Chapter 1

Fraud Detection - Dense Subgraph Detection

Contributed by Tong Zhao

1.1 Introduction

Fraud behaviors can be spotted everywhere on online applications such as social networks where the behavior data can be represented as large bipartite graphs which consist of links between followers and followees. Detecting the fraudsters such as bot followers tend to be an unsupervised problem as the size of such social network graphs are huge and labeling even a small portion of the graph will take too much human effort. Luckily, fraudulent actions such as fake followers usually result with creating subgraphs with unexpected high density. For example, as a large number of follower buyers buy followers from one major follower seller, these follower buyers together with the bot followers controlled by the seller will form a subgraph with high density. Therefore, many existing detection methods [17, 29, 27] estimate the suspiciousness of users by identifying whether they are within a dense subgraph.

1.2 The Problem as a Graph

Here we define the definitions of density for graphs according to [6, 14, 20].
Let $G = (V, E)$ be a undirected graph with vertices V and edges $E \subseteq V \times V$. $E(V)$ stands for the set of edges induced by V, that is

$$E(V) = \{(i, j) \in E : i \in V, j \in V\}$$

Then the density of subgraph induced by $S \subseteq V$ can be defined as

$$d(S) = \frac{|E(S)|}{|S|}$$

Note that $2d(S)$ is actually the average degreee of the subgraph induced by S. The Densest Subgraph problem can be defined as

$$DS(G) = \max_{S \subseteq V} \{d(S)\}$$

For directed graphs. Let $G = (V, E)$ be a directed graph with vertices V and edges $E \subseteq V \times V$. $E(S, T)$ stands for the set of edges from vertices in $S \subseteq V$ to vertices in $T \subseteq V$, that is

$$E(S,T) = \{(i,j) \in E : i \in S, j \in T\}$$

Then the density of subgraph induced by $S, T \subseteq V$ can be defined as

$$d(S,T) = \frac{|E(S,T)|}{\sqrt{|S||T|}}$$

The Densest Subgraph problem can be defined as

$$DS(G) = \max_{S,T \subseteq V} \{d(S,T)\}$$

1.3 Some Realistic Data Sets

The data sets that this application encounter can come from social networks (botnet followers.) Thus the graph size can be huge. For example, the Twitter follower-followee data set used in Fraudar [17] contains 41.7 million nodes with 1.47 billion edges. Similar data sets for social networks such as Twitter can be found on SNAP [23] or other platforms. It is intuitive that the size of graphs for such problem in real industry will be growing continuously. Hence it is important for the algorithms to have a linear or near linear run-time or be able to parallelize.

1.4 Dense Subgraph Detection-A Key Graph Kernel

Multiple algorithms exists for detecting the dense subgraphs. One commonly used algorithm is proposed by Charikar in 2000 [6], which is an approximation algorithm by greedy approach. Although Charikar's algorithm sacrificed quality of the result subgraph for much better time complexity, this algorithm still has a provable 2-approximation guarantee [21]. That is, if the densest existing subgraph S' has edge density of $d(S') = \lambda$, the result subgraph S of Charikar's algorithm will have edge density of $d(S) \geq \lambda/2$.

The greedy idea of Charikar's algorithm is to remove the vertex that is least likely in the densest subgraph at each step according to certain rule. In the case of undirected graph, the rule can obviously be to remove the vertex with lowest degree. Then Charikar's algorithm can be described as following [6].

procedure DENSEST-SUBGRAPH(G)
    Input: Undirected graph $G = (V, E)$.
    Output: Dense sugraph S of G.
    $n \leftarrow |V|$
    $G_n \leftarrow G$
    for $k \leftarrow n$ down to 1 **do**
        $v \leftarrow$ the vertex with smallest degree in G_k
        Delete all edges incident on v.
        Delete all vertices with 0 degree.
        $G_{k-1} \leftarrow$ the remaining of graph G_k
    return The subgraph with maximum density amoung $G_1, G_2, \ldots, G_n$.

A detailed proof for this algorithm to achieve a 2-approximation can be found in [21].

