# Temporal Egonet Transitions

Lucas Parzianello
University of Notre Dame
Notre Dame, IN, USA
lparzianello@nd.edu

Sophia Abraham
University of Notre Dame
Notre Dame, IN, USA
sabraha2@nd.edu

Eric Tsai
University of Notre Dame
Notre Dame, IN, USA
ctsai@nd.edu

Daniel Gonzalez
University of Notre Dame
Notre Dame, IN, USA
dgonza26@nd.edu

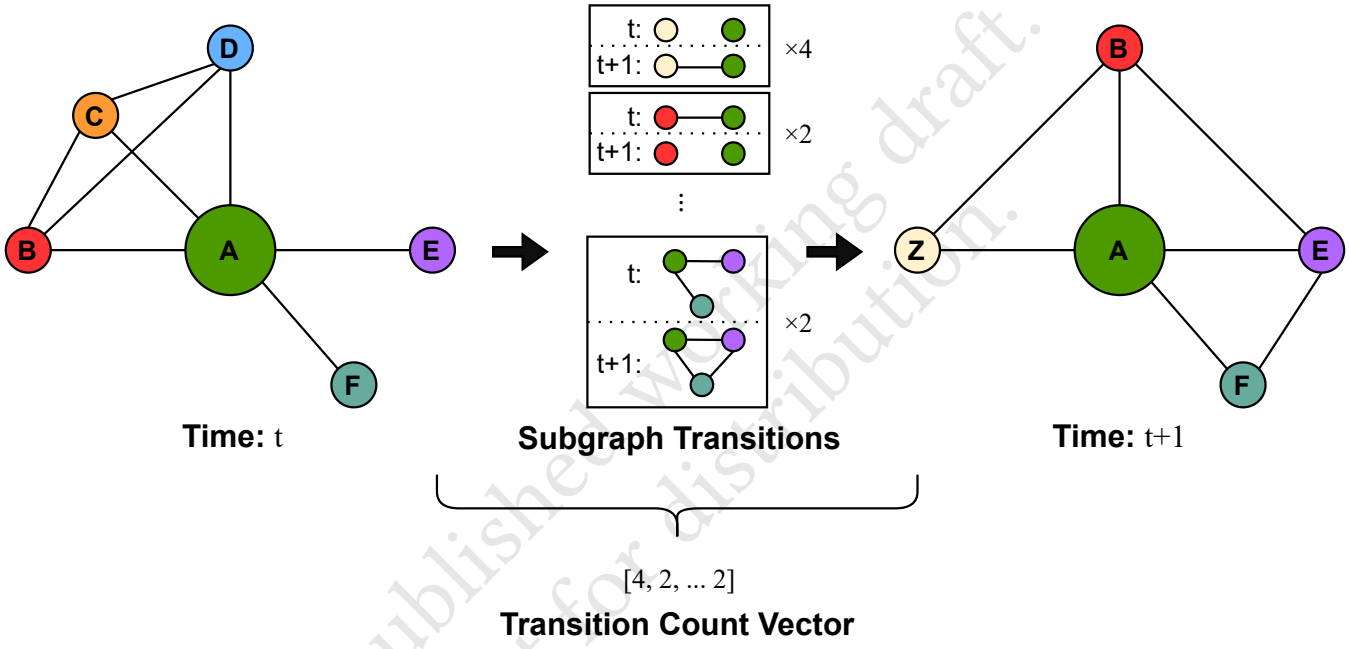


**Figure 1**

## ABSTRACT

How can we succinctly capture peoples' behavioral patterns over time? We introduce Temporal Egonet Transitions (TET) as a step in the direction of answering that question.

(this abstract is a work-in-progress)

## KEYWORDS

temporal network, node embedding, behavior modeling


*Woodstock '21, June 03–05, 2021, Woodstock, NY*

ACM ISBN 978-1-4503-XXXX-X/21/06…$15.00
https://doi.org/10.1145/1122445.1122456



**ACM Reference Format:**
Lucas Parzianello, Sophia Abraham, Eric Tsai, and Daniel Gonzalez. 2021. Temporal Egonet Transitions. In *Woodstock '21: ACM Symposium on Neural Gaze Detection, June 03–05, 2021, Woodstock, NY*. ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/1122445.1122456

## 1 INTRODUCTION

The problem of anomaly detection in graphs has a multitude of real-world applications in many fields. An anomaly may be a rare item, event, or entity that does not fit with the more 'typical' trends in a set of data - i.e. an outlier. Examples include bank fraud in a set of financial transactions, typos in a text, intrusions in networks, 'bot' users in social media, or a sudden drop in sales caused by a global pandemic. In this paper, we propose Temporal Egonet Transitions (TET): a comprehensive way of summarizing the topological behavioral patterns of nodes in a network over time.

