a host (*e.g.*, hardware specs, host name, domain name, etc.) are often irrelevant to host behaviors. Also, a host is unnecessarily "bound" to certain persons. The owners of a host can dynamically change in an enterprise information network, *e.g.*, in a company usually there is a specific group of administrators who can access to all hosts. Second, the events can reflect a host's behavioral patterns, and yet we have meager prior knowledge of how to translate the "event space" to a "feature space" that can reflect a host's behavioral patterns. Third, even if we treat each unique event as an attribute for a host, the huge scale of events will make both the augmented graph and the generative model too complex. Finally, the statistical assumptions of distribution and dependency in those probabilistic models unnecessarily hold for hosts and events in a large-scale enterprise information network.

Therefore, given the massive host-level events in an enterprise information network, our first goal is to find the community membership for each host, in the context of host behavior. We propose to leverage both network-level events and process-level events *at the same time* and *in the same space*. In particular, we consider a network-level event as an interaction between two hosts and a process-level event as a connection between a host and an event. Given these observed pairwise interplays, we map hosts and events into the same latent space. The idea is to allow both types of events to seamlessly bridge among hosts so that we can better investigate the intricate behavioral patterns of each host, which leads to an accurate community detection model.

The discovered host community structure, which offers insights of behavioral affinity, ultimately can benefit many studies of comparative analysis in enterprise information networks. One important task is to assess the anomaly status of all hosts and identify those with suspicious behaviors. The key idea is that host's community membership, based on their massive behavioral histories, can reflect their behavioral normalcy. Now if we review the status of host behavior after a time period (*e.g.*, one week), we are supposed to see a host behaves similarly to its peers/community members. If not, it has a status of "suspicious behavior". On the other hand, if a host behaves distantly with its "non-community peers", it is unnecessarily abnormal, which can also mitigate the false positive issues in many circumstances of anomaly detection.

Hence, our second goal in this paper is to extend the identified host communities for the task of host anomaly assessment, in the context of host behavior. However, we are facing several challenges. For example, how to seamlessly exploit host's community membership information for host anomaly assessment? How to quantify the severity level of a host's status? And how to evaluate our approach? With all these questions in mind, we propose an efficient method that intelligently extends the community detection results, in both perspectives of host and event.

In summary, this paper makes the following contributions:

- We look into the problem of community detection in enterprise information networks, where the large-scale heterogeneous event data calls for a new angle on the traditional task.
- We propose an embedding framework to capture the host behaviors based on network-level events and process-level events.
- We further propose an efficient method to measure the suspiciousness of a host, based on the identified host communities through our embedding framework.
- Our empirical studies on a real enterprise network demonstrate the effectiveness of our method.

2 PRELIMINARIES

Network-level event. A network-level event can be defined as a multiple-tuple. More specifically, in our data, a network-level event is a seven-tuple ⟨src-ip, src-port, dst-ip, dst-port, connecting-process, protocol-num, timestamp⟩, where src-ip and src-port are the IP address and port of source host, and dst-ip and dst-port are the IP and port of destination host. Connecting-process is the

process that initializes the connection, protocol-num indicates the protocol of connection, and timestamp records the connection time.

Process-level Event. A process-level event can also be defined as a multiple-tuple. More specifically, in our data, a process-level event is a five-tuple ⟨host-id, user-id, process, object, timestamp⟩, where host-id indicates the host where the agent is installed, user-id identifies the user who runs the process, timestamp records the event time, process is the subject of the event, and object is the object of the event. The object can be a file, another process, or a socket that contains the connection information. According to the object type, the process-level events can be further classified into three categories: the process-file events, the process-socket events, and the process-process events.

Problem 1: Host Community Detection. Given a set of m hosts $H = \{h_0, h_1, ..., h_{m-1}\}$, a time window T, and a set of n events $E = \{e_0, e_1, ..., e_{n-1}\}$ collected from all hosts in H within T. The goal is to automatically identify non-overlapped community membership L for all hosts in H.

Problem 2: Host Anomaly Assessment. Besides the inputs of Problem 1, given the identified community membership L for all the m hosts in H, the goal is to find a function $f : H \rightarrow \mathcal{R}$ such that $\forall h_i \in H$, f returns an anomaly score $f(h_i) \in \mathcal{R}$ that assesses the behavioral status of host h_i.

3 HOST COMMUNITY DETECTION

In this section, we propose an embedding based approach to model the host community detection problem into a unified optimization framework.

Finding communities is essential to gain insights of pairwise affinities. Hence, for our problem, the first task naturally is to model every host's behavioral patterns. However, the input data are only those historical events collected from individual hosts. It is challenging to accurately capture the host behavior purely based on its historical events. So, we first propose to model the host-level behavior inside each host based on the corresponding historical events. Then, given the host-level behavioral modeling, we identify groups of hosts that share similar behaviors by modeling the host structures. Finally, we propose a unified optimization model that takes consideration of both models.

3.1 Modeling Host-level Behavior

We first model the host-level behavior. Given a host $h \in H$, we get a set of n events $E_h = \{e_0, e_1, ..., e_{n-1}\}$ monitored from h, including both process-level events and network-level events. For the set of process-level events $\{e_0, e_1, ..., e_{i-1}\}$, it can be seen as a collection of i pairs $\{(h, e_0), (h, e_1), ..., (h, e_{i-1})\}$. And for the set of network-level events $\{e_i, e_{i+1}, ..., e_{n-1}\}$, it can be rewritten as a collection of $n - i$ triples $\{(h, e_i, \hat{h}_0), (h, e_{i+1}, \hat{h}_1), ..., (h, e_{n-1}, \hat{h}_j)\}$, where $\{\hat{h}_0, \hat{h}_1, ..., \hat{h}_j\}$ is the set of hosts that are reached by h's network-level events. As aforementioned in Section 2 that an event always contains the timestamp attribute. In order to capture a host's long-term behavioral patterns, however, we need to exclude the timestamp attribute, otherwise, every event is unique and we lose the opportunity of studying any repeating behavioral pattern based on events. Next, a key task is to find a way of leveraging these two different types of events to model the host behavior, at the same time and in the same space.