For directed graphs, Khuller and Saha proposed a approximation algorithm based on Charikar's algorithm in 2009 [21] that ultilized the same greedy idea. The key point of this algorithm for directed graphs is to first duplicate all the vertices and construct a bipartite graph such that one copy of the vertices have only outgoing edges and the other copy of the vertices have only incoming edges. Then the algorithm can be described as following. [21]

procedure DENSEST-SUBGRAPH-DIRECTED(G)
    Input: Directed graph $G = (V, E)$.
    Output: Dense sugraph S of G.
    $n \leftarrow |V|$
    $G_{2n} \leftarrow G$
    for $k \leftarrow 2n$ down to 1 **do**
        $v \leftarrow$ the vertex with smallest degree in G_k
        if v has outgoing edges **then**
            Delete all the outgoing edges incident on v.
        else
            Delete all the incoming edges incident on v.
        Delete all vertices with 0 degree.
        $G_{k-1} \leftarrow$ the remaining of graph G_k
    return The subgraph with maximum density amoung $G_1, G_2, \ldots, G_{2n}$.

A detailed proof for this algorithm to achieve a 2-approximation can also be found in [21].

Both algorithms has time complexity of $O(|V| \log |V|)$ and space complexity of $O(|V|^2|E|)$. To evaluate the performance of these two algorithms, both density on the result subgraph and runtime can be used.

1.5 Prior and Related Work

1.5.1 Dense Subgraph Problem

The history for Dense Subgraph problem for static graphs has a rather short history, as the best exact solution was proposed by Goldberg in 1984 [14] and the best approximation algorithm so far were proposed by Charikar in 2000 [6] and Khuller and Saha in 2009 [21] for undirected and directed graphs.

Goldberg's solution works only for undirected graph. His idea was to interestingly transfer this dense subgraph problem into a well-know min-cut problem by adding two vertices s and t. Both s and t are connected with all the vertices in graph $G = (V, E)$. For each vertex $v_i \in V$, edge (s, v_i) has edge weight that is the same as the degree of v_i and edge (v_i, t) has edge weight of a positive constant c. All the edges in the original graph has an edge weight of 1. Then buy performing a min-cut call that splits s and t into two subgraphs, one of the subgraphs would be the densest subgraph of G after removing s or t.

Since min-cut problem can be solved using the parametric max-flow algorithm, this algorithm has a $O(|V||E|)$ time complexity. Thus Goldberg's algorithm is not scalable for large graphs; faster approximation algorithms are more preferred in industry situations.

In the year of 2000, Charikar [6] proposed an algorithm for detecting the dense subgraph by a greedy approximation algorithm, which we talked in the last section. In 2009, Khuller, et al. [21] further extended Charikar's algorithm to directed graphs and proved both algorithms to be 2-approximation, which are so far the best algorithms with fast run-time and theoretically guaranteed acceptable results.

1.5.2 Fraud Detection

Data-driven approaches have received great success in the field of fraud detection [22, 17]: most methods indentify unexpected dense regions of the bipartite graph, as creating fake reviews/ratings unavoidably generates edges in the graph [18, 8, 25, 38, 31, 3].

Unexpected spectral patterns. Global graph mining methods model the entire graph to find fraud based on singular value decomposition (SVD), latent factor models, and belief propagation (BP). SpokEn [29] considered the "spokes" pattern produced by pairs of eigenvectors of graphs, and was later generalized for fraud detection. fBox [30] focuses on mini-scale attacks missed by spectral techniques. BP has been used for fraud classification on eBay [27], link farming on Twitter [11], and fake software review detection [1].

Unexpected high density in subgraphs. Finding dense subgraphs has been studied from a wide array of perspectives such as mining frequent subgraph patterns [24, 39], detecting communities [12, 7, 28], and finding quasi-cliques [13, 35, 10, 32]. Charikar [6] shows that average degree of subgraph can be maximized with approximation guarantees. Tsourakakis, et al. [36] optimize the density of adjacency matrix of subgraph with quality guarantees. Hooi, et al. [17] adopt both node degree and edge density to model suspiciousness of subgraph and further increases accuracy in binary adjacency matrix of bipartite graph.

Unexpected high density in time-series. Typically there are two kinds of representation on density in time-series. One is dense subgraphs in evolving graphs [9]. CopyCatch [4] uses local search heuristics to find Δt-bipartite cores in which users consistently likes the same Facebook pages at the same short time interval. The other is dense subtensors in high-order tensors of a time dimension [26, 19, 33] or tensor streams [34]. [37, 15] consider fraud detection methods that are robust to camouflage attacks. Hooi, et al. [16] adopt a Bayesian model to find early spikes of outlier ratings in time series. All these methods focus on the time-series domain, observing changes in the behavior from system access logs rather than graph data.