We represent interactions between users over time as a sequence of graphs, representing the evolution of interactions between a set

of nodes over time. In this framework, we are interested in modeling the behavior individual nodes exhibit in order to distinguish different patterns of interaction. To that end, TET creates a vector representation of each node using the changes in its neighborhood – or *egonet* – over time. These vector representations can then be used for traditional tasks, such as clustering and classification, depending on the quality of the data and the application domain.

TET is flexible enough to lend itself to a variety of standard tasks and application domains because it does not depend on labeled data or any kind of iterative learning procedure. Nodes' vector representations are computed purely based on how their egonets change over time. Once in the embedding space, the data can be analyzed from various perspectives. For example, given labeled data, the latent representations can be used for classification tasks such as bot-detection on social networks. However, in the absence of ground truth regarding nodes, the latent representations can still be analyzed to find *communities*: clusters of points in the embedding space signalling nodes with similar temporal behavior.

Despite is flexibility, TET makes some important assumptions about the data. First, the data must be graphical, meaning it can be represented as interactions (edges) between objects (nodes). Further, the data should be represented as a sequence of graphs $G_1, G_2, \ldots G_T$, where the events described by graph $G_t$ precede those described by $G_{t+1}$. This means that datasets representing temporal interactions as timestamped streams of edges need to be *binned* appropriately; if too much fidelity is lost in the binning process, then the dataset will not be amenable to analysis using TET. Finally, the graphical representation of the data should lend itself to node-centric analysis. While some dataset and problem domains may be concerned with clustering or classifying edges – as is the case in, for example, detecting fraudulent transactions on financial networks – those would be beyond the scope of TET and this paper without further manipulation or transformation of the data.

## 2 RELATED WORK

Diverse techniques exists for anomaly detection in literature. Approaches include density-based techniques like k-nearest neighbor [12], local outlier factor [3], isolation forests [16] and other variations [27]. For higher dimensional data - subspace [13], correlation [14], single class support vector machines [24] and tensor based methods [7] have been proposed. Neural network approaches include replicators [10], Bayesian Networks [24], hidden markov models [24]. Additionally, cluster analysis based [4], fuzzy logic and ensemble techniques with feature bagging [21] and score normalization [26] have been applied.

Although these methodologies have minimal systematic advantages across data sets and parameters, many are applied in the realm of static graphs.

To narrow the scope in relation to our proposed method regarding anomaly detection algorithms for dynamic graph networks a deeper exploration will be applied to network embedding and streaming anomaly approaches.

### 2.1 Streaming Anomaly Detection

Streaming anomaly detection utilizes a series of graphs or edges over time and can be utilized for a variety of anomalies.

*2.1.1 Streaming Graphs.* Streaming graphs can be applied to anomalous node detection like dynamic tensor analysis (DTA) [29] which approximates an adjacency matrix for a graph at time $t$ with incremental matrix factorization and utilizes a high reconstruction error to. Anomalous subgraph detection is illustrated in methods like [28] which uses *k-core* which is the maximal subgraph in which all vertices have degree at least $k$ and use patterns related to *k-core* to find anomalous subgraphs. Streaming graphs can also be utilized for anomalous event detection such as changes in first and second derivatives in PageRank [32].

*2.1.2 Streaming Edges.* Streaming anomaly detection can also be applied to streaming edges. Anomalous nodes can be detected using methods like [33] which uses an incremental eigenvector update algorithm based in von Mises interations and discover hotspots of local changes in dynamics streams. SedanSpot [6] utilizes streaming edges to detect anomalous edges by exploiting edges that connect part of the graph that are sparsely connected and finding where bursts of activity occur.

