N2N: Network Derivative Mining

Jian Kang*
University of Illinois at Urbana-Champaign
jiank2@illinois.edu

Hanghang Tong*
University of Illinois at Urbana-Champaign
htong@illinois.edu

ABSTRACT

Network mining plays a pivotal role in many high-impact application domains, including information retrieval, healthcare, social network analysis, security and recommender systems. State-of-the-art offers a wealth of sophisticated network mining algorithms, many of which have been widely adopted in real-world with superior empirical performance. Nonetheless, they often lack effective and efficient ways to characterize how the results of a given mining task relate to the underlying network structure.

In this paper, we introduce network derivative mining problem. Given the input network and a specific mining algorithm, network derivative mining finds a derivative network whose edges measure the influence of the corresponding edges of the input network on the mining results. We envision that network derivative mining could be beneficial in a variety of scenarios, ranging from explainable network mining, adversarial network mining, sensitivity analysis on network structure, active learning, learning with side information to counterfactual learning on networks. We propose a generic framework for network derivative mining from the optimization perspective and provide various instantiations for three classic network mining tasks, including ranking, clustering, and matrix completion. For each mining task, we develop effective algorithm for constructing the derivative network based on influence function analysis, with numerous optimizations to ensure a linear complexity in both time and space. Extensive experimental evaluation on real-world datasets demonstrates the efficacy of the proposed framework and algorithms.

CCS CONCEPTS

• **Information systems** → **Data mining**; **Link and co-citation analysis**; **Clustering**; **Collaborative filtering**; • **Discrete mathematics** → **Graph algorithms**; • **Theory of computation** → *Adversarial learning*;

KEYWORDS

Network mining; network intervention; framework

ACM Reference Format:
Jian Kang and Hanghang Tong. 2019. N2N: Network Derivative Mining. In *The 28th ACM International Conference on Information and Knowledge Management (CIKM '19), November 3–7, 2019, Beijing, China.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3357384.3357910

*This work was partly done while the authors were at Arizona State University.

CIKM '19, November 3–7, 2019, Beijing, China

ACM ISBN 978-1-4503-6976-3/19/11…$15.00
https://doi.org/10.1145/3357384.3357910

1 INTRODUCTION

Network mining plays a pivotal role in many important real-world applications, including information retrieval [23, 36], healthcare [39], social network analysis [41], security [45], and recommender systems [15]. Throughout the years, researchers have developed many network mining algorithms for various mining tasks. To name a few, HITS [23] is a well-known and widely used ranking algorithm to measure node importance by considering the network structure; spectral clustering [34] is a popular technique for community detection and image segmentation; and matrix factorization-based completion [26] on a bipartite network is a key enabling technology for modern recommender systems.

State-of-the-art network mining algorithms have been widely adopted in various real-world applications, which often deliver a strong empirical performance in finding interesting patterns, e.g., which webpages are the most important, who are grouped into the same online community and which movies best suit users' tastes, etc. Despite the tremendous progress, it remains opaque on *how the results of a given mining algorithm relate to the underlying network structure.* Consequently, it is often hard to answer questions like *how* the ranking results for webpages might be manipulated by malicious link farms; *why* two seemingly different users are grouped into the same online community; *how* sensitive the recommendation results are due to the random noisy or fake ratings; *what* would have happened to the epidemic dynamics *if* we had distributed vaccines in a different way, etc.

To tackle this issue, we propose a paradigm shift of network mining and introduce *network derivative mining (N2N)* problem. To be specific, given an input network and a mining algorithm, it aims to find a derivative network, each of whose edges provides a quantitative measurement of the influence of the corresponding edge of the input network on the given mining algorithm. In detail, we define the influence of edges as the rate of change of a function over the mining results induced by the given mining algorithm. We envision that network derivative mining will benefit a variety of aspects. First, it is directly applicable to adversarial network mining, where users can identify potential edges which, if attacked, will drastically affect network mining results. Second, it will render the crucial explainability of the network mining model, by identifying the most responsible/relevant edges for the mining results. It can further help answer questions like why a node belongs or does not belong to a certain cluster. Third, the derivative network can be used as a quantitative reference for sensitivity analysis on network structure. Fourth, the network derivative mining has great potential in active learning where edges with the high influence act as the most valuable data points to query the oracle. Fifth, it will offer an effective way to encode side information to boost some network

mining tasks, e.g., to learn an optimal network based on user feedback [31]. Finally, it allows the end users to quickly examine how the mining results would differ should the underlying network have changed, and thus it naturally fits for counterfactual learning on networks.

Besides the problem definition, the main contributions of this paper are summarized as follows.

- **Algorithmic Framework**. We formulate network derivative problem as an optimization problem, and propose a generic algorithmic framework. Its key idea is to measure the influence as the rate of change of a scalar valued function over the given network mining task.
- **Instantiations and Computation**. We instantiate the proposed framework by three classic network mining tasks, including ranking, clustering and completion. For each task, we propose effective and efficient algorithm to construct the corresponding derivative network with a linear complexity in both time and space.
- **Empirical Evaluations**. We perform extensive experiments on diverse, real-world datasets. The experimental results demonstrate that our proposed method (a) is effective in adversarial network mining for different instantiations and (b) scales linearly with respect to the number of nodes and edges in the network.

The rest of the paper is organized as follows. Section 2 formally defines the network derivative mining problem. Section 3 proposes a generic framework for network derivative mining problem. We provide three examples of network derivative mining in Section 4. Experimental evaluations are shown in Section 5. After reviewing related work in Section 6, we conclude the paper in Section 7.

2 PROBLEM DEFINITION

In this section, we first present a table of symbols used throughout the paper (Table 1). Then we review the general procedure of classic network mining tasks. Finally, we formally define the network derivative mining problem.