1.6 A Sequential Algorithm

To implement the algorithms we talked in Section 1.4, we use Python 3 with machine learning libraries `numpy`[1] and `scipy`[2]. The implementation stores all graphs in sparse matrix format which is provide by `scipy`, so graph libraries were not used in the basic implementation.

1.7 A Reference Sequential Implementation

The source codes of Fraudar [17] is publicly available online[3]. With minor modifications on the density calculation functions, the codes of Fraudar can become the exact Python 3 implementation of the algorithm for directed graphs we talked in Section 1.4.

In order to get the algorithm's best performance, a priority tree data structure must be used to store all the degrees of vertices so that retrieving the vertex with minimum degree and updating the degree of its neighbors can be done in logarithmic time. Following is the Python implementation of priority tree in the source codes of Fraudar [17] with minor modifications.

[1] http://www.numpy.org
[2] https://www.scipy.org
[3] https://www.andrew.cmu.edu/user/bhooi/projects/fraudar/index.html

```
import math
class MinTree:
    def __init__(self, degrees):
        self.height = int(math.ceil(math.log(len(degrees), 2)))
        self.numLeaves = 2 ** self.height
        self.numBranches = self.numLeaves - 1
        self.n = self.numBranches + self.numLeaves
        self.nodes = [float('inf')] * self.n
        for i in range(len(degrees)):
            self.nodes[self.numBranches + i] = degrees[i]
        for i in reversed(range(self.numBranches)):
            self.nodes[i] = min(self.nodes[2 * i + 1], self.nodes[2 * i + 2])

    def getMin(self):
        cur = 0
        for i in range(self.height):
            if self.nodes[2 * cur + 1] <= self.nodes[2 * cur + 2]:
                cur = (2 * cur + 1)
            else:
                cur = (2 * cur + 2)
        return (cur - self.numBranches, self.nodes[cur])

    def changeVal(self, idx, delta):
        cur = self.numBranches + idx
        self.nodes[cur] += delta
        for i in range(self.height):
            cur = (cur - 1) // 2
            nextParent = min(self.nodes[2 * cur + 1], self.nodes[2 * cur + 2])
            if self.nodes[cur] == nextParent:
                break
            self.nodes[cur] = nextParent

    def dump(self):
        cur = 0
        for i in range(self.height + 1):
            for j in range(2 ** i):
                cur += 1
```

1.8 Sequential Scaling Results

All experiments are run on a 2.7 GHz Intel Core i7 Macbook Pro, 16 GB RAM, running OS X 10.14.1. The graphs used in experiments are generated by a public available Python graph generator[4], which generates bipartite graphs according to given number of vertices and average degree. The vertices of generated graphs have power-law degree distributions as shown in Figure 1.1.

In the experiments, average degree is fixed as 20 and the results are shown in Table 1.1 and