### 2.2 Network Embedding

Network embedding approaches consist of learning low-dimensional feature representation of the nodes or links within a network. Many advances [5, 15, 31, 35] in the network embedding space present new ways to learn representations for networks which can be used to detect anomalies. Feature learning strategies for extracting network embedding have been widely used in language models such as Skip-gram [20] in which the the defining neighborhood attributes for words in a sentence are preserved. In a similar manner, many popular methodologies such as DeepWalk [23] and Node2vec [8] use sequences of vertices from the graphs and learn the representation of these vertices by maximizing the preservation of the structure inherent to a neighborhood of vertices in the network. A large portion of work has also been dedicated to representation learning in other manners for graph network architectures [30, 31, 36] and compact representations of the graph have been used to find anomalies within the network [18].

These methods have also been specifically applied for anomaly detection in dynamic temporal graphs. Specifically, NetWalk [34] learns network representations which are dynamically updated as the network evolves. This is done by by learning the latent network representations by extracted sequences of vertices from the graph, otherwise termed as "walks" from the initial network and then extract deep representations by minimizing the pairwise distance of the vertices when encoding the vector representations and adding global regularization by using a deep autoencoder reconstruction error. NetWalk is updated over dynamic changes with reservoir sampling strategy and a dynamic clustering model is used to find anomalies.

In [11], dynamic graph evolution is modeled with subgraph to subgraph transitions (SST) by fitting linear SVM models to SST count vectors. This can be used for link prediction of static, temporal, directed and undirected graphs while providing interpretable

results. The idea of counting SSTs inspires our proposed methodology, however, instead of utilizing an edge based approach as this paper did, we propose a neighborhood based approach for detecting anomalies based on clusters which is further expanded upon in Section 4.

# 3 DATA SOURCES

We are focused on modeling behavior over time from graph-structured data, so we need datasets with the following attributes:

- nodes with persistent IDs
- interactions between nodes, either directed or undirected
- timestamps for the interactions

## 3.1 Explanation of Datasets

Although we initially wanted to tackle node classification problems using TET, we ran into problems finding datasets which had both timestamped edges and ground-truth labels for the nodes. Most of the datasets with ground-truth we found had labels for the edges instead. Therefore, we are considering unsupervised tasks like clustering for testing the quality of our node embeddings.

Below, we describe some of the datasets we have found.

*3.1.1 For Reputation: Snap-Bitcoin OTC and Snap-Bitcoin Alpha.* These datasets record who-rate-whom-when bitcoin transaction. The data format contains:

- SOURCE: a node id of source
- TARGET: a node id of target
- RATING: a source node rates target node from -10 to +10
- TIME: when this transaction happen

*3.1.2 For Suspicion: Kaggle-Paysim synthetic dataset of mobile money transactions.* This dataset contains some key features that we can use:

- NameOrg: customer that started the transaction
- NameDest: customer that receive the transaction
- Amount: the total amount of transaction
- Step: when this transaction happen (each digit represents a hour)

*3.1.3 Enron Email Dataset.* The nodes of this dataset represent email addresses and directed edges depict sent/received relations. Enron email network dataset contains a total of 80, 884 nodes and 700 timestamp in total.

- Source IP: a source node id of an user
- Destination IP: a target node id of an user
- timestamp: timestamp in dates

*3.1.4 CollegeMsg temporal network.* This is a dataset containing private messages at an online social network at the University of California, Irvine. Here an edge (u, v, t) means that during time t, an user u sent a message to an user v.

- SRC: a source node id of an user
- TGT: a target node id of an user
- UNIXTS: timestamp in seconds.

*3.1.5 Email-Eu-core temporal network.* This dataset contains 986 nodes and 332,334 temporal edges, with the timestamp of 803 in total.

- SRC: a source node id of an user
- TGT: a target node id of an user
- TS: timestamp in seconds(started from 0).

## 3.2 Data Preprocessing

Our model makes the assumption that the data is structured as a sequence of graphs $G_1, G_2, \ldots G_T$ on a shared node set $V$,so that nodes' egonet subgraph transitions are well-defined and sufficiently nontrivial. However, the datasets described in Section 3.1 represent temporal graphs as a continuous stream of timestamped interactions between nodes.

The first step in processing these datasets is to discretize the time dimension using *binning*. If the first timestamp in a given dataset is $t_0$, the last timestamp is $t_k$, and the range is given by $\Delta = t_k - t_0$, we divide the range $[t_0, t_k]$ into $T$ equally-sized chunks $[t_0, t_0 + \Delta), [t_0 + \Delta, t_0 + 2\Delta), \ldots [t_k - \Delta, t_k]$ and distribute each edge into the appropriate bin according to its timestamp. Each bin then becomes one of the graphs in the sequence $G_1, G_2, \ldots G_T$.