Table 1: Table of Symbols

Symbols	Definitions and Descriptions
$\mathbf{G} = (\mathcal{V}, \mathcal{E})$	the input network
(i, j)	edge from node i to node j
$\mathbf{A}$	a matrix
$\mathbf{A}(i, j)$	the element at the i^{th} row and the j^{th} column
$\mathbf{A}(i, :)$	the i^{th} row of matrix $\mathbf{A}$
$\mathbf{A}(:, j)$	the j^{th} column of matrix $\mathbf{A}$
$\mathbf{A}'$	transpose of the matrix $\mathbf{A}$
$\mathbf{A}^{-1}$	inverse of the matrix $\mathbf{A}$
$\mathbf{u}$	a vector
$\mathbf{u}(i)$	the i^{th} element of vector $\mathbf{u}$
$\mathbb{I}(i, j)$	the influence of edge (i, j)
$L(\cdot)$	the loss function for a mining task
$\mathcal{Y}^*$	the optimal model output
θ	a set of parameters
$f(\mathcal{Y}^*)$	a scalar function over the mining results
n	number of nodes
m	number of edges

In this paper, we denote matrices with bold upper-case letters (e.g., $\mathbf{A}$), vectors with bold lower-case letters (e.g., $\mathbf{x}$), sets with calligraphic fonts (e.g., $\mathcal{S}$), and scalars with lower-case letters (e.g., c). For matrix indexing conventions, we use the rules similar to Matlab as follows. We use $\mathbf{A}(i, j)$ to denote the entry of matrix $\mathbf{A}$ at the i^{th} row and the j^{th} column, $\mathbf{A}(i, :)$ to denote the i^{th} row of matrix $\mathbf{A}$, and $\mathbf{A}(:, j)$ to denote the j^{th} column of matrix $\mathbf{A}$. We use prime to denote the transpose of matrix (i.e., $\mathbf{A}'$ is the transpose of matrix $\mathbf{A}$).

Generally speaking, given a network $\mathbf{A}$ with n nodes and m edges, a network mining algorithm aims to learn mining results $\mathcal{Y}^*$ by optimizing a loss function $L(\mathbf{A}, \mathcal{Y}, \theta)$, where $\mathcal{Y}^* = \operatorname{argmin}_{\mathcal{Y}} L(\mathbf{A}, \mathcal{Y}, \theta)$ is the optimal model output, and θ is a set of additional parameters that corresponds to a specific mining task. Let us illustrate this using three classic examples. Table 2 presents a summary.

The first example is HITS [23], which is a widely-used ranking algorithm that measures the importance of nodes with hub scores $\mathbf{u}$ and authority scores $\mathbf{v}$ for network structure analysis. It solves the linear system $\mathbf{u} = \mathbf{A}\mathbf{v},\ \mathbf{v} = \mathbf{A}'\mathbf{u}$, which can be naturally formulated as the following optimization problem,

$$\min_{\mathbf{u},\mathbf{v}} \quad ||\mathbf{A} - \mathbf{u}\mathbf{v}'||_F^2 \tag{1}$$

The second example is spectral clustering, which aims to find a matrix $\mathbf{U}$ with r orthonormal column vectors by the following optimization problem,

$$\begin{aligned} \min_{\mathbf{U}} \quad & \text{Tr}(\mathbf{U}'\mathbf{L}\mathbf{U}) \\ \text{s.t.} \quad & \mathbf{U}'\mathbf{U} = \mathbf{I} \end{aligned} \tag{2}$$

where $\mathbf{L}$ is the Laplacian matrix of adjacency matrix $\mathbf{A}$, and r is the number of clusters. It is well-known that $\mathbf{U}'\mathbf{L}\mathbf{U}$ is essentially the diagonal matrix of the r smallest eigenvalues of $\mathbf{L}$, and columns in $\mathbf{U}$ are the associated eigenvectors.

The third example is matrix-factorization based completion, where we are given a bipartite network $\mathbf{A}$ with n_1 users, n_2 items and m observations. We denote $\mathbf{A}(i, j)$ as the rating of the j^{th} item made by the i^{th} user. With the low-rank assumption, matrix factorization-based completion finds two low-rank matrices with r latent factors, namely $\mathbf{U}$ and $\mathbf{V}$, such that

$$\min_{\mathbf{U},\mathbf{V}} \quad ||\text{proj}_{\Omega}(\mathbf{A} - \mathbf{U}\mathbf{V}')||_F^2 + \lambda_u ||\mathbf{U}||_F^2 + \lambda_v ||\mathbf{V}||_F^2 \tag{3}$$

where $\Omega = \{(i, j) : \mathbf{A}(i, j) \text{ is observed}\}$, λ_u and λ_v are two hyperparameters for regularization. Each row of the factorized $n_1 \times r$ matrix $\mathbf{U}$ and $n_2 \times r$ matrix $\mathbf{V}$ represent a latent vector for the corresponding user and item, respectively.

Though these network mining algorithms have achieved a remarkable empirical performance in finding various patterns for the corresponding network mining tasks, they often lack effective and efficient ways to characterize how such results relate to the input network's structure. Following an overarching principle laid in [20, 24], we adopt influence functions to quantify the impact of network structure (e.g., edges) when perturbed. Based on that, we propose to go the extra mile to further construct a *derivative network*, where each edge measures the influence of the corresponding edge of the input network on the given mining algorithm. Formally, we define the network derivative mining problem as follows.

Problem 1. *Network Derivative Mining Problem (N2N).*

Input: (1) an input network with adjacency matrix $\mathbf{A}$; and (2) a network mining algorithm represented as $L(A, \mathcal{Y}, \theta)$, where $L(\cdot)$ is the loss function, $\mathcal{Y}^* = \text{argmin}_{\mathcal{Y}} L(A, \mathcal{Y}, \theta)$ is the model output, and θ contains all the additional parameters.

Output: a derivative network $\mathbf{B}$, which has the same node set as the input network $\mathbf{A}$, where $\mathbf{B}(i, j)$ measures the influence of edge $\mathbf{A}(i, j)$ on $\mathcal{Y}^*$, and $\mathbf{B}(i, j) = 0$ if $\mathbf{A}(i, j)$ does not exist.

Remarks. In this paper, we focus on the derivatives of existing edges (i.e., $\mathbf{A}(i, j) = 1$). Nonetheless, the proposed technique for computing the influence $\mathbf{B}(i, j)$ naturally applies to non-existing edges (i.e., $\mathbf{A}(i, j) = 0$).

3 N2N ALGORITHMIC FRAMEWORK

In this section, we present a generic algorithmic framework for network derivative mining problem. We first define the influence of edges, and then formulate the network derivative mining problem from the optimization perspective, followed by a generic algorithmic framework to solve it. Formally, we define the influence as the rate of change in $f(\mathcal{Y}^*)$ for different edges.

