Relational Network-Service Clustering Analysis with Set Evidences

Li Pu
Artificial Intelligence Laboratory
EPFL/IC/LIA, Station 14
Lausanne, Switzerland
li.pu@epfl.ch

Boi Faltings
Artificial Intelligence Laboratory
EPFL/IC/LIA, Station 14
Lausanne, Switzerland
boi.faltings@epfl.ch

Qiang Yang
CSE Department
Hong Kong University of Science and Technology
Kowloon, Hong Kong
qyang@cse.ust.hk

Derek Hao Hu
CSE Department
Hong Kong University of Science and Technology
Kowloon, Hong Kong
derekhh@cse.ust.hk

ABSTRACT

Network administrators are faced with a large amount of network data that they need to sift through to analyze user behaviors and detect anomalies. Through a network monitoring tool, we obtained TCP and UDP connection records together with additional information of the associated users and software in an enterprise network. Instead of using traditional payload inspection techniques, we propose a method that clusters such network traffic data by using relations between entities so that it can be analyzed for frequent behaviors and anomalies. Relational methods like Markov Logic Networks is able to avoid the feature extraction stage and directly handle multi-relation situations. We extend the common pairwise representation in relational models by adopting set evidence to build a better objective for the network service clustering problem. The automatic clustering process helps the administrator filter out normal traffic in shorter time and get an abstract overview of opening transport layer ports in the whole network, which is beneficial for assessing network security risks. Experimental results on synthetic and real datasets suggest that our method is able to discover underlying services and anomalies (malware or abused ports) with good interpretations.

Categories and Subject Descriptors

K.6.5 [**Management of Computing and Information Systems**]: Security and Protection—*Invasive software (e.g., viruses, worms, Trojan horses)*; I.5.3 [**Pattern Recognition**]: Clustering—*Algorithms*

General Terms

Experimentation, Security

Keywords

network service, relational learning, clustering

1. INTRODUCTION

In an enterprise network, various applications that connect from client machines to servers depict certain behavior of the whole network. Administrators need to ensure the security of the network based on in-depth understanding of such behaviors. It is, however, very challenging to find out the most important part that needs special attention from the huge amount of network data.

A common method for investigating software behavior in a network environment uses filters for automatically selecting a small subset of the data, and then manually process the selected data by experts. The anomaly detecting filters can be implemented with various techniques [10, 6, 4, 26], but many of them require expert knowledge of the entire network status.

Another approach addressing the same problem reduces the amount of human work by applying clustering techniques so that only the suspicious clusters need to be inspected. We call a cluster of connections of similar functionalities a *service*. A typical method for identifying network services exhaustively examines the transport layer ports. But in real data there are too many connections and ports where most of them are not used for their registered purpose (although the Internet Assigned Numbers Authority tries to maintain a list of port numbers and functionalities). Some software also uses dynamic ports to bypass port number filters.

From the machine learning point of view, the second class of methods first maps the raw data into a metric space and then uses various clustering algorithms for classifying network traffic. It relies on extracting feature vectors from each connection or sequence of connections, e.g., type and length of the connection, entropy of payload, existence of keywords and other statistics, so the quality of selected features plays an important role. After getting these features, one can build a supervised or unsupervised classifier on the dataset and assign each connection a class label [20, 27, 7, 5, 15].

Through collaboration with the company Nexthink (www. nexthink. com), we obtained the TCP or UDP connection records of all client machines in an enterprise network. For each connection record, besides the commonly used TCP/UDP 5-tuple (protocol, source IP, source port, destination IP, destination port), it is also associated with the user and application name that initialized the connection. This additional information allows us to combine the connections from all client machines to make deeper inference about network status. In this paper, we try to utilize only the relations between connections, ports, applications and servers (destinations) for discovering underlying services, i.e., assign each open port in the network to a cluster that describes its functionality.

In our work, the goal is not to assign an optimal service label to every TCP/UDP connection, but to create an overview of port clusters for the whole network, which provides a better interface for in-depth inspection the administrator. For example, in Windows NT/2000/XP, file sharing is implemented on top of NetBT (NetBIOS over TCP/IP, port TCP/139) and directly on port TCP/445, so usually both ports can be found on those servers that provide file sharing service. If the administrator sees such ports cluster in a server, he knows what is the role of this server and corresponding security risks. Table 1 shows the servers open ports TCP/139 or TCP/445 in a real enterprise network in which most of the servers open both ports. Our clustering task is to assign the ports into several disjoint clusters by the evidences that connections provide. Once the ports are presented within clusters, the administrator would have a clearer picture about the services running in the network and their security risks.

TCP/139	10.201.0.2, 10.130.10.49, 10.130.10.69, 10.130.10.98, 10.130.10.107, 10.130.10.111, 10.130.10.113, 10.130.10.125, 10.130.10.159, 10.130.10.161, 10.130.10.222, 10.130.10.226
TCP/445	10.201.0.2, 10.130.10.69, 10.130.10.107, 10.130.10.111, 10.130.10.113, 10.130.10.125, 10.130.10.159, 10.130.10.222, 10.130.10.226

Table 1: Servers open ports TCP/139 and TCP/445 in an enterprise network.

Our clustering approach is based on a machine learning technique called statistical relational learning (SRL). SRL, more specifically Markov Logic Networks, encodes the probability distribution of possible states into a set of logic formulas with weights, and takes full advantage of first-order logic to achieve richer expressiveness [25, 9]. The problem is written with first-order logic formulas that describe the relations and truth values of observable ground predicates. Markov Logic associates each formula with a certain weight and computes probabilities by summing the total weight of those satisfied formulas. But in the clustering problem, instead of obtaining full probability distributions over all possible evaluations of unobservable ground predicates, we are more interested in finding out the most probable evaluation for all ground predicates, which exactly provides us the information of the underlying network services. Compared to the vector space model methods (for example k-means, support vector machine), the relational model allows us to directly model relations between entities rather than embed them into the features. And the user can directly specify what are the clustering variables in the relational model, which is sometimes not clear in the vector space model.

The rest of the paper is organized as follows. In Section 2 we enumerate the related research topics and results. Then the formal definitions of set evidence, modularity and Markov Logic can be found in Section 3. Network-service clustering problem is described with Markov Logic and improved by considering set evidences in Section 4. In Section 5, we present details of synthetic and real enterprise network datasets, experimental setup and discussions of the results. Finally Section 6 concludes our work with future research topics.