After the temporal discretization is done, the final step in the preprocessing is to construct a shared node set for all of the graphs in the sequence. This is simply done by asserting that each graph's vertex set is the union of all of the vertex sets of all the graphs in the sequence.

# 4 MODEL

The basic idea behind TET is that changes in a node's egonet can be summarized by counting *subgraph transitions*. A subgraph transition for a node $v$ at a given time transition $(t, t+1)$ is defined as a pair of graphs $(H_t, H_{t+1})$ on a shared node set $V(H_t, H_{t+1})$ such that:

- $H_t$ is a subgraph of the adjacency neighborhood of $v$ at time $t$ if any missing nodes from the neighborhood at time $t+1$ are included
- $H_{t+1}$ is a subgraph of the adjacency neighborhood of $v$ at time $t+1$ if any missing nodes from the neighborhood at time $t$ are included
- $H_t$ and $H_{t+1}$ are not isomorphic.

In the interest of computational tractability, we only consider subgraph transitions up to a certain number of vertices $N$ (e.g., 3 or 4). By enumerating and canonically ordering all of the subgraph transitions for subgraphs on $N$ or less vertices, we can construct a vector of *subgraph transition counts* for the node $v$ during the time hop from $t$ to $t+1$. This subgraph transition count vector summarizes the change in $v$'s behavior from the previous time step to the next time step. Performing this for each time transition results in a sequence of count vectors $\vec{v}_1, \vec{v}_2, \ldots \vec{v}_{T-1} \in \mathbb{R}^d$, where $d$ is the number of subgraph transitions enumerated ($d$ may be *very* large, even for small $N$). We can then condense these counts into a single vector representation for $v$ by applying some element-wise *aggregation function* (e.g., mean, sum, min, max).

At the time of this writing, the code for this model is still being implemented, but we will briefly describe some of the techniques and packages that will be leveraged.

In order to most-efficiently enumerate and count subgraph transitions, it will be useful to conceptualize a *subgraph labeller* and a

*transition counter*. The subgraph labeller will be tasked with enumerating all graphs, up to isomorphism, on $N$ or less vertices, and assigning them a canonical (i.e., isomorphism-invariant) label. That way, when a subgraph is mined from a node's egonet, we have an unambiguous representation of the subgraph free from its initial context. We plan on using a variation of the *color refinement* algorithm [2] to accomplish this. The transition counter will then be tasked with recognizing and counting pairs of subgraphs and generating the transition count vectors for each node.

For dealing with graphical representations of data and most manipulations, we make good use of the NetworkX package for Python [9], which provides a fast intuitive framework for processing graphs. In order to quickly enumerate graphs and efficiently compute graph isomorphisms and automorphisms, we take advantage of Nauty [19] and the Python bindings provided by PyNauty.

# 5 EVALUATION

## 5.1 Unsupervised Evaluation

Evaluating unsupervised algorithms is challenging. Most of the traditional metrics applied in supervised learning do not make sense when the data does not have labels. Thus, we need alternative metrics to discern model qualities. When looking into clustering results, we can take some factors that do not rely on labels into account, as described in [22], such as: the clustering tendency (as opposed to a random structure), the number of clusters, use internal information to assess the quality of the clustering, and comparing clustering solutions among themselves.

Measures of cohesion and separation of clusters are used to assess how close the nodes of one cluster are to each other, and how far they are to nodes in other clusters respectively. Generally, a clustering is considered good when it has a high separation between clusters and a high cohesion within clusters.

These metrics give us a good sense of how our clustering performs, but there is still one question yet to be answered: how do we answer the meaning of these clusters? Do they correspond to the pattern that we would like to detect, separating "anomalous" from "normal" behavior? In order to answer that, we need to obtain a dataset with labels and perform a supervised evaluation.

## 5.2 Supervised Evaluation

In the case that we find a properly labeled dataset to evaluate TET, we can use more commonly found metrics to compare our results in an anomaly-detection problem. We probably will not use accuracy of classification directly, however. That is because anomaly datasets in general are heavily skewed towards the normal behavior, thus, any classifier that labeled them all as normal would achieve an accuracy close to 100%.