Definition 1. *(Edge Influence). Let* $\mathbf{B}$ *be the derivative network,* $\mathcal{Y}^*$ *be the optimal result of a network mining task, and* $f(\cdot)$ *be a scalar function defined over the mining result* $\mathcal{Y}^*$*, the influence of an edge* (i, j) *is defined as the derivative of* $f(\mathcal{Y}^*)$ *with respect to the corresponding edge in the input network* $\mathbf{A}$*, i.e.,* $\mathbb{I}(i, j) = \mathbf{B}(i, j) = \frac{\mathrm{d}f(\mathcal{Y}^*)}{\mathrm{d}\mathbf{A}(i,j)}$.

Based on Definition. 1, the network derivative mining problem can be naturally formulated as the following optimization problem,

$$\begin{aligned} \mathbf{B} &= \frac{\mathrm{d}f(\mathcal{Y}^*)}{\mathrm{d}\mathbf{A}} \\ \text{s.t. } \mathcal{Y}^* &\in \underset{\mathcal{Y}}{\text{argmin}}\, L(\mathbf{A}, \mathcal{Y}, \theta) \end{aligned} \tag{4}$$

where $L(\cdot)$ is the loss function of a network mining task with θ being the additional parameters from Table 2. To be specific, we have that

$$\mathbf{B} = \frac{\mathrm{d}f(\mathcal{Y}^*)}{\mathrm{d}\mathbf{A}} = \begin{cases} \frac{\partial f(\mathcal{Y}^*)}{\partial \mathbf{A}} + (\frac{\partial f(\mathcal{Y}^*)}{\partial \mathbf{A}})' - \text{diag}(\frac{\partial f(\mathcal{Y}^*)}{\partial \mathbf{A}}), & \text{if undirected} \\ \frac{\partial f(\mathcal{Y}^*)}{\partial \mathbf{A}}, & \text{if directed} \end{cases} \tag{5}$$

We can see that for both directed and undirected networks, the key to constructing the derivative network $\mathbf{B}$ is $\frac{\partial f(\mathcal{Y}^*)}{\partial \mathbf{A}}$. Therefore, in the remaining of this paper, we will mainly focus on effective and efficient computation of this quantity (i.e. $\frac{\partial f(\mathcal{Y}^*)}{\partial \mathbf{A}}$). Based on Eq. (5), we propose a generic algorithmic framework to solve Problem 1, which is summarized in Algorithm 1. The key idea is to generate the derivative network by applying Eq. (4) and (5). In Algorithm 1, it first runs the network mining algorithm and get the optimal model output $\mathcal{Y}^*$ (step 1). Based on the output $\mathcal{Y}^*$ and the corresponding scalar function f, it calculates the partial derivatives of $f(\mathcal{Y}^*)$ with respect to the network adjacency matrix $\mathbf{A}$ (step 2), and then uses it to generate the derivative network finally (step 3).

There are two key challenges remained in Algorithm 1,

- **(C1) How to compute Eq.** (4) **to generate the derivative network?** The key step to compute Eq. (4) requires the partial derivative of the optimal mining result $\mathcal{Y}^*$ with respect to the entire input network $\mathbf{A}$, and the optimal mining result $\mathcal{Y}^*$ itself involves a potentially complicated optimization problem.
- **(C2) How to scale up the computation to large networks?** Even if we can compute the influence of each edge with a reasonable time complexity (e.g. linear complexity w.r.t. the input network size), the entire Algorithm 1 (which iterates over *every* edge and calculates the corresponding influence) could still bear a superlinear complexity in both time and space, which makes it hard to scale up to large networks.

Algorithm 1: N2N Algorithm Framework

Input : The adjacency matrix $\mathbf{A}$, a mining algorithm $L(\mathbf{A}, \mathcal{Y}, \theta)$, and a scalar function $f(\cdot)$.
Output : The derivative network $\mathbf{B}$.

1 calculate $\mathcal{Y}^* = \underset{\mathcal{Y}}{\text{argmin}}\, L(\mathbf{A}, \mathcal{Y}, \theta)$;
2 calculate partial derivative $\frac{\partial f(\mathcal{Y}^*)}{\partial \mathbf{A}}$;
3 generate derivative network $\mathbf{B} = \frac{df(\mathcal{Y}^*)}{d\mathbf{A}}$ by Eq. (5) ;
4 **return** $\mathbf{B}$;

In the next section, we instantiate the proposed framework using three classic network mining tasks with effective and efficient algorithms to address these two challenges.

4 N2N INSTANTIATION AND COMPUTATION

In this section, we provide the instantiations of the proposed framework for three classic network mining tasks shown in Table 2. For each mining task, we start with the specific choice of $f(\cdot)$ function, then present the mathematical details on how to compute the derivative network $\mathbf{B}$ (i.e., C1 challenge), and finally design efficient ways to scale-up the computation (i.e., C2 challenge).

4.1 Instantiation #1: Ranking by HITS

A. Choice of $f(\cdot)$ function. Given an input network $\mathbf{G} = (\mathcal{V}, \mathcal{E})$ with $\mathbf{A}$ being the adjacency matrix, and let $\mathbf{u}$ and $\mathbf{v}$ be its associated hub vector and authority vector, respectively, HITS algorithm iteratively solves the linear equations $\mathbf{u} = \mathbf{A}\mathbf{v}$, $\mathbf{v} = \mathbf{A}'\mathbf{u}$ until converge to find the final hub and authority vectors. It is well-known that the hub vector $\mathbf{u}$ is the principal eigenvector associated with the leading eigenvalue of $\mathbf{A}\mathbf{A}'$, while the authority vector $\mathbf{v}$ is the principal eigenvector associated with the leading eigenvalue of $\mathbf{A}'\mathbf{A}$.

Our choice of $f(\cdot)$ function for HITS is inspired by [35], which proves that hubs and authorities are actually sensitive to the eigengap of $\mathbf{A}\mathbf{A}'$ and $\mathbf{A}'\mathbf{A}$. Furthermore, it is known that the eigenvalues of $\mathbf{A}\mathbf{A}'$ and $\mathbf{A}'\mathbf{A}$ are the same. Therefore, we choose the scalar function over the mining results $f(\cdot)$ to be the eigengap between first and second largest eigenvalues to reflect the stability of the ranking results. That is, $f(\mathcal{Y}^*) = \lambda_1 - \lambda_2$, where λ_1 and λ_2 are the first and second largest eigenvalues of $\mathbf{A}\mathbf{A}'$ and $\mathbf{A}'\mathbf{A}$.