Recent advances in distributed text representation learning have offered effective text embedding methods [21] that capture syntactic and semantic word relationships. Inspired by their breakthrough, our idea is to model a host as the context of all its process-level events and as the context of all hosts reached by its network-level events. By doing so, we integrate both types of events into one model at the same time. Next, we embed all pairs of hosts and events into a common latent space, where their interplays are all preserved in the same space.

Concretely, we design a neural probabilistic model that is trained using the maximum likelihood principle. A process-level event e on the host h can be modeled as the conditional probability of host h given event e, *i.e.*, the probability that e is observed from h, via the following Softmax function:

$$P(h|e) = \frac{\exp(v_h \cdot v_e)}{\sum_{\hat{h} \in H} \exp(v_{\hat{h}} \cdot v_e)}, \tag{1}$$

where v_h and v_e are the embedding vectors for the host h and the event e, respectively. H is the set of hosts.

Similarly, a network-level event can be modeled as $P(\hat{h}|h, e)$ — the conditional probability of a host $\hat{h}$ given a host h and a network event e, *i.e.*, the probability that host h issues a network-level event e that connects to host $\hat{h}$. We can compute this probability as:

$$P(\hat{h}|h, e) = P(\hat{h}|h) = \frac{\exp(v_{\hat{h}} \cdot v_h)}{\sum_{h_i \in H} \exp(v_{h_i} \cdot v_h)}, \tag{2}$$

where v_h and $v_{\hat{h}}$ are the embedding vectors for the hosts h and $\hat{h}$, respectively. H is the set of hosts.

Given the whole set of n events $E = \{e_0, e_1, ..., e_{n-1}\}$ collected from all hosts in H, we would like to learn the embedding vectors for every host $h \in H$ and the events $E_h \subseteq E$ generated on it, so that we maximize:

$$\sum_{h \in H, e \in E_h} P(h|e) + P(\hat{h}|h, e).$$

This optimization model has been known to be very expensive to solve, due to the denominators of both equations summing over all hosts of H — we have to compute and normalize each probability for each host at every training step.

Therefore, we follow the idea of *negative sampling* [21] to address this challenge. Negative sampling follows a very similar idea of *noise-contrastive estimation (NCE)* [12]. In general, to avoid dealing with too many hosts to be iterated over, we only update for a sample of them. Of course, we should consider all those co-occurring host-event pairs observed from our input data, and we need to artificially sample a few "noisy pairs" — they are not supposed to co-occur so the conditional probabilities among them should be low. Hence, negative sampling offers an approximate update for the Softmax function, but it is computationally appealing, since the calculation now scales with the size of noise only.

In the end, after taking the logarithm and employing negative sampling, we aim to minimize the following two objective functions:

$$O_1 = -\frac{1}{|D_P|}\Big[\sum_{(h,e) \in D_P} \log \sigma(v_h \cdot v_e) + \sum_{(h',e') \in D'_P} \log \sigma(-v_{h'} \cdot v_{e'})\Big] \tag{3}$$

$$O_2 = -\frac{1}{|D_N|}\Big[\sum_{(h,\hat{e},\hat{h})\in D_N} \log\sigma(v_h \cdot v_{\hat{h}}) + \sum_{(\dot{h},\ddot{e},\ddot{h})\in D'_N} \log\sigma(-v_{\dot{h}} \cdot v_{\ddot{h}})\Big]. \tag{4}$$

In the above equations, σ is the logistic function, D_P is the collection of pairs of process-level events, and D_N is the set of triples of network-level events. D'_P and D'_N are the two sets of negative samples constructed by certain sampling scheme for process-level events and network-level events, respectively.

3.2 Modeling Host Community Structure

To solve the problem of host community detection, our goal is to find behavior-based communities for all hosts. A good community detection result is determined by several factors: First, since the learned embedding vectors of hosts by modeling host-level behaviors are their latent behavioral representations in the same space, the community detection result should be consistent with the similarities computed between the learned embedding vectors; Second, the communities that are discovered should be a set of hosts that have more connections (more similar) inside than outside. To meet these requirements, we propose an expectation-maximization (EM) based model as follows:

$$O_c = \sum_{j=0}^{K-1} \sum_{h\in H} \left\|v_h - c_j\right\|^2, \tag{5}$$

where c_j is the community centroid for community j, v_h is the embedding vector for the host h, and K is the number of communities.

3.3 The Unified Model and Learning Algorithm

By putting all the above models (Eq. 3, Eq. 4, and Eq. 5) together, we get the final objective function as follows:

$$\begin{aligned} O_u &= O_c + O_1 + O_2 \\ &= \sum_{j=0}^{K-1} \sum_{h\in H} \left\|v_h - c_j\right\|^2 \\ &- \frac{1}{|D_P|}\Big[\sum_{(h,e)\in D_P} \log\sigma(v_h \cdot v_e) + \sum_{(h',e')\in D'_P} \log\sigma(-v_{h'} \cdot v_{e'})\Big] \\ &- \frac{1}{|D_N|}\Big[\sum_{(h,\hat{e},\hat{h})\in D_N} \log\sigma(v_h \cdot v_{\hat{h}}) + \sum_{(\dot{h},\ddot{e},\ddot{h})\in D'_N} \log\sigma(-v_{\dot{h}} \cdot v_{\ddot{h}})\Big]. \end{aligned} \tag{6}$$

By minimizing this equation, we can get the community centroids $C = \{c_1, c_2, ..., c_K\}$.

