Hypergraph Clustering for Better Network Traffic Inspection

Li Pu, Boi Faltings
Artificial Intelligence Laboratory, EPFL/IC/LIA, Lausanne, Switzerland
{li.pu,boi.faltings}@epfl.ch

Abstract

Networked computing environments are subject to configuration errors, unauthorized users, undesired activities and attacks by malicious software. These can be detected by monitoring network traffic, but network administrators are overwhelmed by the amount of data that needs to be inspected. In this paper, we describe how clustering can be used for this application to reduce the amount of data that has to be inspected. Rather than a system that attempts to directly detect malicious software and user, we propose a data-mining component to group the open ports and users in the network and let a human system administrator analyze the results. With empirical study, we show that the behaviors of softwares and users are very different. They should be clustered by the appropriate clustering algorithm accordingly.

Introduction

Networked computing environments are subject to configuration errors, undesired activities and attacks by malicious software. These can be detected by monitoring network traffic, but network administrators are overwhelmed by the amount of data that needs to be inspected. Filtering techniques are widely used by administrators to pick out the unusual traffic (Chandola, Banerjee, and Kumar 2009). But this approach requires the pre-defined filters, so it might miss some unknown anomalies. To deal with this problem, more intelligent techniques need to be adopted. Our work aims to develop a clustering strategy based on hypergraph model for the administrator so that the network traffic can be efficiently inspected with a concise list.

The real-time enterprise desktop monitoring software developed by Nexthink (http://www.nexthink.com) continuously monitors network traffic and application activities (processes) on all desktops in an enterprise network (Figure 1). Unlike traditional system logs, the monitoring software called Nexthink Collector is installed on the client side rather than the server side. This feature allows complete record of TCP/UDP connections, which includes not only the common 5-tuple (source IP, destination IP, source port, destination port, and protocol) but also the information of the application that initialized the connection (e.g. application name, application version, the user who is using the application, etc.). In this work, we empirically study two problems with the nexthink dataset, namely network service identification and user affiliation identification. Both problems are modeled with hypergraph where the goal is a partition of vertices.

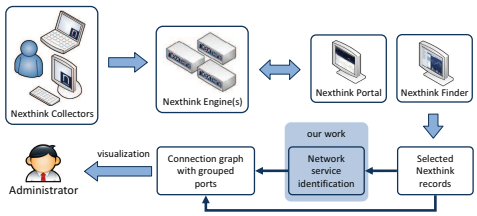

Figure 1: Nexthink tool architecture.

With Nexthink Finder the administrator can select the part of records of particular interest (by smart filters) and generate a connection graph for inspection. In the current version, connections are grouped by the port number range (e.g. 1-1000, 1001-2000, etc.), which is obviously not a good grouping strategy because in each port number range there could be ports of different functionalities. Our work aims to provide a better grouping strategy of the ports based on network service, i.e. a set of ports that serves for the same functionality. In the network service identification problem, clusters of special interest are usually small (e.g. malware), so the clustering approach that is able to isolate small clusters is more preferable than the approach that generates good average performance.

The similar idea is applied to identify the affiliation of a user. Users from the same department tend to use the same set of applications to access similar destinations since they do similar works. The affiliation information can be directly obtained from the profile of the user, but there might exist latent "affiliations" that reflect some unknown work profiles. Thus an unsupervised learning result of user groups is particularly interesting from the perspective of network management. The identified user groups can be used for generating concise connection graph in Nexthink Finder, as well as profiling work flows and improving security rules.

The above two problems in fact have different natures. Although they take the same type of evidence which consists of the connections from application to destination, the

structures of the induced hypergraph model are very different. The structure of hypergraph that involves software behaviors is less randomized than that of a hypergraph which involves human behaviors. Depending on the structure of the hypergraph, we should choose the appropriate clustering algorithm accordingly. A measure called *hyperedge affinity* is proposed in the paper for studying the structure of the hypergraph.

Our previous work has isolated malware in the network traffic (Pu et al. 2010), while in this paper we compare a broader range of state-of-the-art algorithms with the applications of clustering network traffic, including a new task that identifies the affiliation of users. We also study the strategy of choosing suitable clustering algorithm with hyperedge affinity before any partition result is produced.

Related work

There are several published works that address similar problems in the context of networks. Karagiannis et al. developed an approach called BLINC to classify network flows according to the traffic patterns at three different levels (Karagiannis, Papagiannaki, and Faloutsos 2005). 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 finer granularity and more complete. Bayer et al. developed a scalable clustering algorithm for grouping malware execution traces by computing the approximated nearest neighbors with a technique called locality sensitive hashing (Bayer et al. 2009). Perdisci et al. tested different clustering algorithms (single-linkage hierarchical clustering, complete-linkage hierarchical clustering, and x-means) on the malware signature generation problem (Perdisci, Lee, and Feamster 2010). The single-linkage hierarchical clustering algorithm produces the best result.

Problem Statement

From the Nexthink records, five types of entities are considered: *connections* that represent TCP/UDP sessions; *applications* that denote processes of the same executable filename; *destinations* are servers of an unique IP address that open some ports to the applications; *ports* are combinations of port type (TCP or UDP) and port numbers; and *users* who are using the applications to access destinations. The source IP address and source port information in the original records are omitted because we treat the same application on different client machines equally. Then a *network service* is defined as a cluster that consists of one or more ports. The goal of the network service identification problem is to extract network services in the dataset and identify the machines/applications which provide/use those services.