B. Calculating the derivative network B. A key step in generating the derivative network $\mathbf{B}$ is to calculate the partial derivative $\frac{\partial f(\mathcal{Y}^*)}{\partial \mathbf{A}} = \frac{\partial \lambda_1}{\partial \mathbf{A}} - \frac{\partial \lambda_2}{\partial \mathbf{A}}$, which is summarized in Lemma 1.

Lemma 1. *For a given input network with adjacency matrix* $\mathbf{A}$*, the partial derivative of eigengap with respect to the adjacency matrix is* $\frac{\partial f(\mathcal{Y}^*)}{\partial \mathbf{A}} = 2(\mathbf{u}_1\mathbf{u}_1'\mathbf{A} - \mathbf{u}_2\mathbf{u}_2'\mathbf{A})$*, where* $\mathbf{u}_1$ *and* $\mathbf{u}_2$ *are the eigenvectors associated with* λ_1 *and* λ_2*, respectively.*

Table 2: Examples of Network Mining Algorithms

Mining Tasks	Loss Function L	Mining Results $\mathcal{Y}$	Additional Parameters θ	[1]Scalar Function $f(\cdot)$
Ranking by HITS	$\min_{\mathbf{u},\mathbf{v}} \ \|\|\mathbf{A}-\mathbf{u}\mathbf{v}'\|\|_F^2$	hub vector $\mathbf{u}$ authority vector $\mathbf{v}$	none	$f(\mathcal{Y}^*)=\lambda_1-\lambda_2$
Spectral Clustering	$\min_{\mathbf{U}} \ \mathrm{Tr}(\mathbf{U}'\mathbf{L}\mathbf{U})$ s.t. $\mathbf{U}'\mathbf{U}=\mathbf{I}$	matrix $\mathbf{U}$	number of clusters r	$f(\mathcal{Y}^*)=\sum_{i=1}^{r}\lambda_i$
Matrix Completion	$\min_{\mathbf{U},\mathbf{V}} \ \|\|\mathrm{proj}_\Omega(\mathbf{A}-\mathbf{U}\mathbf{V}')\|\|_F^2$ $+\lambda_u\|\|\mathbf{U}\|\|_F^2+\lambda_v\|\|\mathbf{V}\|\|_F^2$	user matrix $\mathbf{U}$ item matrix $\mathbf{V}$	latent dimensions r regularization parameters λ_u,λ_v	$f(\mathcal{Y}^*)=\|\|\mathbf{U}\mathbf{V}'\|\|_F^2$

Proof. We mainly show how to calculate $\frac{\partial\lambda_1}{\partial\mathbf{A}}$ since $\frac{\partial\lambda_2}{\partial\mathbf{A}}$ can be calculated in a very similar way. The key idea is to obtain the representation of $\frac{\partial\lambda_1}{\partial\mathbf{A}(i,j)}$, which is the partial derivative with respect to each element in $\mathbf{A}$. By the chain rule of matrix derivative, we have that $\frac{\partial\lambda_1}{\partial\mathbf{A}(i,j)}=\mathrm{Tr}[(\frac{\partial\lambda_1}{\partial\mathbf{A}\mathbf{A}'})'\frac{\partial\mathbf{A}\mathbf{A}'}{\partial\mathbf{A}(i,j)}]$.

We follow a similar strategy to calculate the first term in the chain rule by computing $\frac{\partial\lambda_1}{\partial(\mathbf{A}\mathbf{A}')(i,j)}$. Since $\mathbf{A}\mathbf{A}'$ is a real symmetric matrix, the partial derivative of its eigenvalue can be written as

$$\frac{\partial\lambda_1}{\partial(\mathbf{A}\mathbf{A}')(i,j)}=\mathbf{u}_1'\frac{\partial\mathbf{A}\mathbf{A}'}{\partial(\mathbf{A}\mathbf{A}')(i,j)}\mathbf{u}_1=\mathbf{u}_1'(i)\mathbf{u}_1(j) \tag{6}$$

where $\mathbf{u}_1$ is the eigenvector associated with λ_1. By Eq. (6), we can write out its matrix form solution where each element is the derivative of eigenvalue with respect to the corresponding element in $\mathbf{A}\mathbf{A}'$ as follows

$$\frac{\partial\lambda_1}{\partial\mathbf{A}\mathbf{A}'}=\mathbf{u}_1\mathbf{u}_1' \tag{7}$$

Then we show how to calculate the second term in chain rule $\frac{\partial\mathbf{A}\mathbf{A}'}{\partial\mathbf{A}(i,j)}$. By property of derivative of matrix multiplication, we have that

$$\frac{\partial\mathbf{A}\mathbf{A}'}{\partial\mathbf{A}(i,j)}=\frac{\partial\mathbf{A}}{\mathbf{A}(i,j)}\mathbf{A}'+\mathbf{A}\frac{\partial\mathbf{A}'}{\mathbf{A}(i,j)}=\mathbf{S}^{ij}\mathbf{A}'+\mathbf{A}\mathbf{S}^{ji} \tag{8}$$

where $\mathbf{S}^{ij}$ is a single-entry matrix with 1 at the i^{th} row and the j^{th} column and 0 elsewhere. Putting Eq. (7) and Eq. (8) together, we have that $\frac{\partial\lambda_1}{\partial\mathbf{A}(i,j)}=2[\mathbf{u}_1(i)(\mathbf{u}_1'\mathbf{A})(j)]$. With that in mind, we obtain its matrix form solution as

$$\frac{\partial\lambda_1}{\partial\mathbf{A}}=2\mathbf{u}_1\mathbf{u}_1'\mathbf{A} \tag{9}$$

Following the same strategy for the second largest eigenvalue λ_2, we can write out $\frac{\partial\lambda_2}{\partial\mathbf{A}(i,j)}=2[\mathbf{u}_2(i)(\mathbf{u}_2'\mathbf{A})(j)]$ with the matrix form solution to be

$$\frac{\partial\lambda_2}{\partial\mathbf{A}}=2\mathbf{u}_2\mathbf{u}_2'\mathbf{A} \tag{10}$$

Combining Eq. (9) and Eq. (10) together, we obtain the matrix form solution to complete the proof, i.e.,

$$\frac{\partial f(\mathcal{Y}^*)}{\partial\mathbf{A}}=\frac{\partial\lambda_1}{\partial\mathbf{A}}-\frac{\partial\lambda_2}{\partial\mathbf{A}}=2(\mathbf{u}_1\mathbf{u}_1'\mathbf{A}-\mathbf{u}_2\mathbf{u}_2'\mathbf{A}) \tag{11}$$

where each element in the matrix is its partial derivative with respect to corresponding element in the adjacency matrix $\mathbf{A}$. □

Based on Lemma 1, the derivative network $\mathbf{B}$ can be calculated by applying Eq. (11) and Eq. (5).