There are two sets of parameters in Eq. 6: (1) the embedding vectors of hosts and events V_h, V_E, and (2) community centroids $C = \{c_1, c_2, ..., c_K\}$. Thus, we design a two-step iterative learning method, where the embedded vectors V_H, V_E and the community centroids C mutually enhance each other. In the first step, we fix the community centroids C, and community assignment $l(V_H)$, and learn the best embedded vectors V_H, V_E. In the second step, we fix the predicted embedded vectors V_H, V_E and learn the best C and $l(V_H)$. The overall steps of the proposed host community detection algorithm are shown in Algorithm 1. Notice that, in the algorithm, we have two updating processes.

Algorithm 1: Host Community Detection

Input: Process-level events D_P, Network events D_N. Parameters k_P and k_N. Step size η. Mini-batch sizes b_P and b_N.
Output: Learned embedding vectors $\mathbf{V_H}$ and $\mathbf{V_E}$. Community membership $l(V_h)$ and community centroids C.

```
Initialize V_H, V_E, l(V_h), C randomly;
while not converged do
    while not reaching the inner l(V_h), C difference threshold do
        Updating l(V_h), C by fixing V_H, V_E.
    while not reaching the inner V_H, V_E difference threshold do
        Updating V_H, V_E by fixing l(V_h), C.
return V_H, V_E, l(V_h), C
```

Algorithm 2: Updating V_H and V_E Given C and $l(V_H)$

Input: Collection of process-level events D_P; Collection of network events D_N; Two parameters k_P and k_N controlling the sizes of negative samples; Step size η; Mini-batch sizes b_P and b_N;
Output: Learned embedding vectors $\mathbf{V_H}$ and $\mathbf{V_E}$

1 Initialize host embedding vectors $V_H = \{v_h | \forall h \in H\}$ and event embedding vectors $V_E = \{v_e | \forall e \in E\}$, $t = 0$
2 **while** *not converged* **do**
3 Choose mini-batch $D_{b_P} \subset D_P$ of size b_P and $D_{b_N} \subset D_N$ of size b_N, uniformly at random
4 Generate two sets of negative samples D'_{b_P} and D'_{b_N}, according to a noise distribution and k_P and k_N
5 Let $O(V_H, V_E) = -\frac{1}{b_P}\Big[\sum_{(h,e)\in D_{b_P}} \log\sigma(v_h \cdot v_e) + \sum_{(\hat{h},e')\in D'_{b_P}} \log\sigma(-v_{\hat{h}} \cdot v_{e'})\Big] - \frac{1}{b_N}\Big[\sum_{(h,e,\hat{h})\in D_{b_N}} \log\sigma(v_h \cdot v_{\hat{h}}) + \sum_{(\dot{h},\ddot{h})\in D'_{b_N}} \log\sigma(-v_{\dot{h}} \cdot v_{\ddot{h}})\Big]$
6 Update $V_H^{(t+1)} \leftarrow V_H^{(t)} - \eta\nabla O_u(V_H^{(t)})$
7 Update $V_E^{(t+1)} \leftarrow V_E^{(t)} - \eta\nabla O_u(V_E^{(t)})$
8 $t = t + 1$
9 **return** V_H, V_E

The process of updating V_H and V_E, given C and $l(V_h)$, is shown in Algorithm 2. Given two sets of data samples D_P and D_N, in each iteration, we uniformly sample two mini-batches of process-level events and network events, D_{b_P} and D_{b_N} (step 3). Then, we generate a set of negative samples D'_{b_P} for each data sample from D_{b_P} and a set of negative samples D'_{b_N} for each data sample from D_{b_N}, which are drawn according to a noise distribution and two parameters controlling the sizes of negative samples k_P and k_N

(step 4). Then, mini-batch gradient descent over $\mathbf{V_H}$ and $\mathbf{V_E}$ is used to minimize the objective function $O(\mathbf{V_H}, \mathbf{V_E})$ until converged (step 5-8). In the end, Algorithm 2 returns the embedding vectors of all hosts and events. We follow the ideas in most existing embedding learning work [15, 21, 29] to empirically tune a few parameters. Finally, we set the embedding dimension as 32, the number of negative samples as 5, the batch size for process-level events as 128, and the batch size for network-level events as 64 (b_P and b_N in Algorithm 2, respectively). The initial value of η is 0.1 and it linearly drops until it reaches 0.0001.

Algorithm 3 shows the details of updating C and $l(V_h)$, given V_H and V_E. It essentially shares a similar idea with the expectation-maximization (EM) algorithm. The inputs include the learned embedding vectors for all hosts returned by Algorithm 2 and an integer K as the number of host communities to be found.

First, we select K hosts at random as the centers (centroids) of communities (step 1; details of determining K are described in Section 5). Then, we assign every host to its closest group centroid according to the Euclidean distance function (step 2-3). Next, we recalculate the centroid of all hosts in each group (step 5-8). The values of the centroids are updated, taken as the geometric mean of the points that have the same label as the centroid's (step 9-13). We repeat iteratively until all the hosts (vectors) can no longer change groups. The algorithm outputs a set of labels corresponding to the community membership of each host.