Instead, we would use metrics commonly found in genuine-impostor kinds of problem. These are less affected by this imbalance. The ROC curve, for example, is one of measuring such classifiers, and the area under the curve (or AUC) is a way of transforming each ROC curve into a single number, useful for comparison. The higher the area under the ROC curve, the better the classifier.

## 5.3 Baselines

In order to assess the quality of our method, we will compare against competing methods. Potential dynamic anomaly detection methods to test against include:

- **Count-Min Sketch Based Outlier Detection[25]**: Utilizes Count-Min sketch to approximate properties to generate outlier detection model based on global and local structure of a stream.
- **NetWalk[34]**: Finds anomalies with a dynamic clustering model.
- **Outlier Detection in Graph Streams[1]**: Uses a structural connectivity model to define outliers in graph streams.

Additionally, we could test against different network embedding methods such as:

- **Spectral Clustering[17]**: No assumption on the shapes of the clusters and can thus model complex scenarios.
- **DeepWalk[23]**: Generalizes on language modeling from sequences of words to graphs by using local information from sequences of vertices in the graph and learning representations by treating them equivalently to sentences.
- **Node2Vec[8]**
- **Structural Deep Network Embedding (SDNE)[31]**: Semi-supervised deep model with layers of non-linear functions which uses first and second order proximity to capture the global network structure.



# REFERENCES


[1] Charu C Aggarwal, Yuchen Zhao, and S Yu Philip. 2011. Outlier detection in graph streams. In *2011 IEEE 27th International Conference on Data Engineering*. IEEE, 399–409.

[2] V. Arvind, Johannes Köbler, Gaurav Rattan, and Oleg Verbitsky. 2016. Graph Isomorphism, Color Refinement, and Compactness. *computational complexity* 26, 3 (2016), 627–685. https://doi.org/10.1007/s00037-016-0147-6

[3] Markus M Breunig, Hans-Peter Kriegel, Raymond T Ng, and Jörg Sander. 2000. LOF: identifying density-based local outliers. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*. 93–104.

[4] Ricardo JGB Campello, Davoud Moulavi, Arthur Zimek, and Jörg Sander. 2015. Hierarchical density estimates for data clustering, visualization, and outlier detection. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 10, 1 (2015), 1–51.

[5] Shiyu Chang, Wei Han, Jiliang Tang, Guo-Jun Qi, Charu C Aggarwal, and Thomas S Huang. 2015. Heterogeneous network embedding via deep architectures. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*. 119–128.

[6] Dhivya Eswaran and Christos Faloutsos. 2018. Sedanspot: Detecting anomalies in edge streams. In *2018 IEEE International Conference on Data Mining (ICDM)*. IEEE, 953–958.

[7] Hadi Fanaee-T and Joao Gama. 2016. Tensor-based anomaly detection: An interdisciplinary survey. *Knowledge-Based Systems* 98 (2016), 130–147.

[8] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. 855–864.

[9] A A Hagberg, D A Schult, and P J Swart. 2008. Exploring network structure, dynamics, and function using NetworkX. *7th Python in Science Conference (SciPy 2008)* SciPy (2008), 11–15.

[10] Simon Hawkins, Hongxing He, Graham Williams, and Rohan Baxter. 2002. Outlier detection using replicator neural networks. In *International Conference on Data Warehousing and Knowledge Discovery*. Springer, 170–180.

[11] Justus Isaiah Hibshman, Daniel Gonzalez, Satyaki Sikdar, and Tim Weninger. 2021. Joint Subgraph-to-Subgraph Transitions: Generalizing Triadic Closure for Powerful and Interpretable Graph Modeling. *Proceedings of the 14th ACM International Conference on Web Search and Data Mining* (2021). https://doi.org/10.1145/3437963.3441817

[12] Edwin M Knorr, Raymond T Ng, and Vladimir Tucakov. 2000. Distance-based outliers: algorithms and applications. *The VLDB Journal* 8, 3 (2000), 237–253.