The only evidence we extract from the original records 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 simple evidence generating rule helps to merge similar ports into one cluster and actually defines a group of subsets where each subset can be named by the application-destination pair. In the end the whole dataset is transformed into subsets of ports. These subsets are called *set evidence*. It can be observed that the ports are not always used for the registered purpose as shown in IANA (Internet Assigned Numbers Authority), but we find that the functionality of one port is quite stable within the enterprise network. So it is reasonable to assume that one port only belongs to one service.

The above process of extracting set evidence from raw data assumes the correlation between the service and a collection of distinguishable application-destination pairs. Although this assumption is quite tenable in our dataset, it could be circumvented by malware that mimics the behavior of normal software such that malicious ports are identified as the same service as normal ports. But in order to hide itself, the malware has to know the ports of other services. Thus we can offset the impact of this adversarial behavior by excluding the well known ports to make it harder to hide in a known service. As a transport layer approach, the assumption is also vulnerable to the attacks that directly use well known static ports. But this may cause port conflicts and easy-identifiable anomaly.

The evidence used for user affiliation identification is similar to that of the network service problem. Two or more users are likely to be in the same affiliation if they use the same application to access the same destination. Since network users usually spend a lot of time on destinations outside the enterprise network, all the IP addresses (not only local IP addresses) are included as destinations in the set evidence. We also reduce the number of destinations by replacing the IP addresses with its corresponding net-names and/or domain names. This procedure usually reduces the raw IP addresses to less than 10% destinations.

Hypergraph model

Every port and every user in the processed data can be expressed as a feature vector whose length equals the number of set evidence. The entry in the feature vector is set to 1 if the port or user is contained in the set evidence. By computing the distances between the feature vectors of ports or users, one can apply any distance-based clustering algorithm (e.g. k-means) to the processed data. In this work, however, we use the hypergraph model to represent the co-occurrence relationship among the ports/users.

A hypergraph $\mathcal{H} = \{\mathcal{V}, \mathcal{E}\}$ consists of a finite *vertex set* $\mathcal{V}$ and a finite *hyperedge set* $\mathcal{E}$. Each hyperedge $e \in \mathcal{E}$ is a subset of $\mathcal{V}$, i.e. $e \subseteq \mathcal{V}$. Each hyperedge e is associated with a positive real-valued weight $w(e)$. Let W denote the set of weights. We call $\mathcal{H} = \{\mathcal{V}, \mathcal{E}, W\}$ a *weighted hypergraph*. A hyperedge e is *incident* with a vertex v if $v \in e$. The weighted degree of a vertex is $deg(v) = \sum_{v \in e, e \in \mathcal{E}} w(e)$. The degree of a hyperedge is $deg(e) = |e|$. For the network service problem, the set of all ports is taken as the vertex set, and each piece of set evidence is translated to a hyperedge. For the user affiliation problem, the set of all users is taken as the vertex set, and each piece of set evidence forms a hyperedge. In this paper, the weights of all hyperedges are set to 1 for simplicity.

From a weighted hypergraph $\mathcal{H}$, we can build a simple graph $\bar{\mathcal{H}} = \{\mathcal{V}, \bar{\mathcal{E}}\}$ with edge set $\bar{\mathcal{E}}$ where an edge is added

between $v_i, v_j \in \mathcal{V}$ if v_i and v_j belong to the same hyperedge in $\mathcal{E}$. Each edge $\bar{e} = (v_i, v_j) \in \bar{\mathcal{E}}$ has weight $w(\bar{e}) = \sum_{v_i,v_j \in e} w(e)/deg(e)$. This simple graph $\bar{\mathcal{H}}$ is called the *induced graph* of $\mathcal{H}$.

A *k-way partition* $\mathcal{T}$ consists of k clusters where each cluster $t \subseteq \mathcal{V}$, $\bigcup_{t\in\mathcal{T}} t = \mathcal{V}$, and $\forall t_1, t_2 \in \mathcal{T}$, $t_1 \neq t_2$, $t_1 \cap t_2 = \emptyset$. For cluster t, let t^c denote the compliment of t. The volume of t is $Vol(t) = \sum_{v\in t} deg(v)$. A hyperedge e is said to be incident with a cluster t if $e \cap t \neq \emptyset$.

Extracting Clusters with Hypergraph Min-Cut

Unsupervised learning on simple graphs or hypergraphs is a well-studied topic with many applications. Various techniques can be used to form the partition according to the clustering fitness measures, e.g. graph cut (Schaeffer 2007), hypergraph cut (Karypis and Kumar 1999), graph modularity (Brandes et al. 2007), kernel-defined measure (Kulis et al. 2009) , etc. The idea of cut-based approaches is to search for the partition that minimizes the cost of (hyper)edges on the partition boundary. There are various definitions of the cost of (hyper)graph cut, which lead to different min-cut approaches. We first introduce two existing definitions of hypergraph cut and then propose a hierarchical clustering algorithm based on a non-pairwise hypergraph cut. The hypergraph modularity is used to determine the optimal number of clusters.