Algorithm 3: Updating C and $l(V_h)$ Given V_H and V_E

Input: Host embedding vectors $\mathbf{V_H}$, returned by Algorithm 2; The number of communities K

Output: Host community membership $L = \{l(v_h)|\forall v_h \in V\}$

1 Initialize the set of community centroids $C = \{c_1, c_2, ..., c_K\}$ by randomly selecting K elements from V
2 **for** *each* $v_h \in \mathbf{V_H}$ **do**
3 $l(v_h) = argmin_{j\in\{1,...,K\}}||v_h - c_j||^2$
4 $changed = false$
5 **while** $changed = false$ *and not converged* **do**
6 **for** *each* $c_j \in C$ **do**
7 $S_j = \{v_h|l(v_h) = j\}$
8 $c_j = \frac{1}{|S_j|}\sum_{v_h\in S_j} v_h$
9 **for** *each* $v_h \in \mathbf{V_H}$ **do**
10 $minDist = argmin_{j\in\{1,...,K\}}||v_h - c_j||^2$
11 **if** $minDist \neq l(v_h)$ **then**
12 $l(v_h) = minDist$
13 $changed = true$
14 **return** L, C

4 HOST ANOMALY ASSESSMENT

How to intelligently exploit the obtained community membership information for host anomaly assessment? And how to quantify the severity level of a host's behavioral status? We look into two perspectives: host and event.

From the host perspective, we can see if a host's behavior gets changed from its past status via referring to its behavioral community peers that are found in the past. We propose the behavioral anomaly score $g_h(h)$ of a host h as follows:

$$g_h(h) = 1 - \frac{|S(h) \cap S'(h)|}{|S(h) \cup S'(h)|}, \tag{7}$$

where $S(h)$ is the set of community peers found in the past, and $S'(h)$ is the set of community peers identified later on. Both $S(h)$ and $S'(h)$ can be computed through Algorithm 1.

From the event perspective, given a set of events collected from one host, since we also have obtained the embedding vectors of all events via Algorithm 1, we can compute how different the events on that host and the events on its community peers are. This can be computed by:

$$g_e(h) = 1 - \frac{1}{|E_h||S(h)|}\sum_{e\in E_h}\sum_{h'\in S(h)}\max_{e'\in E_{h'}}\cos(e, e'), \tag{8}$$

where E_h is the set of events monitored from host h. $\cos(e, e')$ is the similarity between the embedding vectors of event e and event e'. In particular, for each event on a host h, we compute the distance between it and its closest event on each community peer of h, in terms of the cosine similarity. Then, we take the average over all peer hosts. Finally, we take the average of all events from one host as the returned score for this host.

We combine those two together and have the following:

$$g(h) = \alpha g_h(h) + \beta g_e(h), \tag{9}$$

which returns the anomaly score of host h. The two parameters α and β are weighting factors that indicate the contributions of host perspective and event perspective, respectively.

5 EXPERIMENTS

Data. We collect data from a real enterprise information network of an Information Technology company. The data is similar to the example shown in Fig. 1. Within three weeks, we collect more than 60 million system events (including 59.6 million process-level events and 0.5 million network-level events (see Section 2 for event data description)) from 71 machines. We treat Windows and Linux separately, since hosts with different operating systems can have distinct behavioral patterns (*e.g.*, process, port, protocol and *et al.*). Hence, if a host installs both operating systems, we consider it as two hosts with distinct host behaviors.

5.1 Host Community Detection

Experiment Setup. Determining the optimal number of communities is a fundamental issue in practice. Unfortunately, there is no rule of thumb addressing this issue. In most cases, domain knowledge is needed, and yet such information is rarely available in reality. One simple attempt is to inspect the dendrogram generated by a hierarchical clustering alike algorithm. However, that still calls for a pre-determined distance threshold in order to produce the host communities we need.

In this work, we adopt three popular approaches that mitigate this issue and suggest the number of communities to detect:

- Average silhouette method [27]. The silhouette coefficient of a data instance measures how closely it is matched to data within its community (cluster) and how loosely it is matched to data of its neighboring community. Average silhouette method computes the average silhouette score over all observations for different numbers of communities. The estimate of the best is the one that maximizes the average silhouette over a range of possible values. In concrete, silhouette coefficient for one sample is $\frac{D_1-D_0}{\max(D_0,D_1)}$ where D_0 is that sample's mean intra-cluster distance, and D_1 is the mean distance between this sample and the nearest cluster it does not belong to.
- Gap statistic method [30]. Gap statistic method compares the total within intra-cluster variation with the expected variation under null reference distribution of the data, for different numbers of clusters. The estimate of the best is the one that yields the largest gap statistic — when the clustering structure is far away from the random uniform distribution of all data instances.
- G-means method [13]. G-means is based on a statistical test for the hypothesis that a subset of data follows a Gaussian distribution. In particular, it repeatedly tests whether the data in the neighborhood of a cluster centroid looks Gaussian, and if not it splits the cluster.

Each of these three methods can automatically return a suggestion for the number of communities. In some cases, however, these suggestions may not agree with each other. We take the "majority vote" if it exists. Otherwise, we need a manual inspection and elbow method [17] is one of the most popular methods. Elbow method looks at the total within-cluster sum of square (WSS) as a function of the number of clusters. One would like to choose a number of clusters so that adding another cluster does not improve much better the total WSS. This heuristic approach usually tries to find a "elbow criterion" when plotting WSS against the number of clusters.