[4]https://github.com/cooperative-computing-lab/graph-benchmark

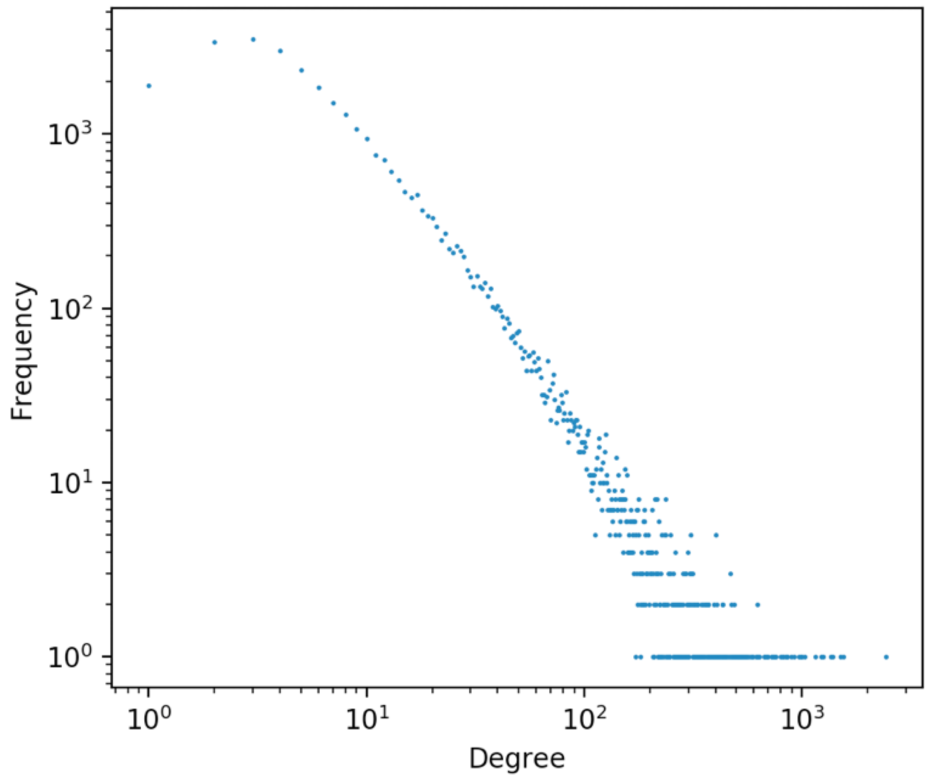

Figure 1.1: Degree distribution of generate graphs.

plotted in a log-log plot in Figure 1.2. From the results and the plot it is noticeable that this algorithm has a almost linear performance.

1.9 An Enhanced Algorithm

Two ways exist for enhancing the algorithms for dense subgraph detection. The first way is to make the algorithm possible to run on dynamic graphs. As all kinds of platforms have a large number of data coming in every single day, it is crucial for such graph algorithms to be able to run on dynamic graphs. Bhattacharya, et al. [5] proposed a space- and time-efficient algorithm for maintaining dense subgraphs on the one-pass dynamic streams in 2015, which is based on (α, D, L)-decomposition and very complicate. The second way is to make the current algorithms to be able to run on much larger datasets in a acceptable time. In the following few sections, we will focus on the second kind of enhancement.

The current Charikar's algorithm implementation loads the whole graph into RAM at the beginning. When the graph gets extremely large, it is not possible for almost all computer to do so. Hence the idea is to load only the vertices with their degree information into RAM, because the number of edges in usually far larger than the number of vertices in most of the graphs. However, the negative result of doing so is that the algorithm will have to go through the whole edge list, which is stored in the disk, every time it needs any edge information or the neighbors of one vertex. Therefore, implementing Charikar's algorithm in this way will result with a $O(n)$ pass algorithm, where n is the number of vertices in the graph and 1 pass means the algorithm needs to go through the whole graph in disk one time. [2] Apparently, when the size of the graph is large, going through

Number of vertices	Running time (s)
2^{10}	0.285
2^{11}	0.533
2^{12}	0.778
2^{13}	1.394
2^{14}	2.860
2^{15}	5.992
2^{16}	13.259
2^{17}	26.932
2^{18}	67.042
2^{19}	153.725
2^{20}	351.798
2^{21}	668.633

Table 1.1: Running time results for the sequential algorithm.