2. RELATED WORK

SRL has been widely used in many fields such as natural language processing, social network analysis, and semantic webs [18, 14, 16, 23]. Clustering applications like entity resolution in relational data bring additional information to the traditional feature vector space model methods [1, 23]. The unsupervised learning application for natural language processing with Markov Logic Networks can be found in [23]. In this paper we extend it by adopting set evidences.

To incorporate the linkage information in the clustering problem, people studied another class of methods, namely *community detection*. The goal of the community detection problem is to get the partition of connected objects into clusters such that the intra-cluster connections are denser than inter-cluster connections[22, 21, 3, 8]. In the community detection literature, the problem is often modeled as a graph where the nodes denote objects and the edges denote relations between objects. Our proposed greedy algorithm for the clustering problem in Markov Logic with set evidence is similar to the *agglomerative algorithm* in community detection [22]. But community detection algorithms usually allow only one type of relation represented by the edges, while in our work the relations are more diverse.

Moore [19] proposes an ensemble method for classifying network applications. The authors report that more than 99% of the packages can be correctly identified. But this work requires the software to look into the payload, which is time consuming and becomes invalid in the case of an encrypted payload. Karagiannis [13] develops an approach called BLINC to classifying network flows according to the traffic patterns at three different levels. This work is similar to our work in the sense that it tries to associate hosts with applications rather than solely classify applications. But in our work the information about applications is in a fine granularity and more complete.

Kandula [12] presents a system that takes information from various sources in the enterprise network to perform detailed diagnosis of performance problems. It provides an evidence that combining network traffic and application information could bring a new perspective to the problem. Homer [11] proposes a SAT-solving approach to obtain the optimal configuration changes given current information security status, attacks, usability requirements and costs of actions. This system allows the administrator to balance the security and the usability of the network, but it assumes that all the attacks, misconfigurations and vulnerabilities are known.

3. PRELIMINARIES

3.1 Set Evidence

With the graph representation, relations between objects are often described in a pairwise manner. This evidence description method is not natural in cases where the evidence refers to sets of objects, such as the fact that some objects are probably in the same cluster. Taking the example of ports connected from the same application to the same server, we can infer they are probably in the same cluster. But it is not justified to assume a relation between every pair of ports, because there might exist some sub-clusters structure, i.e., the ports actually belong to more than one cluster.

Consider n ground-truth clusters $C_i, (i = 1, 2, ..., n, C_p \cap C_q = \emptyset, R = \cup C_i)$, m set evidences $G_j \subseteq R, (j = 1, 2, ..., m)$, and a partition $T = \{T_i\}$ over R, the goal of our algorithm is to recover the clusters from G_j such that $T_i = C_i$. In the network service clustering problem, R is the set of all ports, each set evidence G_j consists of some ports which is indicated by the relation with application-destination pairs (we will explain the details in the next section). We say a partition T is *consistent* with a set evidence G_j if all members of the set evidence are in the same cluster T_i, otherwise G_j is *inconsistent*. For building port clusters, if all ports in a set evidence G_j are assigned to the same cluster, it is consistent. In practice it is hard to generate a partition which is consistent with all set evidences if m is large. Therefore, we would like to find out the optimal partition in the sense that a minimal set evidence inconsistency is introduced.

In practice, set evidence can be found in many applications. If a group of people with similar interests always goes to the same club, we can infer that they are in the same community (cluster). If there are multiple clubs or institutions, we can introduce more set evidences. But it is improper to assume a connection between every pair of persons that appeared in the club, because there might exist a sub-community structure, i.e., one person does not necessarily know all other persons in the club. This is why we use the set evidence relation instead of pairwise relations for clustering. Another example naturally appears in recommender systems, where a certain group of people buy the same item have higher probability to fall into the same community.

3.2 Modularity

Given a partition over the set of objects on which the clustering algorithm is performed, we would like to have an evaluation measure to assess the goodness of the result. If the ground-truth is known, this can be done by comparing differences between the algorithm output and the ground-truth. But in many cases the ground-truth is not available because it is exactly what we would like to discover. Many measures are developed to evaluate the goodness of results without knowing the ground-truth. For example, the *modularity* on a graph $\mathscr{G}$ and a partition T counts the number of actual edges in the cluster minus the expected edges in the cluster as if the edges are randomly placed between nodes with an expected probability [22, 21]. If the set of all nodes is divided into k communities (clusters), a $k \times k$ matrix E can be constructed where each entry e_{ij} denotes the fraction of all edges in the graph that link nodes in cluster i to nodes in cluster j. Modularity is defined as [21]

$$Q(\mathscr{G}, T) = \sum_i \left(e_{ii} - (\sum_j e_{ij})^2 \right) = Tr(E) - |E^2|,$$

where $|E^2|$ indicates the sum of the elements of the matrix E^2. The modularity suggests stronger community structure if it is close to 1, and usually ranges from 0.3 to 0.7 in practice.

In this work, the evidence, however, is not expressed in a pairwise way, but in the form of set evidence, so the definition of modularity needs to be modified. The first modification adapts set evidence into a graph. The set of nodes of the graph is the set of all elements in R. An edge between node a and node b is added to the graph if both nodes are contained in the same set evidence G_j. The modularity of this induced graph is denoted as Q_1. This procedure would produce a relatively dense graph because all nodes in the set evidences are fully connected even if they are actually unrelated.

We also propose a second modification where the modularity is directly computed from set evidences,

$$Q_2(G, T) = \sum_i \left(e_i - a_i^2 \right), \; a_i = \frac{|\{G_k | G_k \cap T_i \neq \emptyset\}|}{m}$$

$$e_i = \frac{|\{G_k | G_k \cap T_i \neq \emptyset \text{ and } \forall j \neq i, G_k \cap T_j = \emptyset\}|}{m}$$

e_i is the fraction of set evidences that only intersects with T_i and a_i includes those set evidences also intersect with other clusters. This modularity is denoted as Q_2.

In this work, modularity is used as an evaluation measure rather than an optimization objective. Since the ground-truth of cluster labels for each object is unknown for real data, higher modularity does not necessarily imply a better clustering result because it only indicates that the result better fits our assumption about community structure – the intra-cluster connections are denser than inter-cluster connections.