To evaluate the performance of the proposed model, we need the ground truth of host communities. Such ground truth does not exist and there is no general rule for preparing the ground truth. Thus, we inquire with three security experts from the company, where we collect the data. These experts determine which hosts are supposed to behave similarity, based on the user and the role of each host. For example, the hosts of the department administrators and HRs from different departments are labeled as the Operating community. And the 8 servers, running the data analytic application, are labeled as the Analytic community. Note that this manual labeling serves only for evaluation purpose. Labeling for all hosts is an extremely expensive task, even for domain experts. This motivates us to design a data-driven approach purely based on massive system monitoring data. The proposed framework is completely unsupervised and assumes no partial information of host community is known.

Evaluation Metrics. We pick four popular metrics in community detection evaluation — *purity*, *normalized mutual information (NMI)*, *adjusted mutual information (AMI)*, and *adjusted rand index (ARI)* [10, 20, 31].

The purity score is calculated through the function:

$$purity(\Omega, G) = \frac{1}{N}\sum_k max_j |w_k \cap g_j| \tag{10}$$

where $\Omega = \{w_1, ..., w_k\}$ is the set of detected labels and $G = \{g_1, ..., g_j\}$ contains the true labels, *i.e.*, ground truth.

The formula to compute NMI is:

$$NMI(\Omega, G) = \frac{I(\Omega; G)}{\sqrt{S(\Omega)S(G)}} \tag{11}$$

where I is the mutual information and S is the entropy.

An extension of NMI is AMI. AMI is proposed more recently and normalized against chance through using the expected value of I:

$$AMI = \frac{I - E[I]}{\max(S(\Omega), S(G)) - E[I]}. \tag{12}$$

And the ARI is computed via

$$ARI = \frac{RI - E[RI]}{max(RI) - E[RI]}. \tag{13}$$

RI is the raw (unadjusted) Rand Index given by $RI = (a + b)/C_2^N$, where C_2^N is the total number of possible host pairs; a is the number of host pairs that belong to the same community in both ground truth and returned result; and b is the number of host pairs that are in different communities in both ground truth and returned result.

Baselines. Our problem setting includes both process-level and network-level events. Traditional community detection methods can not handle process-level events in enterprise information networks. Recent work have started combining network topology and note attribute, but they require that each host has well-curated attributes that are not available in our data (see more discussions in Section 6). Fortunately, the network-level events naturally form a graph in which an edge connects from a source host to a destination host. Hence, we build such a directed graph, where edges are weighted by event frequency, and apply three classic graph-based community detection algorithms, as baselines. We add two more baselines by considering two variants of our proposed framework.

- Optimal Modularity (OM) [23]. This method computes the optimal modularity score of the graph and the corresponding community structure by solving a large integer optimization problem via linear programming.
- Random Walk (RW) [26]. This random walk based algorithm finds the community structure of the graph based on the idea that short random walks tend to stay in the same community.
- Leading Eigenvector (LE) [24]. This algorithm recursively splits the graph into two components according to the leading eigenvector of the graph's modularity matrix.
- Network-only. It considers only network-level events in our proposed framework, *i.e.*, excluding O_1 in Eq. 6.
- Process-only. It considers only process-level events in our proposed framework, *i.e.*, excluding O_2 in Eq. 6.

Results. The evaluation results are shown in Table 1. Our method surpasses other alternatives across all four evaluation metrics. In concrete, our approach can achieve at least 0.95 in terms of any metric whereas the baselines perform no more than 0.83 in three metrics. Our proposed framework also works better when considering either network-level events only or process-level events only. All improvements here are significant, validated by statistical testing. LE and OM, both derived from modularity maximization, have

close performances in all cases. The random walk based model does not work well in our problem setting. All these findings demonstrate the effectiveness of our proposed approach.

Table 1: Performances of host community detection on four metrics

	Proposed	Network-only	Process-only	LE	OM	RW
Purity	**.982**	.904	.927	.894	.895	.737
NMI	**.966**	.893	.918	.825	.820	.719
AMI	**.957**	.796	.814	.772	.770	.584
ARI	**.959**	.733	.801	.657	.641	.493

5.2 Host Anomaly Assessment

Experiment Setup. We need to prepare the ground truth of "abnormal hosts" to evaluate the proposed approach. Unfortunately, in reality, compared to normal events the intrusion attacks rarely happen. In one single enterprise network, real attacks happen only several times in a year. Therefore, we want to make the evaluation as close to real scenarios as possible. We internally build the "attack testbed", and randomly pick a few hosts as targets on which different types of attacks are injected. In particular, we collaborate with an industrial company working on commercial enterprise security products. The attacks were performed by professional hackers hired by the company. We choose six typical attacks in the following:

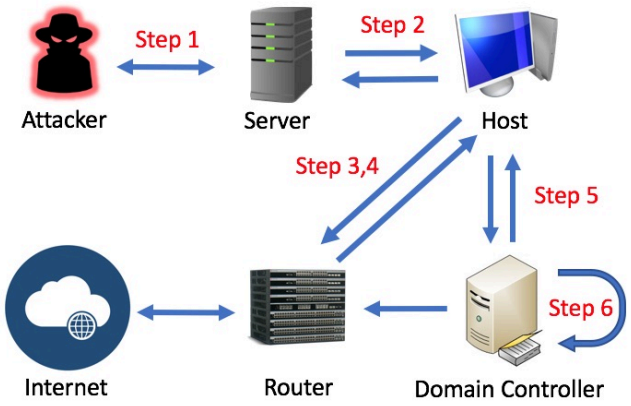


Figure 2: Attack testbed example related to the Diversifying Attack Vectors attack