[1]It is worth mentioning that our proposed N2N framework is applicable to other choices of loss function $f(\cdot)$ as long as it is a differentiable or subdifferentiable scalar valued function, e.g., *L_p norm* of a vector, *L_1/L_2 norm* of a matrix, *soft maximum* to approximate the largest entry value in the vector, etc. It would be an interesting future direction to study (1) how to automatically choose the 'best' loss function for a specific mining task, and (2) how to generalize the proposed N2N framework for non-differentiable loss function.

C. Scale-up Computation. We have shown in Lemma 1 how to calculate $\mathbf{B}$. However, directly calculating $\mathbf{u}_1\mathbf{u}_1'\mathbf{A}$ and $\mathbf{u}_2\mathbf{u}_2'\mathbf{A}$ in Eq. (11) requires matrix-matrix multiplication for $n\times n$ matrices, which would impose an $\Omega(n^2)$ lower-bound on the complexity. To address this issue, we explore the low-rank structure of Eq. (11), where $\mathbf{u}_1$ is the $n\times 1$ eigenvector and $\mathbf{u}_1'\mathbf{A}$ is a $1\times n$ row vector. Similar properties also hold for $\mathbf{u}_2$ and $\mathbf{u}_2'\mathbf{A}$. Furthermore, since $\mathbf{u}_1$ is the eigenvector of $\mathbf{A}\mathbf{A}'$ which is actually the first left singular vector associated with the largest singular value of $\mathbf{A}$, we can show in Lemma 2 that $\mathbf{u}_1\mathbf{u}_1'\mathbf{A}$ can be computed by truncated singular value decomposition (SVD) on $\mathbf{A}$ in a much more compact way.

Lemma 2. *For a given input network with adjacency matrix $\mathbf{A}$, the partial derivative $\frac{\partial f(\mathcal{Y}^*)}{\partial\mathbf{A}}=2(\mathbf{u}_1\mathbf{u}_1'\mathbf{A}-\mathbf{u}_2\mathbf{u}_2'\mathbf{A})$ can be calculated by a rank-2 truncated SVD on $\mathbf{A}$.*

Proof. Let δ_1 and δ_2 be the first and second largest singular values associated with the left singular vectors $\mathbf{u}_1$ and $\mathbf{u}_2$, and let $\mathbf{v}_1$ and $\mathbf{v}_2$ be the corresponding right singular vectors. By truncated SVD on $\mathbf{A}$, we have

$$\mathbf{A}=[\mathbf{u}_1\ \mathbf{u}_2]\begin{bmatrix}\delta_1 & \\ & \delta_2\end{bmatrix}\begin{bmatrix}\mathbf{v}_1' \\ \mathbf{v}_2'\end{bmatrix}+\Delta \tag{12}$$

where $\Delta=\sum_{3\le i\le n}\mathbf{u}_i\delta_i\mathbf{v}_i'$ is defined as the residual matrix. Note that the left singular vectors are unitary, i.e. $\mathbf{u}_i'\mathbf{u}_j=0$ if $i\neq j$, then

$$\mathbf{u}_1\mathbf{u}_1'\mathbf{A}=\mathbf{u}_1\mathbf{u}_1'\mathbf{u}_1\delta_1\mathbf{v}_1'+\mathbf{u}_1\mathbf{u}_1'\mathbf{u}_2\delta_2\mathbf{v}_2'+\mathbf{u}_1\mathbf{u}_1'\Delta=\mathbf{u}_1\delta_1\mathbf{v}_1' \tag{13}$$

Similarly, for $\mathbf{u}_2\mathbf{u}_2'\mathbf{A}$, we have $\mathbf{u}_2\mathbf{u}_2'\mathbf{A}=\mathbf{u}_2\delta_2\mathbf{v}_2'$. Combining it with Eq. (13), we have that

$$\begin{aligned}\frac{\partial f(\mathcal{Y}^*)}{\partial\mathbf{A}}&=2\mathbf{u}_1\mathbf{u}_1'\mathbf{A}-\mathbf{u}_2\mathbf{u}_2'\mathbf{A}=2\mathbf{u}_1\delta_1\mathbf{v}_1'-\mathbf{u}_2\delta_2\mathbf{v}_2' \\ &=2[\mathbf{u}_1\ \mathbf{u}_2]\begin{bmatrix}\delta_1 & \\ & -\delta_2\end{bmatrix}\begin{bmatrix}\mathbf{v}_1' \\ \mathbf{v}_2'\end{bmatrix}\end{aligned} \tag{14}$$

which is equivalent to truncated SVD on the adjacency matrix $\mathbf{A}$ after *reversing* the second largest singular value δ_2. □

To be specific, based on Lemma 2, for each edge $\mathbf{A}(i,j)$, we compute the corresponding edge weight in the derivative network $\mathbf{B}$ as $\mathbf{B}(i,j)=2\delta_1\mathbf{u}_1(i)\mathbf{v}_1(j)-2\delta_2\mathbf{u}_2(i)\mathbf{v}_2(j)$. Then, the derivative network $\mathbf{B}$ can be easily computed by applying Eq. (5) with a linear complexity in both time and space.