3.3 Markov Logic Networks

Markov Logic Networks (MLN) is a language that adopts the expressiveness of undirected probabilistic graphical model and first-order logic proposed by Richardson et al. [25] to deal with both uncertainty and complexity in the applications. A MLN model is a set of first-order logic formulas where each formula is associated with a real-valued weight. The weighted formulas can be viewed as templates for building Markov networks. The joint distribution of the grounded Markov network can be computed by

$$P(X = x) = \frac{1}{Z} \exp\left(\sum_i w_i f_i(x)\right) = \frac{1}{Z} \exp\left(\sum_i w_i n_i(x)\right)$$

where x is the truth value vector of atom vector X, w_i is the weight of formula i, $f_i(x)$ denotes the satisfiability vector of formula i, and $n_i(x)$ is the number of true groundings of formula i with evaluation x. Efficient inference of MLN is implemented by Markov chain Monte Carlo methods, e.g. MC-SAT and the "lazy" version Lazy-MC-SAT [24]. In Lazy-MC-SAT, the predicates are grounded when needed, so it is more efficient in memory. Please refer to [25] for more details.

Vector space model methods like k-means, k-nearest neighbor and unsupervised support vector machine can be also applied for finding the partitions to the relational data, but they all require a feature extraction stage where the feature values are computed from the relations. In a multi-relation dataset, one has to normalize all relations into different scale so that all the dimensions can be considered with different weights, while relational models like MLN directly handle multi-relation situations and perform the clustering algorithm.

4. NETWORK-SERVICE CLUSTERING

We assume the following network traffic and related information are recorded in the dataset: connection ID, source IP address, destination IP address, port of TCP/UDP connection, and the name of process running in a client machine that initiates the connection. The only observations we have are connection records from an application to a destination with a port. Four types of entities from the observations are considered: *connection* that represents a TCP/UDP session; *application* that denotes the process of the same executable filename; *destination* is a server of an unique IP address that open some ports to the applications; and *port* is a combination of port type (TCP or UDP) and port number. The source IP address and source port information are omitted because we treat the same application on different client machines equally.

4.1 MLN representation for Network-Service Clustering

Now we describe the service-clustering problem with the MLN language. First, one more type of variables called *service* needs to be introduced. A service is a cluster that consists of one or more ports. Then the predicates are defined for modeling the problem and describing relations between different objects. `connApp(conn, app)` means that the connection is initialized by the application. `connDest(conn, dest)` indicates that the connection points to the destination. `connPort(conn, port)` means that the port is used by the connection. `haveService(dest, service)` suggests that the service is hosted in a destination. `appUseService(app, service)` is true if and only if the application uses the service. And finally `servicePort(service, port)` denotes that the port is included in the service.

Although the ports are not always used for the registered purpose, we find in the real dataset that the functionality of one port is quite stable within the enterprise network. It is reasonable to assume that one port only belongs to one service. For this assumption, we impose the constraint on predicate `servicePort(service!, port)`. The symbol `!` means mutually exclusive and exhaustive in the MLN syntax. It is worth mentioning that this constraint could largely reduce the number of possible worlds (value assignments to ground predicates) in the problem, since all instantiations of service are unknown and treated equally. The search space can be reduced if we break the symmetry worlds where the same value assignment is applied to `servicePort` with permuted service names. To do this, Poon et al. [23] give a unique name to each new cluster when it is created. In this paper, we identify equivalent classes by sorting the services with the smallest member id contained in each service.

The first formula in Figure 1 states that if more than one connection is associated with the same application-destination pair, the services of the ports of the connections are likely to be the same. This rule helps to merge similar ports into one cluster and actually defines a group of sets where each set evidence can be named by the application-destination pair. In first-order logic, set evidence is often represented in a pairwise way like in Figure 1, but this representation is not efficient and sometimes not appropriate in the clustering settings. We will discuss this issue and refine MLN to directly incorporate set evidence later.

The second and third formulas (with weights w_2 and w_3) suggest that if the port of a connection belongs to a service, the application and destination associated with this connection are likely to use or host this service. If two ports that originally belong to the same service are assigned to two services, one more instantiation can be satisfied for each formula and a bigger weight is obtained. So these two rules encourage more clusters and the total weight of these two rules is proportional to the number of services. The last two formulas set the prior on `appUseService` and `haveService` to prevent them from being always true.

As long as we have these formulas and weights, the goal of clustering is to maximize a posterior probability $P(A|Y) \propto P(A,Y) = P(A)P(Y|A)$ where A is the cluster assignment vector and Y is the vector of observed values. In our problem, $A = \texttt{servicePort(s,p)}$ and $Y = \{\texttt{connApp(c,a)}, \texttt{connDest(c,d)}, \texttt{connPort(c,p)}\}$. The best cluster assignment vector is the one that maximizes the total weight of satisfied ground formulas or minimize the total weight of unsatisfied formulas. The prior distribution $P(A)$ and conditional distribution $P(Y|A)$ is encoded into the MLN formulas. Exact inference in MLN is NP-hard, so Monte Carlo method or heuristics are usually adopted to find the solution. Intuitively the first formula tends to gather ports into less clusters while other formulas prefer more clusters. By tuning the weights, we can control the final number of clusters. The weights are learned from a subset of the dataset by the alchemy software [17].

4.2 Refinement with Set Evidence

Consider three set evidences that intersect with one another as shown in Figure 2 (a). If the final cluster assignments divide a set evidence into more than one part, some ground formulas will not be satisfied so that the total weight will decrease. It is shown in Figures 2 (b) and (c) that with pairwise representation the weight loss depends on the size of set evidence and the partition (the filled polygons). If we add edges to all pairs of objects in a set evidence, the partition in (b) breaks 9 edges and another partition in (c) breaks 5 edges. The two partitions introduce different weight losses, but in principle the weight losses under these two partitions should be the same because the elements in set evidence are equally treated. In other words, there should be no preference between evenly dividing a set evidence or unevenly dividing it. Another problem with pairwise representation is that violating a large set evidence is much harder than separating elements from a small set evidence, because the former brings larger weight loss. To deal with these problems, we propose a set evidence function as an extension to the standard MLN. This function takes the form as following,

$$w_f = w \cdot \texttt{SetE}(\texttt{e}, \texttt{z}, f).$$

It maps a first-order formula f to a real valued total weight

w_1; $(\texttt{connApp(c1,a)} \wedge \texttt{connDest(c1,d)} \wedge \texttt{connPort(c1,p1)}) \wedge (\texttt{connApp(c2,a)} \wedge \texttt{connDest(c2,d)} \wedge \texttt{connPort(c2,p2)}) \Rightarrow$
$(\texttt{servicePort(s,p1)} \wedge \texttt{servicePort(s,p2)})$
w_2; $(\exists \texttt{c,p}(\texttt{connApp(c,a)} \wedge \texttt{connPort(c,p)} \wedge \texttt{servicePort(s,p)})) \wedge \texttt{appUseService(a,s)}$
w_3; $(\exists \texttt{c,p}(\texttt{connDest(c,d)} \wedge \texttt{connPort(c,p)} \wedge \texttt{servicePort(s,p)})) \wedge \texttt{haveService(d,s)}$
w_4; $\texttt{appUseService(a,s)}$
w_5; $\texttt{haveService(d,s)}$

Figure 1: MLN formulas for network service clustering.