Hypergraph cut

Let $D(e) = \{t|e\cap t \neq \emptyset, t \in \mathcal{T}\}$ denote the incident clusters of e. For each cluster t, let $A(t) = \{e|e \cap t \neq \emptyset, \forall t' \neq t, e \cap t' = \emptyset\}$ denote the set of hyperedges that are only incident with t, $C(t) = \{e|e \cap t \neq \emptyset\}$ denote the set of hyperedges that are incident with t, and $B(t) = C(t) \setminus A(t)$ denote the set of hyperedges that are not only incident with t but also with some other cluster.

By (Karypis and Kumar 1999; Xiao 2010), the k-cut of a hypergraph is the weighted sum of hyperedges that have to be removed to produce k disconnected parts, i.e.

$$kCut(\mathcal{H}, \mathcal{T}) = \sum_{e\in\cup_t B(t)} w(e). \quad (1)$$

By (Zhou, Huang, and Scholkopf 2007), the cut volume of t is $Vol_2(\partial t) = \sum_{e\in B(t)} w(e)|e\cap t||e\cap t^c|/deg(e)$ where we use subscript 2 because $Vol_2(\partial t)$ actually computes the pairwise connections of all vertices in the cut. The *normalized hypergraph cut* is then defined as

$$NHC_2(\mathcal{H}, \mathcal{T}) = \sum_{t\in\mathcal{T}} Vol_2(\partial t)/Vol(t), \quad (2)$$

where the normalizer $Vol(t)$ introduces a penalty on extremely small clusters and encourages more balanced cluster sizes. One can easily show that the normalized hypergraph cut equals to the normalized simple graph cut on the induced graph $\bar{\mathcal{H}}$ from the original hypergraph $\mathcal{H}$. Our previous work has shown the connection between NHC_2 cut and the objective of Markov logic in a network service identification problem (Pu et al. 2010).

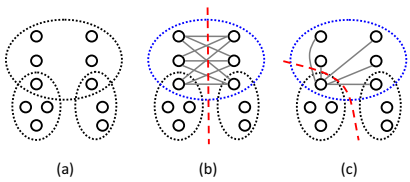

Figure 2: Different definitions of hypergraph cut.

On the other hand, from that fact that $\cup_{t\in\mathcal{T}} B(t) = \{e|D(e) > 1\}$, it is reasonable to directly count the number of pieces of the cut hyperedge as the cost of the cut. Because each hyperedge must be incident with at least one cluster, it only adds a constant (the sum of hyperedge weights) to the total cost if we count the hyperedge pieces in $C(t)$ instead of $B(t)$. For cluster t, we define another cluster cut volume $Vol_0(\partial t) = \sum_{e\in C(t)} w(e)$, which leads to the following definition of hypergraph cut,

$$HC_0(\mathcal{H}, \mathcal{T}) = \sum_{t\in\mathcal{T}} Vol_0(\partial t). \quad (3)$$

HC_0 counts the weighted pieces of cut hyperedges instead of the weighted sum of cut edges in the induced graph. It can be shown that $HC_0 = \sum_{e\in\mathcal{E}} w(e)|D(e)|$. This means we make the same penalty regardless the distribution of vertices across the partition boundary. As shown in Figure 2 (a), we want to divide the hypergraph with 3 hyperedges into 2 parts. The NHC_2 cut by the partition in Figure 2 (b) is 0.46 and by the partition in Figure 2 (c) is 0.27. This pairwise counting approach would prefer an uneven partition of the hyperedge rather than an even partition. It also means that large clusters will dominate the partition while the small clusters have smaller influence on the cut. With the non-pairwise cut HC_0 we can avoid this problem and keep the cuts in both Figure 2 (b) and (c) the same value. The k-cut has the same cost as HC_0 if $k = 2$, but we will show in the experiments that k-cut is not appropriate for clustering tasks when $k > 2$.

For NHC_2 and HC_0 the following property would help to group vertices in the preprocessing stage.

Proposition 1. *For any* $F \subseteq \mathcal{E}$, $K = (\bigcap_{e\in F} e) \setminus (\bigcup_{e\in\mathcal{E}\setminus F} e)$, *there exists a minimum* HC_0 *partition* $\hat{\mathcal{T}}$ *such that* $\exists t \in \hat{\mathcal{T}}$ *and* $K \subseteq t$. *(Proof omitted.)*

The vertices in K can be seen as "flat areas" in the hypergraph. The partition can only be placed on the boundary of "flat areas", so we can assign the vertices in K to the same cluster in the preprocessing stage. But the number of vertices in K depends on the choice of F.

Hypergraph modularity

If no prior knowledge or constraint about the number of clusters is given, minimizing the hypergraph cut would produce a trivial solution with a single cluster. To tackle this problem, a more sophisticated clustering fitness measure called modularity is proposed to consider both non-cut and cut edges in a single real-value number. Modularity is defined as the actual edge weights in the cluster minus the

expected edge weights in the cluster as if the edges are randomly placed between vertices with an expected probability (Newman and Girvan 2004; Brandes et al. 2007). Inspired by the simple graph modularity, for a weighted hypergraph $\mathcal{H} = \{\mathcal{V}, \mathcal{E}, W\}$ and partition $\mathcal{T}$ we extend it to *hypergraph modularity* based on the distribution of hyperedge weights

$$Q(\mathcal{H}, \mathcal{T}) = \sum_{t \in \mathcal{T}} \left(\frac{w(A(t))}{w(\mathcal{E})} - \left(\frac{\sum_{e \in C(t)} \frac{w(e)}{|D(e)|}}{w(\mathcal{E})} \right)^2 \right), \quad (4)$$

where $w(A(t)) = \sum_{e \in A(t)} w(e)$ and $w(\mathcal{E}) = \sum_{e \in \mathcal{E}} w(e)$. If all the hyperedges in $\mathcal{E}$ contain exact two vertices, we get the modularity definition for a simple graph.