Lemma 3. *(Time and space complexities). It takes $O(m+n)$ in time and $O(m+n)$ in space to generate the derivative network $\mathbf{B}$ for HITS on the input network $\mathbf{A}$, where n is the number of nodes and m is the number of edges.*

Proof. It takes $O(r(m+n)+r^2n)$ time complexity to perform the rank-r truncated SVD. Here we strictly let $r=2$, which reveals a $O(m+n)$ time complexity. Computing the partial derivatives of all edges takes $O(m)$ time in total. And it takes $O(m)$ time to calculate the influence of all edges and generate the derivative network $\mathbf{B}$. Thus, the overall time complexity is $O(m+n)$. For space complexity, it takes $O(m)$ space complexity to save the adjacency matrix $\mathbf{A}$ and the derivative network $\mathbf{B}$ in sparse format. It also takes $O(n)$ space to save the rank-2 SVD results. Therefore it has $O(m+n)$ space complexity. □

4.2 Instantiation #2: Spectral Clustering

A. Choice of $f(\cdot)$ function. Here, we have an undirected network $\mathbf{G}=(\mathcal{V},\mathcal{E})$ with $\mathbf{A}$ being the adjacency matrix. We have the Laplacian matrix $\mathbf{L}=\mathbf{D}-\mathbf{A}$, where $\mathbf{D}$ is the diagonal degree matrix of $\mathbf{A}$. Spectral clustering aims to find a matrix $\mathbf{U}$ by solving the optimization problem in Eq. (2), that maximizes the intra-cluster connectivity while minimizing the inter-cluster connectivity. Naturally, we have $\mathcal{Y}^*=\mathbf{U}$ and define $f(\mathcal{Y}^*)=\text{Tr}(\mathbf{U}'\mathbf{L}\mathbf{U})$, which aligns with the objective function of the spectral clustering.

B. Calculating the derivative network B. With our choice of $f(\cdot)$ function shown above, key steps to calculate the partial derivative $\frac{\partial f(\mathcal{Y}^*)}{\partial \mathbf{A}}$ are summarized in Lemma 4.

Lemma 4. *For a given input undirected network with adjacency matrix* $\mathbf{A}$*, the partial derivative of the sum of the* r *smallest eigenvalues with respect to the adjacency matrix is* $\frac{\partial f(\mathcal{Y}^*)}{\partial \mathbf{A}}=\text{diag}(\mathbf{U}\mathbf{U}')\mathbf{1}_{n\times n}-\mathbf{U}\mathbf{U}'$*, where columns in* $\mathbf{U}$ *are the associated eigenvectors,* $\mathbf{1}_{n\times n}$ *is an* $n\times n$ *matrix with all* 1 *as entries.*

Proof. Since $\mathbf{U}$ is the eigenvectors associated with the r smallest eigenvalues, we let λ_i be the i^{th} smallest eigenvalue and re-write the objective as $f(\mathcal{Y}^*)=\text{Tr}(\mathbf{U}'\mathbf{L}\mathbf{U})=\sum_{i=1}^{r}\lambda_i$. We first apply chain rule to get the influence of each edge (i,j) as

$$\frac{\partial f(\mathcal{Y}^*)}{\partial \mathbf{A}(i,j)}=\text{Tr}[(\frac{\partial f(\mathcal{Y}^*)}{\partial \mathbf{L}})'\frac{\partial \mathbf{L}}{\partial \mathbf{A}(i,j)}] \tag{15}$$

To calculate $\frac{\partial f(\mathcal{Y}^*)}{\partial \mathbf{L}}$, we have the following,

$$\frac{\partial f(\mathcal{Y}^*)}{\partial \mathbf{L}(i,j)}=\sum_{l=1}^{r}(\frac{\partial \lambda_l}{\partial \mathbf{L}(i,j)})=\sum_{l=1}^{r}\mathbf{U}(:,l)'\mathbf{S}^{ij}\mathbf{U}(:,l) \tag{16}$$

where $\mathbf{S}^{ij}$ is a single-entry matrix with 1 at the i^{th} row and the j^{th} column, and 0 elsewhere. With this, the matrix form solution of $\frac{\partial f(\mathcal{Y}^*)}{\partial \mathbf{L}}$ can be written as,

$$\frac{\partial f(\mathcal{Y}^*)}{\partial \mathbf{L}}=[\sum_{l=1}^{r}\mathbf{U}(:,l)'\mathbf{S}^{ij}\mathbf{U}(:,l)]_{1\le i\le n,1\le j\le n}=\mathbf{U}\mathbf{U}' \tag{17}$$

Since $\mathbf{L}=\mathbf{D}-\mathbf{A}$, we can calculate $\frac{\partial \mathbf{L}}{\partial \mathbf{A}(i,j)}$ as $\frac{\partial \mathbf{L}}{\partial \mathbf{A}(i,j)}=\mathbf{S}^{ii}-\mathbf{S}^{ij}$.

Putting everything together, we have the partial derivative of $f(\mathcal{Y}^*)$ with respect to $\mathbf{A}$ as follows,

$$\frac{\partial f(\mathcal{Y}^*)}{\partial \mathbf{A}}=\text{diag}(\mathbf{U}\mathbf{U}')\mathbf{1}_{n\times n}-\mathbf{U}\mathbf{U}' \tag{18}$$

where $\mathbf{1}_{n\times n}$ is an $n\times n$ matrix with all 1 as entries. □

Then we can apply Eq. (5) to calculate its corresponding derivative network $\mathbf{B}$.

C. Scale-up computation. Major computational challenges in Eq. (18) lie in the matrix multiplication of $\text{diag}(\mathbf{U}\mathbf{U}')\mathbf{1}_{n\times n}$ and $\mathbf{U}\mathbf{U}'$, which, if computed in a straight-forward way, would require $O(n^3)$ complexity in time and $O(n^2)$ in space. We propose to address the computational challenges by exploring the low-rank structure of $\text{diag}(\mathbf{U}\mathbf{U}')\mathbf{1}_{n\times n}-\mathbf{U}\mathbf{U}'$.