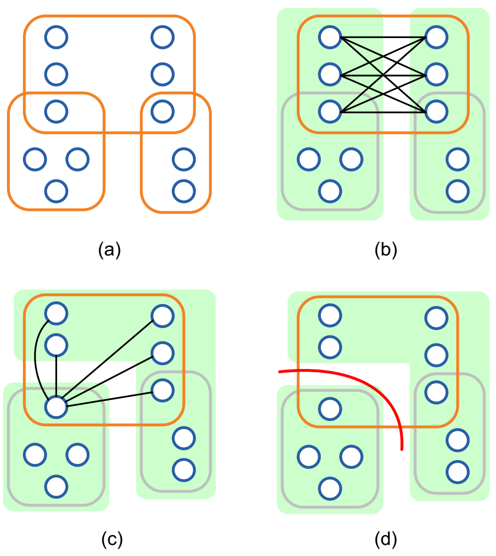

Figure 2: The unsatisfied formulas introduced by inconsistent set evidences.

w_f, and includes another two arguments. The the first argument $\texttt{e}$ is the set of variables representing set evidence and the last argument $\texttt{z}$ denotes the variable needs to be clustered (partition variable) in f. And w is the base weight of formula f. The set of variables in f other than $\texttt{e}$ and $\texttt{z}$ is denoted as $\texttt{v}$. For specific constants $t_\texttt{e}$ instancing the variables in $\texttt{e}$, we can define how $\texttt{SetE}$ computes the total weight. Following the idea shown in Figure 2, it can be defined as,

$$\texttt{SetE}_1(\texttt{e},\texttt{z},f) = \sum_{t_\texttt{e}} \left(|\{t_\texttt{z}|\exists t_\texttt{v}, f(t_\texttt{e},t_\texttt{z},t_\texttt{v}) = \text{true}\}| - 1\right).$$

For each $t_\texttt{e}$, $\texttt{SetE}_1$ computes the number of clusters in the final result that intersect with $t_\texttt{e}$ minus 1. The total weight can be seen as a penalty of breaking set evidences, so the weight w should be a negative value. By this refinement, the position of the partition boundary will have no influence on the weight contribution of formula f as long as the partition has the same number of subsets (as shown in Figure 2 (d)). Another $\texttt{SetE}$ function considers the strength of set evidence, in which bigger set evidence introduces larger penalty.

$$\texttt{SetE}_2(\texttt{e},\texttt{z},f) =$$
$$\sum_{t_\texttt{e}} \sum_{\texttt{z1}\neq\texttt{z2}} \left(|\{t_\texttt{v}|f(t_\texttt{e},t_{\texttt{z1}},t_\texttt{v}) = \text{true}\}| + |\{t_\texttt{v}|f(t_\texttt{e},t_{\texttt{z2}},t_\texttt{v}) = \text{true}\}|\right).$$

But in practice, $\texttt{SetE}_2$ would generate big clusters in some cases, because it repeatedly adds more objects to the cluster if it is much larger than others. To deal with this problem, the third function is defined as,

$$\texttt{SetE}_3(\texttt{e},\texttt{z},f) =$$
$$\sum_{t_\texttt{e}} \sum_{\texttt{z1}\neq\texttt{z2}} \min\left(|\{t_\texttt{v}|f(t_\texttt{e},t_{\texttt{z1}},t_\texttt{v}) = \text{true}\}|, |\{t_\texttt{v}|f(t_\texttt{e},t_{\texttt{z2}},t_\texttt{v}) = \text{true}\}|\right),$$

which takes the minimal value of strength from different clusters. Now the first formula in Figure 1 can be rewritten as $w; \texttt{SetE}_k([\texttt{a},\texttt{d}],\texttt{s},f)$ where $k = \{1,2,3\}$ and f is defined as

$$f = (\texttt{connApp(c,a)} \wedge \texttt{connDest(c,d)} \wedge \texttt{connPort(c,p)})$$
$$\Rightarrow \texttt{servicePort(s,p)}.$$

The set evidence variables are $\texttt{a}$ and $\texttt{d}$, and the partition variable is $\texttt{s}$.

4.3 Clustering Algorithm

Similar to MLN, the refined problem is NP-hard. A random walk algorithm (e.g. MaxWalkSAT or LazySAT) is often adopted in previous works [17, 23]. We propose another approach for MLN which is similar to the greedy algorithm used in community detection [2]. In the context of network service clustering, the algorithm initializes the partition in which each port forms one cluster by assigning each port a different service. Then it iteratively chooses two services and merge their corresponding ports into one service such that

Algorithm 1 **GreedyClustering**(*maxSteps*, *threshold*, *formulas*, $\texttt{e}$, $\texttt{z}$)

```
InCluster($z_i$, $obj_i$) ← true, $i = 1, 2, ...|\{obj\}|$.
for $k$ ← 1 to $maxSteps$ do
  $mergePair$ ← $null$, $maxGain$ ← $-inf$
  for $(i,j) \leftarrow i,j \in \{1,2,...,|\{z\}|\}, i > j$ do
    if ($gain = WGain(formulas, e, z, i, j, InCluster)$) > $maxGain$ then
      $maxGain$ ← $gain$, $mergePair$ ← $(i,j)$
    end if
  end for
  if $maxGain < threshold$ then
    break
  end if
  $(p,q)$ ← $mergePair$
  InCluster($z_p$, obj) ← true, if InCluster($z_q$, obj) == true.
end for
return InCluster(z, obj)
```

the total weight gain is maximized. Because in each step, we only change the values of the predicates in two different services, the total weight gain can be computed efficiently. We repeat the above step until no more weight improvement can be made or step number limit is reached. A general form of this procedure is shown in Algorithm 1. The predicate `InCluster(z,obj)` denotes the cluster assignment where `obj` is the variable on which the clustering is performed, e.g. ports. To move to the next iteration, a minimal weight gain (usually a positive real number) is required, which is denoted by the parameter *threshold*. The function $WGain$ takes current cluster assignment and two candidate clusters, then computes weight gain by merging them.