One can show that hypergraph modularity has similar bounds as modularity for simple graph, which is from -1 to 1. It suggests a good cluster structure if $Q(\mathcal{H}, \mathcal{T})$ is close to 1 and totally non-cluster structure if it is close to -1.

Algorithm

The combinatorial problems of finding the partition that minimizes NHC_2, HC_0, and maximizes (hyper)graph modularity are proven to be NP-hard (Shi and Malik 2002; Brandes et al. 2006). In this work we use a simple agglomerative hierarchical clustering algorithm that minimizes the hypergraph cuts P mentioned in previous section (P could be NHC_2 or HC_0), and determines the optimal number of clusters with hypergraph modularity. The algorithm works as following,

Algorithm 1 : $\mathcal{T}$ = **AgglomerativeClustering**($\mathcal{H}$)

1: Build an initial partition $\mathcal{T}$ in which each cluster contains exact one vertex. Group the must-link vertices according to Proposition 1.
2: Choose $t_1, t_2 \in \mathcal{T}$ and merge them to get a new partition $\mathcal{T}'$ such that the improvement of clustering fitness measure $\Delta P(\mathcal{H}, \mathcal{T}) = P(\mathcal{H}, \mathcal{T}) - P(\mathcal{H}, \mathcal{T}')$ is maximized.
3: Set the output to $\mathcal{T}'$ if $Q(\mathcal{H}, \mathcal{T}')$ is larger than that of any previous iterations. Let $\mathcal{T} = \mathcal{T}'$ and repeat step 2 until $|\mathcal{T}| = 1$.

Usually the above algorithm gives some partition with $|\mathcal{T}| > 1$ because the modularity reaches a local maximum. In step 2 not all the entries of $\Delta P(\mathcal{H}, \mathcal{T})$ need to be recomputed after merging clusters t_1 and t_2 because the clusters other than t_1, t_2 remain the same.

The complexity of the algorithm is following: in step 1 each vertex is chosen in turns as the seed of finding F in Lemma 1. One can first sort the vertices by the adjacency matrix to make the process faster, so the complexity of step 1 is $O(|\mathcal{V}| \log |\mathcal{V}|)$. In step 2 one has to update the improvement of P for the newly combined cluster, which takes time $|\mathcal{E}|$. Then each newly computed improvement is inserted into a sorted list for choosing the next best combination, which takes time $\log(|\mathcal{V}|)$. Finally step 2 could be repeated at most $|\mathcal{V}| - 1$ times, so the worst case complexity of the algorithm is $O(|\mathcal{E}||\mathcal{V}| \log |\mathcal{V}|)$, and the space complexity is $O(|\mathcal{V}|)$.

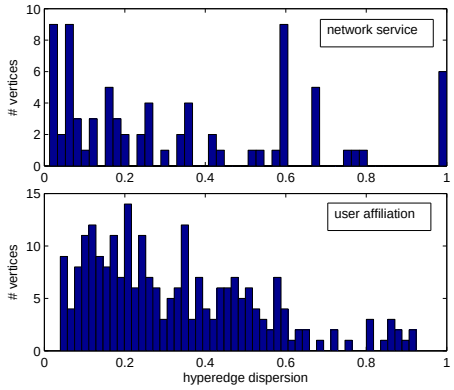

Figure 3: The hyperedge affinity distribution of two hypergraphs.

Hyperedge Affinity

The purpose of hypergraph modularity is to estimate the degree of existence of cluster structure in a hypergraph, but it requires a partition to compute the result. Can we make similar estimation without knowing any partition? The idea is to examine the degree of overlapping between the hyperedges that are incident with a chosen vertex. If these hyperedges cover the same set of vertices, it suggests a strong cluster structure. The hyperedge affinity of a hypergraph $\mathcal{H}$ is a mapping $d : \mathcal{H} \to \mathbb{R}^{|\mathcal{V}|}$ where each dimension d_i corresponds to a vertex v

$$d_i = \frac{2 \sum_{e_1, e_2 \in C(v)} |e_1 \cap e_2|}{|\cup C(v)||C(v)|(|C(v)| - 1)|}, \quad (5)$$

where $C(v)$ is the set of hyperedges that are incident with v. The denominator is just a normalizer that takes the largest possible value of the numerator. It is easy to verify that $0 < d_i \leq 1$. If most vertices have large d_i, it is likely to make a good partition for the hypergraph.