Let us denote the i^{th} row of $\mathbf{U}$ by $\mathbf{u}_i'$ and use $\mathbf{1}_{n\times 1}$ to denote the $n\times 1$ column vector filled by 1s and $\mathbf{1}_{1\times n}$ to denote the $1\times n$ row vector filled by 1s, then $\text{diag}(\mathbf{U}\mathbf{U}')\mathbf{1}_{n\times n}$ can be re-written as follows,

$$\text{diag}(\mathbf{U}\mathbf{U}')\mathbf{1}_{n\times n}=\text{diag}(\mathbf{U}\mathbf{U}')\mathbf{1}_{n\times 1}\mathbf{1}_{1\times n}=\begin{bmatrix}\mathbf{u}_1'\mathbf{u}_1\\ \mathbf{u}_2'\mathbf{u}_2\\ \cdots\\ \mathbf{u}_n'\mathbf{u}_n\end{bmatrix}\mathbf{1}_{1\times n} \tag{19}$$

such that the column vector is of size $n\times 1$ and $\mathbf{1}_{1\times n}$ is a $1\times n$ row vector. Then $\frac{\partial f(\mathcal{Y}^*)}{\partial \mathbf{A}}$ can be represented in the following low-rank form,

$$\frac{\partial f(\mathcal{Y}^*)}{\partial \mathbf{A}}=\begin{bmatrix}\mathbf{u}_1'\mathbf{u}_1\\ \mathbf{u}_2'\mathbf{u}_2\\ \cdots\\ \mathbf{u}_n'\mathbf{u}_n\end{bmatrix}\mathbf{1}_{1\times n}-\begin{bmatrix}\mathbf{u}_1'\\ \mathbf{u}_2'\\ \cdots\\ \mathbf{u}_n'\end{bmatrix}[\mathbf{u}_1\ \mathbf{u}_2\ \ldots\ \mathbf{u}_n] \tag{20}$$

With the low-rank structure, to get the partial derivative of edge (i,j) (i.e. element at the i^{th} row and the j^{th} column in $\frac{\partial f(\mathcal{Y}^*)}{\partial \mathbf{A}}$), we can simply calculate it as $\mathbf{u}_i'\mathbf{u}_i-\mathbf{u}_i'\mathbf{u}_j$. Then the influence of an edge (i,j) can be calculated as $\mathbf{B}(i,j)=\mathbf{u}_i'\mathbf{u}_i+\mathbf{u}_j'\mathbf{u}_j-\mathbf{u}_i'\mathbf{u}_j-\mathbf{u}_j'\mathbf{u}_i$, which takes $O(r)$ in time. In this way, we achieve a linear time and space complexity w.r.t. the input network size, as stated in the following lemma.

Lemma 5. *(Time and space complexities). It takes* $O(r(m+n)+r^2n)$ *complexity in time and* $O(rn+m)$ *in space to generate the derivative network* $\mathbf{B}$ *for the task of spectral clustering, where* n *and* m *are the numbers of nodes and edges of the input network respectively, and* r *is the number of clusters.*

Proof. It takes $O(n)$ time to get the Laplacian matrix $\mathbf{L}$ and $O(r(m+n)+r^2n)$ to perform a rank-r eigen-decomposition on $\mathbf{L}$. Precomputing $\mathbf{u}_i'\mathbf{u}_i$ for $1\le i\le n$ in Eq. (20) takes $O(rn)$ time complexity. Then it takes $O(rm)$ time to calculate the partial derivatives and the influence of all edges in the derivative network. Thus the overall time complexity to calculate the derivative network $\mathbf{B}$ is $O(r(m+n)+r^2n)$. Regarding the space complexity, it takes $O(m)$ space to save the adjacency matrix $\mathbf{A}$ and the derivative network $\mathbf{B}$ in sparse format. Also, it takes $O(n)$ space to store the low-rank form of $\text{diag}(\mathbf{U}\mathbf{U}')\mathbf{1}$ and $O(rn)$ space to save the matrix $\mathbf{U}$. Hence, the space complexity is $O(rn+m)$. □

4.3 Instantiation #3: Matrix Completion

A. Choice of $f(\cdot)$ function. For matrix factorization-based completion, it solves the optimization problem in Eq. (3) to find two low-rank matrices, $\mathbf{U}$ and $\mathbf{V}$ with r latent factors. Here, the input is a user-item bipartite network $\mathbf{A}$ with n_1 users, n_2 items and m observations, where $\mathbf{A}(i,j)$ is the rating of the j^{th} item made by the i^{th} user. Since $\hat{\mathbf{A}}=\mathbf{U}\mathbf{V}'$ is often used to complete/infer the missing entries in the observed rating matrix $\mathbf{A}$, we choose $f(\mathcal{Y}^*)=||\mathbf{U}\mathbf{V}'||_F^2$.

B. Calculating the derivative network B. Since the bipartite network $\mathbf{A}$ is often represented as an asymmetric $n_1\times n_2$ matrix, we have $\mathbf{B}=\frac{\text{d}f(\mathcal{Y}^*)}{\text{d}\mathbf{A}}=\frac{\partial f(\mathcal{Y}^*)}{\partial \mathbf{A}}$ by Eq. (5). Similar to the previous two instantiations, we aim to get the representation of $\frac{\partial f(\mathcal{Y}^*)}{\partial \mathbf{A}(i,j)}$ and

then write out the matrix form solution. We first denote $\mathbf{X} = \mathbf{U}\mathbf{V}'$ and then apply chain rule to get

$$\frac{\partial f(\mathcal{Y}^*)}{\partial \mathbf{A}(i,j)} = \sum_{l=1}^{n_1}\sum_{t=1}^{n_2} \frac{\partial f(\mathcal{Y}^*)}{\partial \mathbf{X}(l,t)} \frac{\partial \mathbf{X}(l,t)}{\partial \mathbf{A}(i,j)} \tag{21}$$

However, it is non-trivial to calculate Eq. (21). We present an accurate and efficient solution to calculate $\frac{\partial f(\mathcal{Y}^*)}{\partial \mathbf{X}(l,t)}$ and $\frac{\partial \mathbf{X}(l,t)}{\partial \mathbf{A}(i,j)}$ in Lemma 6.