5. EXPERIMENTAL RESULTS

In this section, we apply our network service clustering model and algorithm to a network traffic dataset. Since the real enterprise dataset does not have labels for evaluation, the algorithms are first tested on synthetic datasets and then applied to the real data collected by Nexthink. As explained in Section 4, each dataset consists of a number of records and each record is a tuple {*connection*, *application*, *destination*, *port*}. For synthetic dataset, there is one more column *service* in each record indicating the label of the port.

We use a supervised *pairwise F-measure* (PWF) [28] to evaluate the results. Let $\mathcal{T}$ denote the pairs of objects that have the same cluster label, and $\mathcal{S}$ denote the pairs of objects assigned to the same cluster in the result. PWF is computed from the pairwise precision and recall.

$$precision = |\mathcal{T} \cap \mathcal{S}|/|\mathcal{S}|,\ recall = |\mathcal{T} \cap \mathcal{S}|/|\mathcal{T}|$$

$$PWF = \frac{2 \times precision \times recall}{precision + recall}$$

Higher PWF indicates better clustering result. The unsupervised community structure measures Q_1 and Q_2 are also computed for both synthetic and real datasets.

The k-means algorithm is taken as the baseline. Feature vectors are constructed from the relations between application-destination pairs and ports. For each port there is a feature vector of length N_v that equals to the number of unique application-destination pairs. The entry corresponding to application-destination pair `(a,d)` for port `p` is set to 1 if $\exists c, (\texttt{connApp(c,a)} \wedge \texttt{connDest(c,d)} \wedge \texttt{connPort(c,p)}) =$ true, otherwise 0. The number of desired clusters k for k-means algorithm is set to be the same as the true number of clusters, preferring that the k-means algorithm will achieve the best possible performance. In every experiment, the k-means algorithm is repeated 30 times with randomly chosen initial clusters and then the average performance is computed.

5.1 Synthetic Dataset

The process of generating a synthetic dataset simulates the software and service behaviors in a real network. Each port is first associated with a service. The ratio of maximum service size over minimum service size is denoted as α, which indicates the degree of unbalance among services (clusters). Samples of application-destination pairs using the same service are uniformly drawn from all possible pairs, and a service name is assigned to each pair. Then some connections are generated by adding ports in the corresponding

	Synthetic dataset	Nexthink dataset
# connections	8000	13774
# applications	80	320
# destinations	40	840
# ports	180	474
# services	40	n/a

Table 2: The size of synthetic and Nexthink dataset.

services to application-destination pairs. To compare the performance of relational clustering algorithms using set evidence to the baseline, unbalanced clusters are introduced. α is set to 10 and 20 in another two datasets.

To test the performance of the algorithm under noise, the ports of a fraction β of connections are chosen from a random service, which introduces incorrect set evidences into the dataset. We set β to 0.05 in one dataset. The size of synthetic dataset is shown in Table 2.

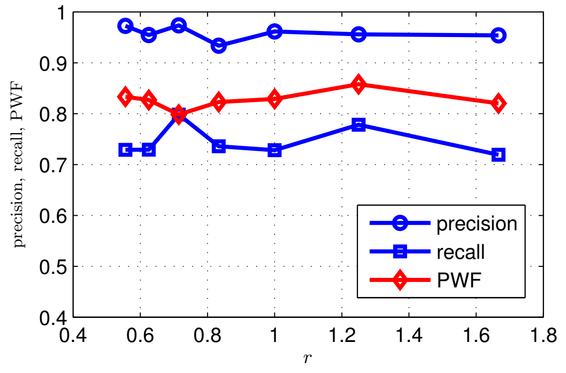

Figure 4: The PWF curve for different weight ratio $r = |w_1|/|w_2|$ on the same dataset.

Figure 3 shows the performance of different algorithms on synthetic datasets. In the the three datasets where β equals 0, the PWF score of $\texttt{SetE}_1$ is always better than k-means and other two relational methods. But methods $\texttt{SetE}_2$ and $\texttt{SetE}_3$ do not have significant improvement compared to k-means in synthetic datasets. In the case of noisy data, the performance of all relational methods decreases a lot. It is unusual to observe that the PWF score of k-means is better with noisy data. This is the consequence of our method of adding noisy data which makes a bridge between feature vectors in the same cluster and brings better results for k-means. In the k-means algorithm, the choice of the metric defined between ports makes a big difference on the final result. We use the cosine distance in this work since the data points have large dimensionality, which shows better performance than the Euclidean distance in the experiments with synthetic data. From Figure 3 we can also observe the correlation between PWF and modularity. Q_1 and Q_2 decrease when the PWF score is lower, so the modularity can be used to evaluate the goodness of the partition when the ground-truth is unknown.

Figure 4 illustrates the relationship between performance and weight ratio on the same noisy dataset. The ratio $r = |w_1|/|w_2|$ ($w_2 = w_3$) indicates the relative strength of