The histograms of $d(\mathcal{H})$ for network service and user affiliation are shown in Figure 3. For the hypergraph of network service, although there exists vertices of low hyperedge affinity, some vertices have very high d_i. Those vertices of high d_i could form several "cores" of the clusters, which allows good partitions. For the hypergraph of user affiliation, more vertices stay in the small hyperedge affinity range. It indicates that the behavior of users is more dispersive than the behavior of softwares. In the histogram of network service there are several peaks generated by some fixed routines in the software, while the histogram of user affiliation is smoother.

We have studied the hyperedge affinity with other datasets. Usually vertices of $d_i > 0.9$ indicate very strong co-occurrence relations and these relations are normally generated by non-human mechanism (e.g. software). On the other

hand, a concentration of vertices in the interval $d_i < 0.2$ suggests that the hypergraph does not carry cluster structure. The experimental results in the next section show that certain algorithms have better performances on hypergraph generated by non-human mechanism, and vice versa. The hyperedge affinity can be used to choose a suitable clustering algorithm.

Experimental Results

Hypergraph clustering approaches

Besides Algorithm 1 based on NHC_2 and HC_0, we also use several other popular clustering algorithms. First k-means is taken as the baseline in the experiments which directly obtains a feature vector of length $|\mathcal{E}|$ for each vertex where the entry is 1 if the vertex is included in the hyperedge, otherwise 0. If all the feature vectors are written in a row, it forms a matrix H of size $|\mathcal{E}| \times |\mathcal{V}|$ which is called the incident matrix. The single-linkage algorithm is reported to be the best clustering approach for network traffic or other signatures generated by softwares (Bayer et al. 2009; Perdisci, Lee, and Feamster 2010). We use the cosine distance for computing the pairwise distance in the single-linkage algorithm. The third algorithm called hMETIS is developed by Electronic Design Automation researchers (http://glaros.dtc.umn.edu/gkhome/views/metis). In hMETIS a multi-level clustering algorithm is implemented to minimize the k-cut.

The same feature vectors are also processed by the algorithm that adopts the spectral clustering (SC) approach described in (Zhou, Huang, and Scholkopf 2007). This approach is operated on the induced simple graph of the original hypergraph. The hypergraph Laplacian is computed from the normalized Laplacian of the induced graph. All the vertices are then mapped into points in a k dimension Euclidian space by the eigenvectors of the hypergraph Laplacian. For $k > 2$ a distance-based clustering algorithm has to be applied to read the cluster information from the mapped points.

We also take the non-negative matrix factorization (NMF) approach as an extension of k-means (Ding, He, and Simon 2005). The idea is to decompose the incident matrix H into two non-negative matrices W and G of low rank such that the difference $\parallel H - W^T G \parallel$ is minimized. If W and G have k rows, the rows of W are the k cluster centroids and columns of G are the cluster indicators for the vertices. We simply choose the index of maximum entry of each column in G as the cluster number of the corresponding vertex.

Finally the theory of centrality and PageRank (Bonacich and Lloyd 2001) is extended to fill in the zero entries of the feature vectors such that the distance between vertices in the same cluster becomes smaller. For each vertex v, we assign a non-negative importance score $s_v(e)$ for every hyperedge e. The score of hyperedges that are directly incident with v is set to 1 as in the feature vector. $s_v(e)$ of other hyperedges is determined by $s_v(e) = \sum_{e_1 \in N(e)} w(e_1) s_v(e_1)$, where $N(e)$ denotes the set of hyperedges that share at least one vertex with e. This computation is similar to a random walk on hyperedges with teleportation, and helps to identify other important hyperedges which is not directly incident with v in the original hypergraph due to noises. The "improved" features filled by the importance scores are then processed by another distance-based clustering approach. We call this approach hyperedge centrality (HCe).

In the experiments, if the algorithm requires the number of desired clusters as input, we set it to the number of clusters in ground-truth.

Network service identification

Two classes of datasets are used for network service identification, i.e., synthetic data and real data collected by Nexthink. 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 by 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 is assigned to each pair. Then some connections are generated by associating ports in the corresponding services to application-destination pairs. To test the performance of the algorithm under noise, the ports of a fraction β of connections are chosen from a random service, which brings incorrect hyperedges into the dataset. In each synthetic dataset, there are 8000 connections, 80 applications, 40 destinations, and 180 ports which form 40 services.

The dataset collected by Nexthink contains records from July 2008 to September 2008 in a real enterprise network. There are 13774 connections, 320 applications, 840 destinations, and 474 ports. All connections are recorded in Windows systems. The connections with destinations outside the enterprise network are omitted, because we are only concerned about the services inside the local network. For the real data, we find many ports are not used as the registered functionality in IANA, so it is hard to label all the ports.

The algorithms are first tested on the synthetic data. We use the micro and macro average F-score to evaluate the results. Another two internal validation indices are also adopted to assess the goodness of the clustering results. The first index *Dunn index* is designed to identify dense and well-separated clusters and defined as the ratio between the minimal intra-cluster distance to maximal inter-cluster distance (Brun et al. 2007). A partition is better if it has larger Dunn index. The second index *Davies-Bouldin index* is measured by the average distance of points in a cluster to the center of the cluster and the distance between cluster centers (Davies and Bouldin 1979). A smaller value of Davies-Bouldin index indicates a better cluster structure.