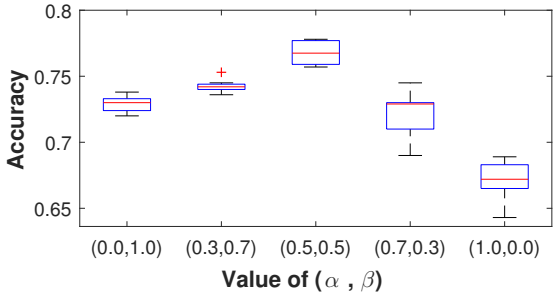
- *Diversifying Attack Vectors.* This intrusion scenario is a six-step attack chain (Fig. 2). First, hackers create malicious php files, download malware binary (*trojan.exe*), and connect back to them. Then, they run the process *notepad.exe* to perform DLL injection. Next, they use *mimikatz* and *kiwi* to perform memory operation inside the *meterpreter* context. Finally, they copy and run *PwDump7.exe* and *wce.exe* on target hosts.
- *Emulating Enterprise Environment.* This intrusion includes seven steps. First, attackers generate telnet processes to create malware binary, open reverse connection, and download malware binary (*trojan.exe*). Then, *trojan.exe* is created to connect back to hackers, and DLL is injected through the running process *notepad.exe*. The hackers use *mimikatz* and *kiwi* for memory operation inside the *meterpreter* context. Finally, malware *PwDump7.exe* and *wce.exe* are copied and run on target hosts.
- *Domain Controller Penetration.* In this five-step attack chain, the hackers first send an email attaching a document that includes the malware *python32.exe*. This malware opens a connection back to hackers so that they can run *notepad.exe* and perform reflective DLL injection to obtain needed privileges. Then, they transfer password enumerator and run the process *gsecdump-v2b5.exe* to get all user credentials. Finally, they probe the SQL server address and dump the database into their own bases.
- *MLS Attack (MLS)*: This attack targets at the */selinux/mls* file, which defines the Multi-Level Security(MLS) classification of files within the host. In general, the */selinux/mls* file should be kept secret to all users except for security administrator, as it exposes security rules of a computer system and enables attackers to find potential vulnerabilities. By the intrusion attack, attackers first exploit the *ssh* process to access */selinux/mls* file. If the file access is successful, file content is sent to attacker's hosts.
- *Snowden Attack (SNO)*: This attack targets at the */etc/passwd* file, which stores the password digest of all users as well as user group information. First, the attacker tries to access */etc/passwd* file by *gvfs* process, which enables easy access from a remote host via FTP. Then the attacker tries to send the file via an *INETSocket*.
- *Botnet Attack (BOT)*: In this attack, the remote intruder employs the *bash* process to scan a sensitive file */var/log/apt/history.log*. This file stores detailed installation messages. Attackers are interested in it as they can exploit the vulnerabilities of installed softwares. The sensitive information is leaked via an *INETSocket*.

We consider all events within the first two weeks as the "long-term normalities" upon which we run Algorithm 1. Then, we introduce those injections discussed above into the events within the third week. We work with domain experts from that security company in order to reflect the real attack scenarios as much as possible. For example, those six types of attacks are sampled by different probabilities based on how often they appear in real intrusion attacks. In the meanwhile, the number of infected normal events is determined by how many events usually get impacted by certain type of intrusion attack.

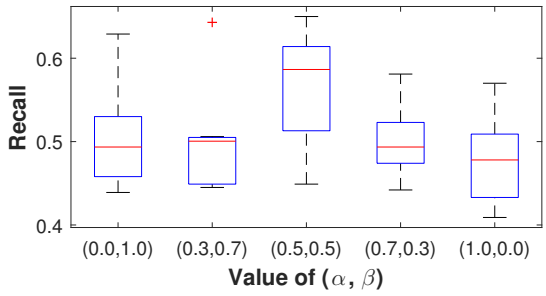
Results. Fig. 3 shows the performances of our algorithm for the task of host anomaly assessment. In particular, our algorithm returns every host an anomaly score and we rank them to find out how many compromised hosts are in the top list. The size of this top list is determined by how many hosts are picked and attacked in our real test. We choose the overall accuracy and the recall as two metrics. The recall metric reflects the false positive, which is a common point of interest (and a big challenge) among many anomaly detection applications.

We repeat our experiment ten times and each time we randomly pick target hosts. As shown in Eq. 9, we have two parameters tuning the contribution weights. Thus, we try their different combinations and find that considering both perspectives of host and event equally gives the best performance in terms of both metrics

(as shown in Fig. 3). We can achieve very promising results — 0.768 accuracy and 0.567 recall. The confidence intervals here are all based on the 95% confidence level. All these observations show that (1) our host community detection method can help identifying hosts with suspicious behaviors; and (2) we get benefit from considering both perspectives of host and event when assessing the host anomaly status.



(a) Accuracy



(b) Recall

Figure 3: Performance of host anomaly assessment under different values of α and β.

We have seen in Table 1 that our approach outperforms other three baselines for the task of host community detection. Now for the second task of host anomaly assessment, we find that many traditional anomaly detection methods can not perfectly handle our inputs — a collection of hosts with massive events generated on them. Most of those approaches focus on either numerical data or in supervised settings, and yet our problem calls for an unsupervised solution on heterogeneous categorical events data. Thus, we first run all those host community detection algorithms shown in Table 1 and then apply our solution of host anomaly assessment. We also compared our algorithm with the state-of-the-art intrusion detection algorithm (APE) [2]. APE detects abnormal events via probabilistic pairwise interaction and entity embedding. The host with any abnormal events identified by APE would be labeled as abnormal host. Table 2 gives the comparison on accuracy and recall. Still, ours works better than the other baselines, and behavior-based community detection can help reduce the false positives compared to event-based intrusion detection algorithm. And we validate the gaps are significant by statistical testing. For instance, the accuracies of LE and OM are barely around 0.6 and RW's is as low as 0.4, and their recalls only span between 0.3 and 0.4, while APE achieves a high recall 0.5 but low accuracy 0.6, because it suffers from the high false positive rate.