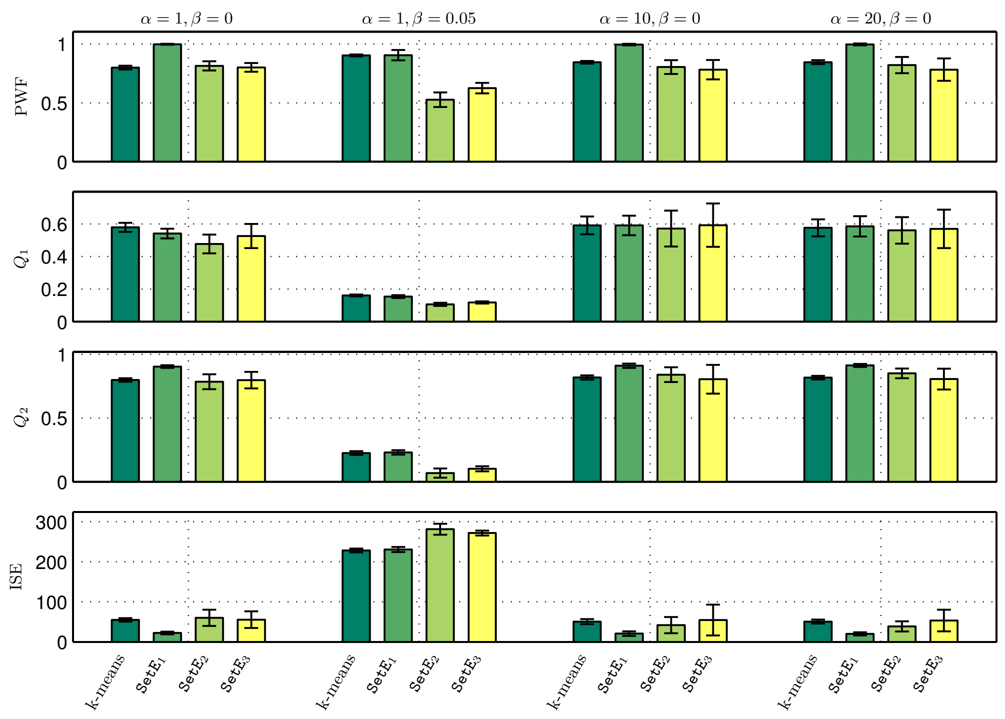

Figure 3: Performance of different algorithms on synthetic datasets (results are shown as mean and standard deviation). "ISE" is short for inconsistent set evidences.

gathering less clusters over the strength of generating more clusters. The result shows that our model is robust w.r.t. the weight change and maintains a stable performance within a weight ratio range.

5.2 Nexthink Dataset

Some features of the dataset collected by Nexthink from July 2008 to September 2008 in a real enterprise network are shown in Table 2. All connections are recorded in Windows systems. The connections with destinations outside the enterprise network are omitted, because we only care about the services inside the local network. For the real data, we find many ports are not used as the registered function in IANA, so it is hard to label the functions for all ports. In this case, only the unsupervised measures Q_1 and Q_2 are used for evaluation.

The relational algorithm usually outputs about 70 to 85 services (clusters) where the exact number depends on the terminal point of the algorithm. From the results, two main types of services can be found: ports range and ports function. Ports range is a service that contains consecutive ports uniquely used by a specific application. Service (cluster) 1 in Table 3 is an example of ports range. Service 2 to 5 are examples of ports function. Service 2 contains the two file sharing ports that we mentioned in the introduction. It provides the list of all file sharing servers without any prior knowledge about functionalities of ports in Windows system. The applications in service 3 actually belong to the same software package "Novadigm Radia software delivery and management tools". With this information, we can also find out which servers are hosting the Radia service, which helps the administrator to manage the software. Applications in service 5 are products of "Symantec AntiVirus Suite".

We also identified some groups of malware in the dataset, which is shown in services 6, 7 and 8 in Table 4. The malware in service 6 and 7 randomly changes its executable filename and sends UDP broadcast packages to communicate with other infected machines in the local network. With the service information, it is easy to query from the dataset about the infected source machines. Service 8 shows another group of malware using port TCP/50000 hosted in the same server (IP address 10.21.49.7). When using the k-means algorithm (services k1, k2 and k3), TCP/2638 and UDP/2638 are mixed with other ports in the result, so the suspicious applications can not be easily identified as malware. This result shows that our relational clustering method is able to discover the unknown network traffic groups and automatically present them to the administrator with clear interpretations. In practice, however, it is hard to tell which relational clustering algorithm would produce the best result without labels, the best strategy is to try different algorithms and look into the results for some reasonable interpretations (e.g. applications from the same company, or ports with the same known service).

Another observation from the results is the fact that only a few servers provide more than one type of service. 81% of the servers host only one service, and 94% of the servers only belong to at most 2 services. This indicates that the servers in the enterprise network are usually dedicated to one or two functionalities. The similar observation can be found for applications, 80% of the applications use only one

	Ports, applications, destinations
1	UDP50758, UDP50760, UDP50767, UDP50769, UDP50770, UDP50771, ... snmp.exe 10.0.0.20
2	TCP139, TCP445 system 10.130.10.111, 10.130.10.107, 10.130.10.226, 10.130.10.98, 10.130.10.222, ...
3	TCP3464, TCP3466 nvdkit.exe, radconct.exe, radstgms.exe, radstgrq.exe 10.130.10.94, 10.144.0.5, 10.136.0.5, 10.60.15.5, 10.140.1.5, 10.20.3.8, ...
4	TCP5008, TCP5009, TCP5011 vau.exe, workstation.exe 10.21.49.176
5	TCP2967, UDP1281, UDP2967, UDP38293 rtvscan.exe,savroam.exe 10.130.10.98, 10.144.0.5, 10.136.0.5, 10.2.0.5, 10.60.15.5, 10.20.3.8, ...

Table 3: Examples of services generated from Nexthink dataset.

service. Figure 5 shows the histogram of number of ports, applications and destinations in all services.

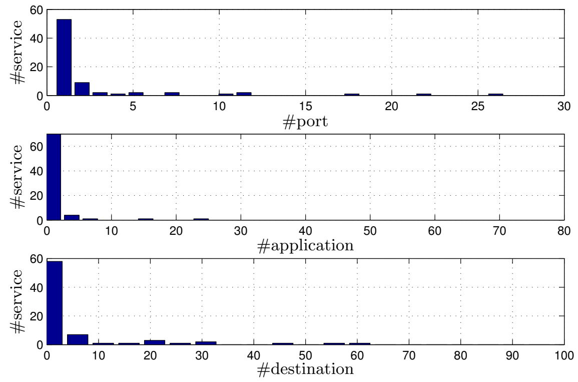

Figure 5: Histogram of number of ports, applications and destinations in services.

The Nexthink dataset has little noise, so the final results of different `SetE` functions are the same if the algorithm terminates by the *threshold* equals to 0. But the intermediate steps are different. Figure 6 shows the differences between relational clustering algorithms. $\texttt{SetE}_2$ and $\texttt{SetE}_3$ have similar intermediate results, while $\texttt{SetE}_1$ has larger difference with other two methods. The same comparison between different relational clustering algorithms and k-means is shown in Figure 7. The k-means result differs a lot from the results of relational algorithms, since we can observe that the curves keep decreasing when the clusters are forming. But $\texttt{SetE}_1$ presents less different clusters with the k-means method, because the feature vectors used in k-means essentially ignore the strength of relations as in $\texttt{SetE}_1$ (i.e. the number of connections associated with each relation is neglected).