Figure 4 shows the average performance on datasets of different degrees of noise and different degrees of unbalanced cluster sizes. We could confirm the fact that the single-linkage algorithm gives the best result in all cases. Since single-linkage finds the nearest neighbor clusters in each step and combines them into a new cluster, the result only depends on the local distances between a vertex and its neighbors. A vertex of high hyperedge affinity value indi-

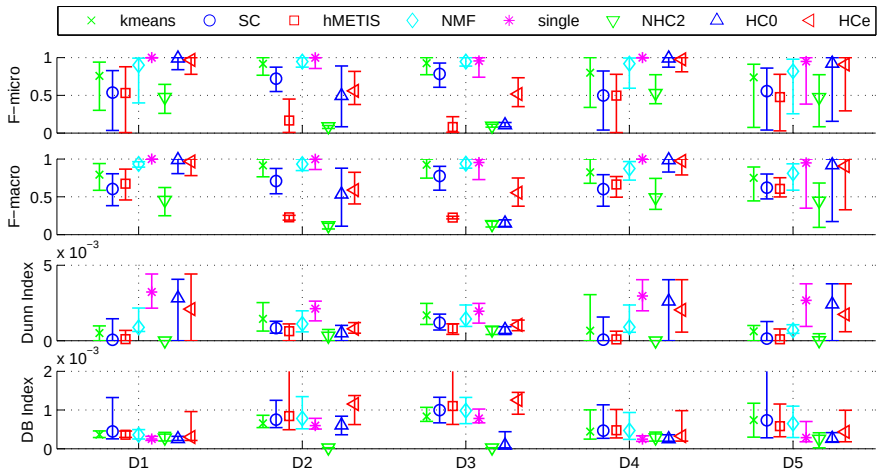

Figure 4: Performance of different algorithms on synthetic datasets. The parameters of the datasets are $D1$: $\alpha = 2$, $\beta = 0$, $D2$: $\alpha = 2$, $\beta = 0.05$, $D3$: $\alpha = 2$, $\beta = 0.1$, $D4$: $\alpha = 5$, $\beta = 0$, $D5$: $\alpha = 10$, $\beta = 0$. For each dataset the performance is averaged on 50 trails. The maximal and minimal values are also shown in the figure.

cates that its neighbors are very close. Thus single-linkage algorithm could easily identify the clusters from the local connectivity. If we known that the dataset is generated by non-human mechanism (e.g. from hyperedge affinity analysis), single-linkage would be the first choice.

One can observe that the HC_0-based algorithm produces good average results (better than that of NHC_2-based algorithm) when the data is noiseless, but it becomes not so good when the data is noisy. Essentially the HC_0-based algorithm is similar to the centroid linkage algorithm, so the result varies due to the change of centroids in the presence of noises. The performance of hyperedge centrality approach also decreases a lot with noisy data, because the connectivity between hyperedges used for computing importance scores is not reliable anymore.

k-means, spectral clustering, and non-negative matrix factorization produce relatively good results in all cases. The latter two methods can be seen as modifications of k-means. What is interesting is that the performance of these three methods becomes better when the data is noisy. This follows the fact that when the data manifold is fractured, some points might be near to the centroid of their true cluster.

The results of hMETIS are not as good as other cut-based algorithms even when the data is noiseless mainly due to the k-cut criterion. One can imagine that with the k-cut criterion big hyperedges are always first removed such that more disconnected clusters can be produced. But with HC_0 or NHC_2, removing big hyperedge would introduce more penalties because big hyperedge usually intersects with more clusters.

In Table 1, the individual F-scores of the smallest 2 clusters and the biggest 2 clusters are computed for all the algorithms. If we compare the unbalanced case ($D5$, $\alpha = 8$) with the more balanced case ($D1$, $\alpha = 2$), one can observe that single-linkage and HC_0 always produce the same good results on small and big clusters, while the other algorithms usually show better F-scores on big clusters and worse F-scores on small clusters. In the network service identification problem, the services of special interest are usually small clusters (rare behaviors, malware, etc.), so the single-linkage and HC_0 algorithm are more capable of isolating small interested services from a huge amount of traffic.

kmeans	TCP2967, UDP2967, UDP38293 rtvscan.exe, savroam.exe 10.0.0.8, 10.100.0.5, 10.104.0.8, ...
SP	TCP2967, UDP1281, UDP2967, ... rtvscan.exe, savroam.exe 10.0.0.8, 10.100.0.5, 10.104.0.8, ...
hMETIS	TCP2967, UDP1153, UDP1179, ... outlook.exe, rtvscan.exe, savroam.exe 10.0.0.8, 10.10.3.21, 10.100.0.5, ...
NMF	TCP2967, TCP8080, UDP8080 rtvscan.exe, yahoomessenger.exe 10.0.0.8, 10.100.0.5, 10.104.0.8, ...
single	TCP2967, UDP2967, UDP38293 rtvscan.exe, savroam.exe 10.0.0.8, 10.100.0.5, 10.104.0.8, ...
NHC2	TCP2967, UDP1281, UDP2967, UDP38293 rtvscan.exe, savroam.exe 10.0.0.8, 10.100.0.5, 10.104.0.8, ...
HC0	TCP2967, UDP1281, UDP2967, UDP38293 rtvscan.exe, savroam.exe 10.0.0.8, 10.100.0.5, 10.104.0.8, ...
HCe	TCP2967 rtvscan.exe, 10.0.0.8, 10.100.0.5, 10.104.0.8, ...