Table 2: Results of host anomaly assessment

	Proposed	Network-only	Process-only	LE	OM	RW	APE
Accuracy	**.768**	.658	.673	.607	.594	.410	.627
Recall	**.567**	.412	.451	.405	.349	.283	.503

Summary. We evaluate the proposed host community detection algorithm on a real enterprise information network, where more than 60 million events are collected within a three-week period. The results show our approach significantly outperforms other baselines, consistently across four popular metrics. Then, for the task of host anomaly assessment, we design a scoring scheme to assess the host behavioral status based on both perspectives of host and event. The results demonstrate quite encouraging potentials that our proposed framework can benefit the application of host anomaly assessment in real enterprise information networks.

6 RELATED WORK

Community Detection. One classic thread of community detection algorithms was inspired by network topology information and derived based on graph theory [4, 26]. For example, Clauset *et al.* proposed a greedy algorithm that maximizes the modularity score of the graph [8]. Newman recursively splits the graph according to the leading eigenvector of the modularity matrix [24]. In a related direction, algorithms based on random walks have become increasingly popular for detecting communities [26, 32]. These approaches, however, typically only consider the tightness of external links between nodes while ignoring the closeness between nodes in terms of their internal characteristics and properties.

Recently, several efforts have begun to combine network topology and node attribute for community detection [18, 28, 33, 34, 37]. For example, Zhou *et al.* used a random walk based algorithm on an augmented attributed graph [37]. Xu *et al.* found a probabilistic model that defines a joint probability distribution over all possible clusterings and attributed graphs [33]. Yang *et al.* also proposed a probabilistic generative model including both network structures and node attributes [34]. These methods, however, require the well-curated categorical attributes of each node as the input, which may not be available in many enterprise information networks.

Anomaly Detection. Existing anomaly detection approaches in large-scale enterprise network systems have been separately considering different data representations. In particular, host-based anomaly detection methods locally extracted patterns from process-level events as the discriminators of abnormal intrusion [9, 14, 22]. In contrast, network-based anomaly detection methods focused on disclosing abnormal subgraph structures from network-level events, most of which were inspired by graph properties [16, 25]. All these related work, however, did not take into account both types of events together and did not involve anything about the task of community detection. Indeed, we have seen some studies took care of both community detection and anomaly detection in

the same time[4–7, 11]. However, we find we cannot directly adopt them into our problem setting with two main reasons. First, we need a unified framework that can handle both process-level and network-level events together. Second, according to the scale of our data, it becomes extremely difficult to run on a matrix/graph structure with millions of columns/nodes.

7 CONCLUSION

In this paper, we propose a unified optimization framework that can tackle two problems in the domain of enterprise information networks — host community detection and host anomaly assessment. Our perspectives in both tasks are based on host behaviors. This particular domain comes up with unique data characteristics, new community formulation, and great challenges of community detection and anomaly assessment. We propose an embedding-based model to investigate intricate behavioral patterns of each host purely based on their historical events. Empirical studies on real enterprise information networks show our proposed model can effectively identify host communities and assess host behavioral anomaly status, and outperform other popular community detection methods. An interesting direction for further exploration would be applying the proposed framework to other applications (such as social networks) and tasks (such as root cause analysis).

REFERENCES

[1] Ting Chen, Lu-An Tang, Yizhou Sun, Zhengzhang Chen, Haifeng Chen, and Guofei Jiang. 2016. Integrating community and role detection in information networks. In *Proceedings of the 2016 SIAM International Conference on Data Mining*. SIAM, 72–80.

[2] Ting Chen, Lu An Tang, Yizhou Sun, Zhengzhang Chen, and Kai Zhang. 2016. Entity embedding-based anomaly detection for heterogeneous categorical events. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*. 1396–1403.

[3] Zhengzhang Chen. 2012. *Discovery of informative and predictive patterns in dynamic networks of complex systems*. Ph.D. Dissertation. North Carolina State University.

[4] Zhengzhang Chen, William Hendrix, Hang Guan, Isaac K. Tetteh, Alok N. Choudhary, Fredrick H. M. Semazzi, and Nagiza F. Samatova. 2013. Discovery of extreme events-related communities in contrasting groups of physical system networks. *Data Mining and Knowledge Discovery* 27, 2 (2013), 225–258.

[5] Zhengzhang Chen, William Hendrix, and Nagiza F Samatova. 2012. Community-based anomaly detection in evolutionary networks. *Journal of Intelligent Information Systems* 39, 1 (2012), 59–85.

[6] Zhengzhang Chen, Kanchana Padmanabhan, Andrea M Rocha, Yekaterina Shpanskaya, James R Mihelcic, Kathleen Scott, and Nagiza F Samatova. 2012. SPICE: Discovery of phenotype-determining component interplays. *BMC Systems Biology* 6, 1 (2012), 40.

[7] Zhengzhang Chen, Kevin A. Wilson, Ye Jin, William Hendrix, and Nagiza F. Samatova. 2010. Detecting and tracking community dynamics in evolutionary networks. In *Proceedings of the 2010 IEEE International Conference on Data Mining Workshops (ICDMW'10)*. 318–327.