	Ports, applications, destinations
6	TCP2638 novoterm.exe, tpmeritve.exe, vlaknagl.exe, vlaknagl1.exe, frichi2.exe, ... 10.21.49.7
7	UDP2638 novoterm.exe,tpmeritve.exe,vlaknagl.exe 10.255.255.255
8	TCP50000 commonupdt.exe, corporateebank.exe, initeformsmandb.exe, commonupdate.exe, ... 10.21.49.7
k1	*TCP2638*, TCP8290, TCP16384 hpqscnvw.exe, *novoterm.exe*, *tpmeritve.exe*, *vlaknagl.exe*, agentservice.exe, ... 10.136.10.2, *10.21.49.7*, 10.0.21.105, 10.130.11.86, 10.100.0.15
k2	UDP138, TCP2869, *UDP2638* system, svchost.exe, explorer.exe, wmpnetwk.exe, *novoterm.exe*, *tpmeritve.exe*, ... 10.136.0.36, 10.200.21.74, 10.200.255.255, 10.140.20.105, *10.255.255.255*, 10.136.0.30, ...
k3	TCP40000, *TCP50000*, TCP1233 mmc.exe, java.exe, *corporateebankmain.exe*, *commonupdt.exe*, *corporateebank.exe*, ... 10.130.10.111, 10.150.31.8, *10.21.49.7*

Table 4: Examples of malware detected from Nexthink dataset. Services 6, 7 and 8 are generated by relational clustering methods, while services k1, k2 and k3 are generated by k-means algorithm. The italic items are the intersecting parts between services from relational clustering methods and services from k-means.

Figure 8 shows the performance over all steps of relational algorithms. $\texttt{SetE}_1$ and $\texttt{SetE}_3$ have almost the same modularity Q_2 curve, where the maximal Q_2 is reached at about 50 steps before the algorithm terminates, while the maximal modularity Q_2 of $\texttt{SetE}_2$ is reached at the very end of the algorithm. In fact this maximal Q_2 point suggests the best clustering solution in the whole process. For example, after the maximal Q_2 step of $\texttt{SetE}_1$, service 2 in Table 3 is absorbed into a bigger cluster, which is obviously undesirable in Nexthink dataset. We can also observe that the maximal Q_2 of $\texttt{SetE}_2$ is larger than that of $\texttt{SetE}_1$ and $\texttt{SetE}_3$. This indicates that in the Nexthink dataset, relational clustering algorithm gets better results if the strength of the set evidence is considered. The max gain curve of $\texttt{SetE}_2$ oscillates a lot with several peaks. They actually suggest the points where new clusters are found. The performance of

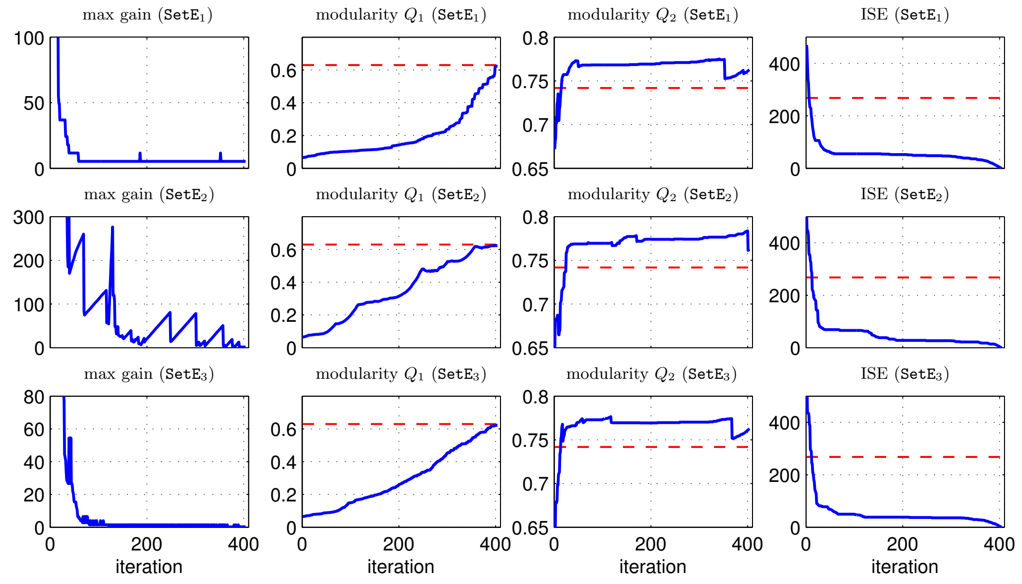

Figure 8: Performance of the relational algorithms on Nexthink dataset. Dashed lines are performance of k-means algorithm. "ISE" is short for inconsistent set evidences.

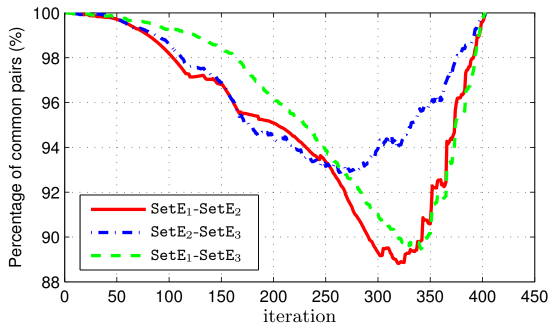

Figure 6: Comparison of intermediate steps for different relational clustering algorithms on Nexthink dataset. The percentage of common pairs is the percentage of port pairs that have the same in-cluster relation ("in the same cluster" or "in different clusters") among all possible port pairs.

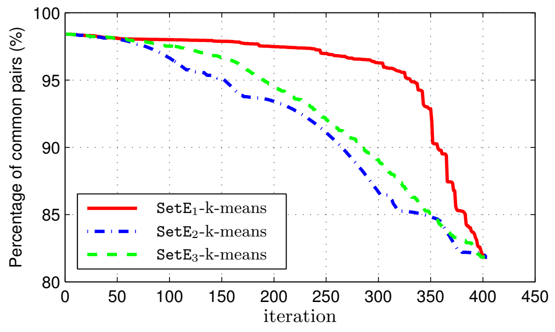

Figure 7: Comparison of intermediate steps between different relational clustering algorithms and k-means on Nexthink dataset.

k-means on the Nexthink dataset (dashed lines in Figure 8) is not satisfactory because it leaves too many set evidences inconsistent.