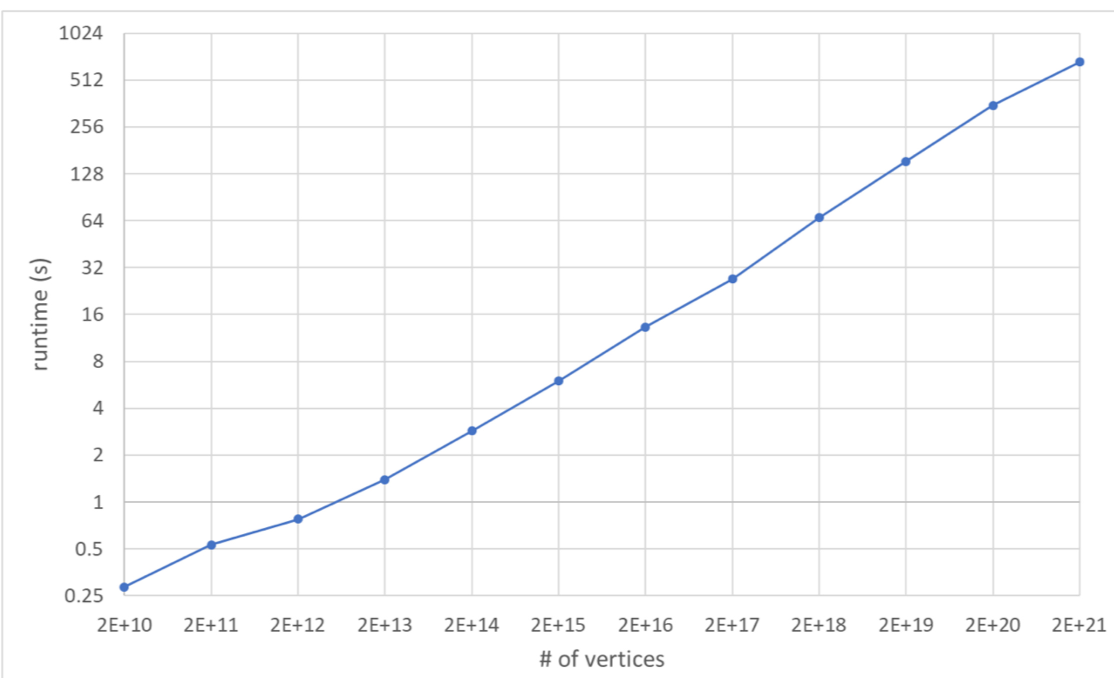

Figure 1.2: Running time results for the sequential algorithm.

the whole graph, which is stored in disk, n times will take a unacceptably long time due to the I/O limitation. Bahmani, et al. [2] proposed a modified version of Charikars algorithm, which is able to maintain a $2(1+\epsilon)$ approximation bound as well as only using $O(\log_{1+\epsilon} n)$ passes for any $\epsilon > 0$. The idea of this algorithm is to remove a set of vertices at each iteration instead of only removing only one vertex. Then the algorithm can be described as following [2].

procedure DENSEST-SUBGRAPH(G)
  Input: Undirected graph $G = (V, E)$.
  Output: Dense sugraph S of G.
  $S, V' \leftarrow V$
  while $V' \neq \varnothing$ **do**
    $A(V') \leftarrow \{i \in V' | \deg_S(i) \leq 2(1+\epsilon)d(V')\}$
    $V' \leftarrow V' \setminus A(V')$
    if $d(V') > d(S)$ **then**
      $S \leftarrow V'$
  return S

This algorithm can also be modified to be used on directed graphs like Charikar's algorithm.

1.10 A Reference Enhanced Implementation

To implement the algorithm we talked in Section 1.9, we use Python 3 without any outer libraries. In the implementation, `Counter` is used to store all the vertices with their degree information and `multiprocessing` is used for parallelization.

1.11 Enhanced Scaling Results

All experiments are run on a 2.7 GHz Intel Core i7 Macbook Pro, 16Gb RAM, running OS X 10.14.1. First of all, we tested the enhanced algorithm with a huge Twitter dataset that contains 41.7 million users (vertices) and 1.47 billion follows (edges). The graph is stored as an edge list in a `.txt` file which is over 25Gb. For a laptop with 16Gb RAM, it is usually impossible to process such a large graph as the RAM cannot even store the whole graph. We first tried the sequential Charikar's algorithm on this graph and the program was killed by the system within a few minutes. However, the enhanced algorithm, although very slow, finished the algorithm within three days. The hyperparameter ϵ was set as 0.5 in this experiment, which resulted with 44 passes of the whole dataset.

Experiments are also done on the same graphs as the sequential algorithm did, and the results are shown in Table 1.2. However, it is not reasonable to compare this set of results with the results in Table 1.1 because this enhanced algorithm is designed for graphs that are too large to fit in RAM. This enhanced algorithm sacrificed performance for space complexity by not storing any edge information in the RAM. Therefore, it is reasonable for it to have the slow results shown in Table 1.2, which fits my anticipation.

1.12 Conclusion

In conclusion, this chapter discussed about several algorithms of dense subgraph detection for fraud detection. One important enhancement for this kernel is the dense subgraph detection algorithm for dynamic graphs that I mentioned in Section 1.9, which was not implemented due to time

Number of vertices	Running time (s)
2^{10}	5.014
2^{11}	21.719
2^{12}	109.217
2^{13}	311.860

Table 1.2: Running time results for the enhanced algorithm.

limit. Therefore, any future work of this chapter should start with the algorithm proposed by Bhattacharya, et al. [5].