[13] Hans-Peter Kriegel, Peer Kröger, Erich Schubert, and Arthur Zimek. 2009. Outlier detection in axis-parallel subspaces of high dimensional data. In *Pacific-asia conference on knowledge discovery and data mining*. Springer, 831–838.
[14] Hans-Peter Kriegel, Peer Kröger, Erich Schubert, and Arthur Zimek. 2012. Outlier detection in arbitrarily oriented subspaces. In *2012 IEEE 12th international conference on data mining*. IEEE, 379–388.
[15] Omer Levy and Yoav Goldberg. 2014. Neural word embedding as implicit matrix factorization. *Advances in neural information processing systems* 27 (2014), 2177–2185.
[16] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. 2012. Isolation-based anomaly detection. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 6, 1 (2012), 1–39.
[17] Jialu Liu, Jiawei Han, C Aggarwal, and C Reddy. 2013. Spectral Clustering.
[18] Emaad Manzoor, Sadegh M Milajerdi, and Leman Akoglu. 2016. Fast memory-efficient anomaly detection in streaming heterogeneous graphs. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 1035–1044.
[19] Brendan D. McKay and Adolfo Piperno. 2014. Practical graph isomorphism, II. *Journal of Symbolic Computation* 60 (2014), 94–112. https://doi.org/10.1016/j.jsc.2013.09.003
[20] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Distributed representations of words and phrases and their compositionality. *arXiv preprint arXiv:1310.4546* (2013).
[21] Hoang Vu Nguyen, Hock Hee Ang, and Vivekanand Gopalkrishnan. 2010. Mining outliers with ensemble of heterogeneous detectors on random subspaces. In *International Conference on Database Systems for Advanced Applications*. Springer, 368–383.
[22] Julio Omar Palacio-Niño and Fernando Berzal. 2019. Evaluation Metrics for Unsupervised Learning Algorithms. *arXiv* (2019).
[23] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. 701–710.
[24] John C Platt, John Shawe-Taylor, Alex J Smola, Robert C Williamson, et al. 1999. Estimating the support of a high-dimensional distribution. *Technical Report MSR-T R-99–87, Microsoft Research (MSR)* (1999).
[25] Stephen Ranshous, Steve Harenberg, Kshitij Sharma, and Nagiza F Samatova. 2016. A scalable approach for outlier detection in edge streams using sketch-based approximations. In *Proceedings of the 2016 SIAM International Conference on Data Mining*. SIAM, 189–197.
[26] Erich Schubert, Remigius Wojdanowski, Arthur Zimek, and Hans-Peter Kriegel. 2012. On evaluation of outlier rankings and outlier scores. In *Proceedings of the 2012 SIAM International Conference on Data Mining*. SIAM, 1047–1058.
[27] Erich Schubert, Arthur Zimek, and Hans-Peter Kriegel. 2014. Local outlier detection reconsidered: a generalized view on locality with applications to spatial, video, and network outlier detection. *Data mining and knowledge discovery* 28, 1 (2014), 190–237.
[28] Kijung Shin, Tina Eliassi-Rad, and Christos Faloutsos. 2018. Patterns and anomalies in k-cores of real-world graphs with applications. *Knowledge and Information Systems* 54, 3 (2018), 677–710.
[29] Jimeng Sun, Dacheng Tao, and Christos Faloutsos. 2006. Beyond streams and graphs: dynamic tensor analysis. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. 374–383.
[30] Fei Tian, Bin Gao, Qing Cui, Enhong Chen, and Tie-Yan Liu. 2014. Learning deep representations for graph clustering. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 28.
[31] Daixin Wang, Peng Cui, and Wenwu Zhu. 2016. Structural deep network embedding. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. 1225–1234.
[32] Minji Yoon, Bryan Hooi, Kijung Shin, and Christos Faloutsos. 2019. Fast and accurate anomaly detection in dynamic graphs with a two-pronged approach. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 647–657.
[33] Weiren Yu, Charu C Aggarwal, Shuai Ma, and Haixun Wang. 2013. On anomalous hotspot discovery in graph streams. In *2013 IEEE 13th International Conference on Data Mining*. IEEE, 1271–1276.
[34] Wenchao Yu, Wei Cheng, Charu C Aggarwal, Kai Zhang, Haifeng Chen, and Wei Wang. 2018. Netwalk: A flexible deep embedding approach for anomaly detection in dynamic networks. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2672–2681.
[35] Wenchao Yu, Guangxiang Zeng, Ping Luo, Fuzhen Zhuang, Qing He, and Zhongzhi Shi. 2013. Embedding with autoencoder regularization. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 208–223.
[36] Lekui Zhou, Yang Yang, Xiang Ren, Fei Wu, and Yueting Zhuang. 2018. Dynamic network embedding by modeling triadic closure process. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 32.