6. CONCLUSION AND FUTURE WORK

In this paper, a relational clustering model is proposed for discovering network services. It takes the advantage of rich expressiveness of first-order logic and encodes a prior and conditional probabilities into the formulas and weights of MLN. If more relations are available, this model can be easily extended by adding more formulas. To efficiently represent the set evidences, we adopt a special function `SetE` into the problem and extend the MLN for computing total weights. Experiments on synthetic dataset show that our method is relatively robust to weight disturbance and achieved better performance than the baseline method. The clusters generated from Nexthink data provide us an abstract overview of the services in the whole enterprise network, which is very helpful for network management and information security assessment.

In this work, only connectivity information is considered for clustering. In the result from real data, we do find 1 to 2 big clusters that contain many ports and applications. They are the results of some "universal" applications that employ many services (like explorer.exe). Other relations like the interactions between applications and users can be used in the future work to eliminate the influence of the "universal" applications.

Since the network traffic is generated by users day after day, it is also desired to have an online version of the clustering algorithm. The main difficulty for an online algorithm is that relational clustering results usually depend on the order of adding new records.

7. REFERENCES

[1] I. Bhattacharya and L. Getoor. Collective entity resolution in relational data. *ACM Transactions on Knowledge Discovery from Data*, 1(1):5, 2007.

[2] V. Blondel, J. Guillaume, R. Lambiotte, and E. Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008.

[3] U. Brandes, D. Delling, M. Gaertler, et al. On modularity clustering. *IEEE Transactions on Knowledge and Data Engineering*, 20(2):172–188, 2007.

[4] V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: A survey. *ACM Computing Surveys (CSUR)*, 41(3):15, 2009.

[5] M. Crotti, M. Dusi, F. Gringoli, and L. Salgarelli. Traffic classification through simple statistical fingerprinting. *ACM SIGCOMM Computer Communication Review*, 37(1):16, 2007.

[6] N. Duffield, P. Haffner, B. Krishnamurthy, and H. Ringberg. Rule-based anomaly detection on IP flows. In *IEEE INFOCOM*, 2009.

[7] J. Erman, M. Arlitt, and A. Mahanti. Traffic classification using clustering algorithms. In *Proceedings of the 2006 SIGCOMM Workshop on Mining Network Data*, 2006.

[8] S. Fortunato. Community detection in graphs. *Physics Reports*, 2009.

[9] L. Getoor and B. Taskar. *Introduction to statistical relational learning*. The MIT Press, 2007.

[10] J. Gómez, C. Gil, N. Padilla, R. Baños, and C. Jiménez. Design of a Snort-Based Hybrid Intrusion Detection System. *Distributed Computing, Artificial Intelligence, Bioinformatics, Soft Computing, and Ambient Assisted Living*, pages 515–522, 2009.

[11] J. Homer and X. Ou. SAT-solving approaches to context-aware enterprise network security management. *IEEE JSAC Special Issue on Network Infrastructure Configuration*, 2009.

[12] S. Kandula, R. Mahajan, P. Verkaik, S. Agarwal, J. Padhye, and P. Bahl. Detailed diagnosis in enterprise networks. In *Proceedings of the 2009 conference on ACM SIGCOMM 2009 conference*, 2009.

[13] T. Karagiannis, K. Papagiannaki, and M. Faloutsos. BLINC: multilevel traffic classification in the dark. *ACM SIGCOMM Computer Communication Review*, 35(4):240, 2005.

[14] C. Kemp, J. Tenenbaum, T. Griffiths, T. Yamada, and N. Ueda. Learning systems of concepts with an infinite relational model. In *Proceedings of the National Conference on Artificial Intelligence*, 2006.

[15] H. Kim, K. Claffy, M. Fomenkov, D. Barman, M. Faloutsos, and K. Lee. Internet traffic classification demystified: myths, caveats, and the best practices. In *Proceedings of the 2008 ACM CoNEXT conference*, 2008.

[16] S. Kok and P. Domingos. Statistical predicate invention. In *Proceedings of the 24th international conference on Machine learning*, 2007.

[17] S. Kok, P. Singla, M. Richardson, P. Domingos, M. Sumner, H. Poon, and D. Lowd. The Alchemy system for statistical relational AI. *Dept. of Computer Science and Engineering, Univ. of Washington, Technical Report. http://www.cs.washington.edu/ai/alchemy*, 2007.

[18] B. Long, Z. Zhang, and P. Yu. A probabilistic framework for relational clustering. In *Proceedings of the 13th ACM SIGKDD*, 2007.

[19] A. Moore and K. Papagiannaki. Toward the accurate identification of network applications. *Passive and Active Network Measurement*, pages 41–54, 2005.

[20] A. Moore and D. Zuev. Internet traffic classification using bayesian analysis techniques. In *Proceedings of the 2005 ACM SIGMETRICS*, 2005.

[21] M. Newman. Modularity and community structure in networks. *Proceedings of the National Academy of Sciences*, 103(23), 2006.

[22] M. Newman and M. Girvan. Finding and evaluating community structure in networks. *Physical review E*, 69(2), 2004.

[23] H. Poon and P. Domingos. Joint unsupervised coreference resolution with Markov Logic. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2008.

[24] H. Poon, P. Domingos, and M. Sumner. A general method for reducing the complexity of relational inference and its application to MCMC. In *Proceedings of the National Conference on Artificial Intelligence*, 2008.

[25] M. Richardson and P. Domingos. Markov logic networks. *Machine Learning*, 62(1):107–136, 2006.

[26] M. Thottan, G. Liu, and C. Ji. Anomaly detection approaches for communication networks. *Algorithms for Next Generation Networks*, pages 239–261, 2010.

[27] N. Williams, S. Zander, and G. Armitage. A preliminary performance comparison of five machine learning algorithms for practical IP traffic flow classification. *ACM SIGCOMM Computer Communication Review*, 36(5):16, 2006.

[28] T. Yang, R. Jin, Y. Chi, and S. Zhu. Combining link and content for community detection: a discriminative approach. In *Proceedings of the 15th ACM SIGKDD*, 2009.