1.13 Response to Reviews

I made modifications according to each of the advises, specifically:

- Added more information and explain in the introduction section.
- Modified the first algorithm.
- Deleted some irrelevant sentences.
- Added one more paragraph introducing more algorithms in section 1.5.1.
- Corrected a lot of misspellings.

Bibliography

[1] Leman Akoglu, Rishi Chandy, and Christos Faloutsos. Opinion fraud detection in online reviews by network effects. In *WSDM*, pages 2–11, 2013.

[2] Bahman Bahmani, Ravi Kumar, and Sergei Vassilvitskii. Densest subgraph in streaming and mapreduce. *Proceedings of the VLDB Endowment*, 5(5):454–465, 2012.

[3] Yikun Ban, Xin Liu, Tianyi Zhang, Ling Huang, Yitao Duan, Xue Liu, and Wei Xu. Badlink: Combining graph and information-theoretical features for online fraud group detection. *arXiv preprint arXiv:1805.10053*, 2018.

[4] Alex Beutel, Wanhong Xu, Venkatesan Guruswami, Christopher Palow, and Christos Faloutsos. Copycatch: stopping group attacks by spotting lockstep behavior in social networks. In *WWW*, pages 119–130, 2013.

[5] Sayan Bhattacharya, Monika Henzinger, Danupon Nanongkai, and Charalampos Tsourakakis. Space-and time-efficient algorithm for maintaining dense subgraphs on one-pass dynamic streams. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, pages 173–182. ACM, 2015.

[6] Moses Charikar. Greedy approximation algorithms for finding dense components in a graph. In *International Workshop on Approximation Algorithms for Combinatorial Optimization*, pages 84–95. Springer, 2000.

[7] Jie Chen and Yousef Saad. Dense subgraph extraction with application to community detection. *IEEE TKDE*, 24(7):1216–1230, 2012.

[8] Carter Chiu, Justin Zhan, and Felix Zhan. Uncovering suspicious activity from partially paired and incomplete multimodal data. *IEEE Access*, 5:13689–13698, 2017.

[9] Alessandro Epasto, Silvio Lattanzi, and Mauro Sozio. Efficient densest subgraph computation in evolving graphs. In *WWW*, pages 300–310, 2015.

[10] Esther Galbrun, Aristides Gionis, and Nikolaj Tatti. Top-k overlapping densest subgraphs. *DMKD*, 30(5):1134–1165, 2016.

[11] Saptarshi Ghosh, Bimal Viswanath, Farshad Kooti, Naveen Kumar Sharma, Gautam Korlam, Fabricio Benevenuto, Niloy Ganguly, and Krishna Phani Gummadi. Understanding and combating link farming in the twitter social network. In *WWW*, pages 61–70, 2012.

[12] Christos Giatsidis, Dimitrios M Thilikos, and Michalis Vazirgiannis. D-cores: Measuring collaboration of directed graphs based on degeneracy. In *ICDM*, pages 201–210, 2011.

[13] Christos Giatsidis, Dimitrios M Thilikos, and Michalis Vazirgiannis. Evaluating cooperation in communities with the k-core structure. In *ASONAM*, pages 87–93, 2011.

[14] Andrew V Goldberg. *Finding a maximum density subgraph.* University of California Berkeley, CA, 1984.

[15] Zhongshu Gu, Kexin Pei, Qifan Wang, Luo Si, Xiangyu Zhang, and Dongyan Xu. Leaps: Detecting camouflaged attacks with statistical learning guided by program analysis. In *Dependable Systems and Networks (DSN), 2015 45th Annual IEEE/IFIP International Conference on*, pages 57–68. IEEE, 2015.

[16] Bryan Hooi, Neil Shah, Alex Beutel, Stephan Günnemann, Leman Akoglu, Mohit Kumar, Disha Makhija, and Christos Faloutsos. Birdnest: Bayesian inference for ratings-fraud detection. In *SDM*, pages 495–503, 2016.

[17] Bryan Hooi, Hyun Ah Song, Alex Beutel, Neil Shah, Kijung Shin, and Christos Faloutsos. Fraudar: Bounding graph fraud in the face of camouflage. In *KDD*, pages 895–904, 2016.

[18] Xia Hu, Jiliang Tang, and Huan Liu. Online social spammer detection. In *AAAI*, volume 14, pages 59–65, 2014.