Table 2: Example of network service detected from Nexthink dataset by different algorithms.

The algorithms usually output about 70 to 85 services (clusters) on Nexthink dataset when the maximum modularity is reached. We identified some network services in the dataset, where one example is shown in in Table 2. Usually the clusters generated by single-linkage, NHC_2 and HC_0 coincide with each other, which indicate that the local connectivity in Nexthink data is very strong. The normalized

	pre.	rec.	F-score	pre.	rec.	F-score	pre.	rec.	F-score	pre.	rec.	F-score
cluster $D1$		S1			S2			L1			L2	
k-means	0.877	0.993	0.906	0.901	0.987	0.918	0.936	0.970	0.941	0.877	0.960	0.897
SP	0.754	0.947	0.769	0.723	0.953	0.752	0.762	0.977	0.817	0.845	0.987	0.883
hMETIS	0.695	1.000	0.773	0.656	1.000	0.749	0.929	0.953	0.929	0.926	0.942	0.919
NMF	0.861	1.000	0.902	0.909	1.000	0.936	0.976	0.963	0.963	0.992	0.955	0.967
single	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
NHC_2	0.490	0.947	0.502	0.490	0.920	0.496	0.546	0.917	0.550	0.681	0.876	0.635
HC_0	0.988	1.000	0.991	0.988	1.000	0.991	1.000	1.000	1.000	1.000	1.000	1.000
HCe	1.000	0.993	0.996	0.990	0.987	0.985	0.989	0.990	0.988	1.000	0.990	0.994
cluster $D5$		S1			S2			L1			L2	
k-means	0.915	1.000	0.937	0.901	0.990	0.925	0.984	0.910	0.933	0.942	0.914	0.910
SP	0.691	1.000	0.745	0.664	0.940	0.681	0.824	0.960	0.859	0.851	0.989	0.897
hMETIS	0.517	1.000	0.616	0.648	1.000	0.716	0.967	0.907	0.926	0.972	0.931	0.943
NMF	0.619	1.000	0.693	0.718	1.000	0.782	0.989	0.965	0.971	0.988	0.959	0.968
single	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
NHC_2	0.816	1.000	0.829	0.575	0.940	0.573	0.706	0.911	0.701	0.701	0.915	0.690
HC_0	0.983	1.000	0.986	0.968	1.000	0.973	1.000	0.997	0.998	0.995	0.998	0.996
HCe	0.894	1.000	0.917	0.910	1.000	0.931	0.994	1.000	0.996	1.000	0.998	0.999

Table 1: The precision, recall, and F-score of the 2 smallest clusters (S1, S2) and the 2 biggest clusters (L1, L2) with datasets $D1$ ($\alpha = 2$) and $D5$ ($\alpha = 8$). The performance is averaged on 50 trails.

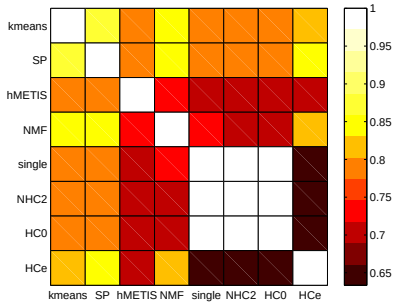

Figure 5: The normalized mutual information between the partitions by different algorithms on Nexthink dataset.

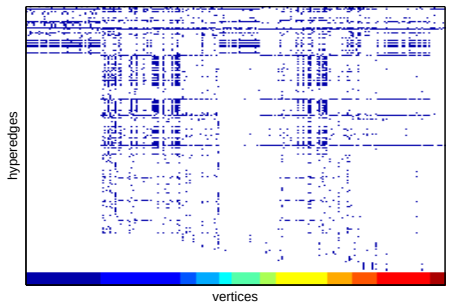

Figure 6: A spy plot of the hypergraph for user affiliation identification. Each vertex represents a user, and each hyperedge denotes a set evidence. There is a dot if the user is included in the set evidence. The vertices are sorted by labels and different colors on the bottom indicate the labels.

mutual information (NMI) between the partitions by different algorithms is shown in Figure 5. We follow the definition of NMI in (Strehl and Ghosh 2003).

User affiliation identification

Another dataset used for user affiliation identification is collected by Nexthink from computer rooms where the users from different departments could access local and Internet contents. With the data collected in a period of two weeks, we get 237 users and 1604 hyperedges after removing those hyperedges which consist of only one user. The affiliation of users is directly taken from the user profile and used as labels. Although the labels may not reflect the underlying group structure of users, we use them for evaluating clustering algorithms. In the label list, there are 12 affiliations. The smallest and largest affiliation contains 7 users and 45 users respectively. A cleaned spy plot of the hypergraph is shown in Figure 6.

The F-scores and modularity are shown in Table 3. What is different from the data generated by a non-human mechanism, we can observe an almost reversed ranking of algorithms. The algorithms that are similar to k-means show better performances than the hierarchical clustering methods. This is mainly due to the fact that the assumed manifold structure behind the nearest neighbor rule becomes unreliable when the user behavior is more dispersive. The F-scores indicate that the partitions of all algorithms are very different from labels. But a partial reason of the low F-score is that the labels from user profiles are not reflecting the real underlying cluster structure. A maximum modularity partition of HCe algorithm consists of only 3 clusters instead of 12 clusters in the labels (Figure 7). The modularity of the labels is actually the lowest among all algorithms.