[8] Aaron Clauset, Mark EJ Newman, and Cristopher Moore. 2004. Finding community structure in very large networks. *Physical Review E* 70, 6 (2004), 066111.

[9] Boxiang Dong, Zhengzhang Chen, Hui (Wendy) Wang, Lu-An Tang, Kai Zhang, Ying Lin, Zhichun Li, and Haifeng Chen. 2017. Efficient discovery of abnormal event sequences in enterprise security systems. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management (CIKM'17)*. 707–715.

[10] Santo Fortunato. 2010. Community detection in graphs. *Physics Reports* 486, 3 (2010), 75–174.

[11] Jing Gao, Feng Liang, Wei Fan, Chi Wang, Yizhou Sun, and Jiawei Han. 2010. On community outliers and their efficient detection in information networks. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 813–822.

[12] Michael Gutmann and Aapo Hyvärinen. 2010. Noise-contrastive estimation: A new estimation principle for unnormalized statistical models.. In *AISTATS*, Vol. 1. 6.

[13] Greg Hamerly and Charles Elkan. 2004. Learning the k in k-means. In *Advances in neural information processing systems*. 281–288.

[14] Steven A Hofmeyr, Stephanie Forrest, and Anil Somayaji. 1998. Intrusion detection using sequences of system calls. *Journal of Computer Security* 6, 3 (1998), 151–180.

[15] Zhiting Hu, Poyao Huang, Yuntian Deng, Yingkai Gao, and Eric Xing. 2015. Entity hierarchy embedding. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, Vol. 1. 1292–1300.

[16] Tsuyoshi Idé and Hisashi Kashima. 2004. Eigenspace-based anomaly detection in computer systems. In *Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 440–449.

[17] David J. Ketchen and Christopher L. Shook. 1996. The application of cluster analysis in strategic management research: an analysis and critique. *Strategic Management Journal* 17, 6 (1996), 441–458.

[18] Yan Liu, Alexandru Niculescu-Mizil, and Wojciech Gryc. 2009. Topic-link LDA: joint models of topic and author community. In *Proceedings of the 26th International Conference on Machine Learning*. ACM, 665–672.

[19] Chen Luo, Zhengzhang Chen, Lu-An Tang, Anshumali Shrivastava, Zhichun Li, Haifeng Chen, and Jieping Ye. 2018. TINET: Learning invariant networks via knowledge transfer. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD'18)*. 1890–1899.

[20] Christopher D Manning, Prabhakar Raghavan, Hinrich Schütze, et al. 2008. *Introduction to information retrieval*. Vol. 1. Cambridge university press Cambridge.

[21] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*. 3111–3119.

[22] Darren Mutz, Fredrik Valeur, Giovanni Vigna, and Christopher Kruegel. 2006. Anomalous system call detection. *ACM Transactions on Information and System Security (TISSEC)* 9, 1 (2006), 61–93.

[23] Mark EJ Newman. 2004. Fast algorithm for detecting community structure in networks. *Physical review E* 69, 6 (2004), 066133.

[24] Mark EJ Newman. 2006. Finding community structure in networks using the eigenvectors of matrices. *Physical Review E* 74, 3 (2006), 036104.

[25] Caleb C Noble and Diane J Cook. 2003. Graph-based anomaly detection. In *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 631–636.

[26] Pascal Pons and Matthieu Latapy. 2005. Computing communities in large networks using random walks. In *International Symposium on Computer and Information Sciences*. Springer, 284–293.

[27] Peter J Rousseeuw. 1987. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics* 20 (1987), 53–65.

[28] Yiye Ruan, David Fuhry, and Srinivasan Parthasarathy. 2013. Efficient community detection in large networks using content and links. In *Proceedings of the 22nd International Conference on World Wide Web*. ACM, 1089–1098.

[29] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. Line: Large-scale information network embedding. In *Proceedings of the 24th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 1067–1077.

[30] Robert Tibshirani, Guenther Walther, and Trevor Hastie. 2001. Estimating the number of clusters in a data set via the gap statistic. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 63, 2 (2001), 411–423.

[31] Nguyen Xuan Vinh, Julien Epps, and James Bailey. 2010. Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance. *Journal of Machine Learning Research* 11, Oct (2010), 2837–2854.

[32] Joyce Jiyoung Whang, David F Gleich, and Inderjit S Dhillon. 2013. Overlapping community detection using seed set expansion. In *Proceedings of the 22nd ACM International Conference on Information & Knowledge Management*. ACM, 2099–2108.

[33] Zhiqiang Xu, Yiping Ke, Yi Wang, Hong Cheng, and James Cheng. 2012. A model-based approach to attributed graph clustering. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. ACM, 505–516.

[34] Jaewon Yang, Julian McAuley, and Jure Leskovec. 2013. Community detection in networks with node attributes. In *Proceedings of the IEEE International Conference on Data Mining*. IEEE, 1151–1156.

[35] Tianbao Yang, Rong Jin, Yun Chi, and Shenghuo Zhu. 2009. Combining link and content for community detection: a discriminative approach. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 927–936.

[36] Xi Zhang, Jian Cheng, Ting Yuan, Biao Niu, and Hanqing Lu. 2013. TopRec: domain-specific recommendation through community topic mining in social network. In *Proceedings of the 22nd International Conference on World Wide Web*. ACM, 1501–1510.

[37] Yang Zhou, Hong Cheng, and Jeffrey Xu Yu. 2009. Graph clustering based on structural/attribute similarities. *Proceedings of the 35th International Conference on Very Large Data Bases* 2, 1 (2009), 718–729.