[19] Meng Jiang, Alex Beutel, Peng Cui, Bryan Hooi, Shiqiang Yang, and Christos Faloutsos. Spotting suspicious behaviors in multimodal data: A general metric and algorithms. *IEEE TKDE*, 28(8):2187–2200, 2016.

[20] Ravi Kannan and V Vinay. *Analyzing the structure of large graphs.* Rheinische Friedrich-Wilhelms-Universität Bonn Bonn, 1999.

[21] Samir Khuller and Barna Saha. On finding dense subgraphs. In *International Colloquium on Automata, Languages, and Programming*, pages 597–608. Springer, 2009.

[22] Srijan Kumar, William L Hamilton, Jure Leskovec, and Dan Jurafsky. Community interaction and conflict on the web. In *WWW*, pages 933–943, 2018.

[23] Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data, June 2014.

[24] Chao Liu, Xifeng Yan, Hwanjo Yu, Jiawei Han, and Philip S Yu. Mining behavior graphs for "backtrace" of noncrashing bugs. In *SDM*, pages 286–297, 2005.

[25] Shenghua Liu, Bryan Hooi, and Christos Faloutsos. Holoscope: Topology-and-spike aware fraud detection. In *CIKM*, pages 1539–1548, 2017.

[26] Koji Maruhashi, Fan Guo, and Christos Faloutsos. Multiaspectforensics: Pattern mining on large-scale heterogeneous networks with tensor analysis. In *ASONAM*, pages 203–210, 2011.

[27] Shashank Pandit, Duen Horng Chau, Samuel Wang, and Christos Faloutsos. Netprobe: a fast and scalable system for fraud detection in online auction networks. In *WWW*, pages 201–210, 2007.

[28] Bryan Perozzi, Leman Akoglu, Patricia Iglesias Sánchez, and Emmanuel Müller. Focused clustering and outlier detection in large attributed graphs. In *KDD*, pages 1346–1355, 2014.

[29] B Prakash, Ashwin Sridharan, Mukund Seshadri, Sridhar Machiraju, and Christos Faloutsos. Eigenspokes: Surprising patterns and scalable community chipping in large graphs. *Advances in knowledge discovery and data mining*, pages 435–448, 2010.

[30] Neil Shah, Alex Beutel, Brian Gallagher, and Christos Faloutsos. Spotting suspicious link behavior with fbox: An adversarial perspective. In *ICDM*, pages 959–964, 2014.

[31] Hua Shen, Fenglong Ma, Xianchao Zhang, Linlin Zong, Xinyue Liu, and Wenxin Liang. Discovering social spammers from multiple views. *Neurocomputing*, 225:49–57, 2017.

[32] Kijung Shin, Tina Eliassi-Rad, and Christos Faloutsos. Corescope: Graph mining using k-core analysispatterns, anomalies and algorithms. In *ICDM*, pages 469–478, 2016.

[33] Kijung Shin, Bryan Hooi, Jisu Kim, and Christos Faloutsos. D-cube: Dense-block detection in terabyte-scale tensors. In *WSDM*, pages 681–689, 2017.

[34] Kijung Shin, Bryan Hooi, Jisu Kim, and Christos Faloutsos. Densealert: Incremental dense-subtensor detection in tensor streams. In *KDD*, 2017.

[35] Charalampos Tsourakakis. The k-clique densest subgraph problem. In *WWW*, pages 1122–1132, 2015.

[36] Charalampos Tsourakakis, Francesco Bonchi, Aristides Gionis, Francesco Gullo, and Maria Tsiarli. Denser than the densest subgraph: extracting optimal quasi-cliques with quality guarantees. In *KDD*, pages 104–112, 2013.

[37] Sankar Virdhagriswaran and Gordon Dakin. Camouflaged fraud detection in domains with complex relationships. In *KDD*, pages 941–947, 2006.

[38] Soroush Vosoughi, MostafaNeo Mohsenvand, and Deb Roy. Rumor gauge: predicting the veracity of rumors on twitter. *ACM TKDD*, 11(4):50, 2017.

[39] Zhaonian Zou, Jianzhong Li, Hong Gao, and Shuo Zhang. Mining frequent subgraph patterns from uncertain graph data. *IEEE TKDE*, 22(9):1203–1218, 2010.