Conclusion

In this work, the network service identification problem and user affiliation identification problem are studied. The results help to group the open ports and users in the network such that a human system administrator could efficiently an-

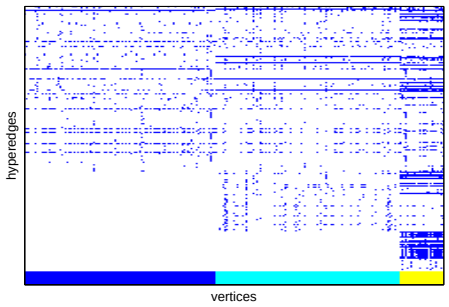

Figure 7: A spy plot of user affiliation with maximum modularity partition from HCe algorithm.

	F-micro	modularity
k-means	0.262 ± 0.085	0.037 ± 0.005
SP	0.225 ± 0.092	0.086 ± 0.012
NMF	0.267 ± 0.064	0.033 ± 0.039
single	0.198 ± 0.000	0.011 ± 0.000
NHC_2	0.203 ± 0.000	0.012 ± 0.000
HC_0	0.205 ± 0.002	0.012 ± 0.007
HCe	0.268 ± 0.000	0.058 ± 0.000
labels	–	-0.121

Table 3: The performance of different algorithms on the user affiliation identification. hMETIS is not listed in the table because it usually fails to give a valid partition.

alyze the records. By adopting the hypergraph model, we propose an agglomerative hierarchical clustering algorithm based on a non-pairwise hypergraph cut HC_0. Experimental results on both synthetic and real data show that hierarchical clustering algorithms based on the nearest neighbor rule performs better on the data generated by a non-human mechanism. With the user generated records, k-means-like algorithms usually produces better results because the assumed manifold structure behind the nearest neighbor rule is not reliable in such cases. The experimental result also confirms the indication suggested by the hyperedge affinity analysis, which allows us to identify the nature of dataset before applying any clustering algorithm.

References

[Bayer et al. 2009] Bayer, U.; Comparetti, P.; Hlauschek, C.; Kruegel, C.; and Kirda, E. 2009. Scalable, behavior-based malware clustering. In *Network and Distributed System Security Symposium (NDSS)*.

[Bonacich and Lloyd 2001] Bonacich, P., and Lloyd, P. 2001. Eigenvector-like measures of centrality for asymmetric relations. *Social Networks* 23(3):191–201.

[Brandes et al. 2006] Brandes, U.; Delling, D.; Gaertler, M.; Goerke, R.; Hoefer, M.; Nikoloski, Z.; and Wagner, D. 2006. Maximizing modularity is hard. *Arxiv preprint*.

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

[Brun et al. 2007] Brun, M.; Sima, C.; Hua, J.; Lowey, J.; Carroll, B.; Suh, E.; and Dougherty, E. 2007. Model-based evaluation of clustering validation measures. *Pattern Recognition* 40(3):807–824.

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

[Davies and Bouldin 1979] Davies, D., and Bouldin, D. 1979. A cluster separation measure. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2):224–227.

[Ding, He, and Simon 2005] Ding, C.; He, X.; and Simon, H. 2005. On the equivalence of nonnegative matrix factorization and spectral clustering. In *Proc. SIAM Data Mining Conf*, 606–610.

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

[Karypis and Kumar 1999] Karypis, G., and Kumar, V. 1999. Multilevel k-way hypergraph partitioning. In *Proceedings of the 36th annual ACM/IEEE Design Automation Conference*, 343–348.

[Kulis et al. 2009] Kulis, B.; Basu, S.; Dhillon, I.; and Mooney, R. 2009. Semi-supervised graph clustering: a kernel approach. *Machine Learning* 74(1):1–22.

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

[Perdisci, Lee, and Feamster 2010] Perdisci, R.; Lee, W.; and Feamster, N. 2010. Behavioral clustering of http-based malware and signature generation using malicious network traces. In *USENIX conference on NSDI*.

[Pu et al. 2010] Pu, L.; Faltings, B.; Yang, Q.; and Hu, D. 2010. Relational network-service clustering analysis with set evidences. In *Proceedings of the 3rd ACM workshop on Artificial intelligence and security*, 35–44.

[Schaeffer 2007] Schaeffer, S. 2007. Graph clustering. *Computer Science Review* 1(1):27–64.

[Shi and Malik 2002] Shi, J., and Malik, J. 2002. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 22(8):888–905.

[Strehl and Ghosh 2003] Strehl, A., and Ghosh, J. 2003. Cluster ensembles—a knowledge reuse framework for combining multiple partitions. *The Journal of Machine Learning Research* 3:583–617.

[Xiao 2010] Xiao, M. 2010. Finding minimum 3-way cuts in hypergraphs. *Information Processing Letters* 110(14-15):554–558.

[Zhou, Huang, and Scholkopf 2007] Zhou, D.; Huang, J.; and Scholkopf, B. 2007. Learning with hypergraphs: Clustering, classification, and embedding. *Advances in Neural Information Processing Systems*.