Lemma 6. *For a given bipartite network* $\mathbf{A}$*, with* $\mathbf{U}$ *and* $\mathbf{V}$ *being the optimal model output of Eq.* (3)*, the influence of* (i,j) *rating is* $\mathbb{I}(i,j) = 2\mathbf{U}(i,:)\mathbf{V}'\mathbf{V}\mathbf{C}_i^{-1}\mathbf{V}(j,:)' + 2\mathbf{V}(j,:)\mathbf{U}'\mathbf{U}\mathbf{D}_j^{-1}\mathbf{U}(i,:)'$*, where* $\mathbf{C}_i = \lambda_u\mathbf{I} + \sum_{k\in\Omega_i}\mathbf{V}(k,:)'\mathbf{V}(k,:)$, $\mathbf{D}_j = \lambda_v\mathbf{I} + \sum_{k\in\Omega_j}\mathbf{U}(k,:)'\mathbf{U}(k,:)$*,* Ω_i *and* Ω_j *are sets of indices for non-zero entries of user* i *and item* j*.*

Proof. We solve the two terms in Eq. (21) one by one. First, we show how to compute $\frac{\partial f(\mathcal{Y}^*)}{\partial \mathbf{X}(l,t)}$. Recall that we define $f(\mathcal{Y}^*) = ||\mathbf{U}\mathbf{V}'||_F^2$ and $\mathbf{X} = \mathbf{U}\mathbf{V}'$. By the derivative of matrix norm, we have $\frac{\partial f(\mathcal{Y}^*)}{\partial \mathbf{X}} = 2\mathbf{X} = 2\mathbf{U}\mathbf{V}'$. Thus, the partial derivative $\frac{\partial f(\mathcal{Y}^*)}{\partial \mathbf{X}(l,t)}$ can be denoted as

$$\frac{\partial f(\mathcal{Y}^*)}{\partial \mathbf{X}(l,t)} = 2\mathbf{X}(l,t) \tag{22}$$

This completes the calculation of the first term $\frac{\partial f(\mathcal{Y}^*)}{\partial \mathbf{X}(l,t)}$ in Eq. (21).

Next, we show how to compute $\frac{\partial \mathbf{X}(l,t)}{\partial \mathbf{A}(i,j)}$. Since $\mathbf{X} = \mathbf{U}\mathbf{V}'$, we have $\mathbf{X}(l,t) = \mathbf{U}(l,:)\mathbf{V}(t,:)'$. Then the second term in Eq. (21) $\frac{\partial \mathbf{X}(l,t)}{\partial \mathbf{A}(i,j)}$ can be re-written as $\frac{\partial \mathbf{X}(l,t)}{\partial \mathbf{A}(i,j)} = \frac{\partial \mathbf{U}(l,:)}{\partial \mathbf{A}(i,j)}\mathbf{V}(t,:)' + \mathbf{U}(l,:)(\frac{\partial \mathbf{V}(t,:)}{\partial \mathbf{A}(i,j)})'$. However, it is hard to directly calculate $\frac{\partial \mathbf{U}(l,:)}{\partial \mathbf{A}(i,j)}$ and $\frac{\partial \mathbf{V}(t,:)}{\partial \mathbf{A}(i,j)}$ since there is no straightforward closed-form solution for $\mathbf{U}$ and $\mathbf{V}$ with respect to $\mathbf{A}(i,j)$. To solve this problem, we follow [30, 33] and consider the KKT conditions of Alternating Least Square (ALS) method, which are shown as follows,

$$\begin{aligned}
\lambda_u\mathbf{U}(l,:) &= \sum_{k\in\Omega_l}(\mathbf{A}(l,k) - \mathbf{U}(l,:)\mathbf{V}(k,:)')\mathbf{V}(k,:) \\
\lambda_v\mathbf{V}(t,:) &= \sum_{k\in\Omega_t}(\mathbf{A}(k,t) - \mathbf{U}(k,:)\mathbf{V}(t,:)')\mathbf{U}(k,:)
\end{aligned} \tag{23}$$

where Ω_l and Ω_t are sets of indices for non-zero entries of user l and item t, respectively. Following the equations in Eq. (23), we obtain the following partial derivatives,

$$\begin{aligned}
\frac{\partial \mathbf{U}(l,:)}{\partial \mathbf{A}(i,j)} &= \begin{cases} \mathbf{V}(j,:)[\lambda_u\mathbf{I} + \sum_{k\in\Omega_l}\mathbf{V}(k,:)'\mathbf{V}(k,:)]^{-1}, & \text{if } i = l \\ 0, & \text{otherwise} \end{cases} \\
\frac{\partial \mathbf{V}(t,:)}{\partial \mathbf{A}(i,j)} &= \begin{cases} \mathbf{U}(i,:)[\lambda_v\mathbf{I} + \sum_{k\in\Omega_t}\mathbf{U}(k,:)'\mathbf{U}(k,:)]^{-1}, & \text{if } j = t \\ 0, & \text{otherwise} \end{cases}
\end{aligned} \tag{24}$$

Since $\mathbf{C}_l = \lambda_u\mathbf{I} + \sum_{k\in\Omega_l}\mathbf{V}(k,:)'\mathbf{V}(k,:)$, $\mathbf{D}_t = \lambda_v\mathbf{I} + \sum_{k\in\Omega_t}\mathbf{U}(k,:)'\mathbf{U}(k,:)$, we have the following equation

$$\frac{\partial \mathbf{X}(l,t)}{\partial \mathbf{A}(i,j)} = \mathbf{V}(j,:)\mathbf{C}_i^{-1}\mathbf{V}(t,:)' + \mathbf{U}(l,:)\mathbf{D}_j^{-1}\mathbf{U}(i,:)' \tag{25}$$

Combining Eq. (22) and Eq. (25), we re-write Eq. (21) as

$$\begin{aligned}
\frac{\partial f(\mathcal{Y}^*)}{\partial \mathbf{A}(i,j)} &= 2\sum_{t=1}^{m}\mathbf{U}(i,:)\mathbf{V}(t,:)'\frac{\partial \mathbf{U}(i,:)}{\partial \mathbf{A}(i,j)}\mathbf{V}(t,:)' \\
&\quad + 2\sum_{l=1}^{n}\mathbf{U}(l,:)\mathbf{V}(j,:)'\mathbf{U}(l,:)(\frac{\partial \mathbf{V}(j,:)}{\partial \mathbf{A}(i,j)})' \\
&= 2\sum_{t=1}^{m}\mathbf{U}(i,:)\mathbf{V}(t,:)'\mathbf{V}(j,:)\mathbf{C}_i^{-1}\mathbf{V}(t,:)' \\
&\quad + 2\sum_{l=1}^{n}\mathbf{U}(l,:)\mathbf{V}(j,:)'\mathbf{U}(l,:)\mathbf{D}_j^{-1}\mathbf{U}(i,:)' \\
&= 2\mathbf{U}(i,:)\mathbf{V}'\mathbf{V}\mathbf{C}_i^{-1}\mathbf{V}(j,:)' + 2\mathbf{V}(j,:)\mathbf{U}'\mathbf{U}\mathbf{D}_j^{-1}\mathbf{U}(i,:)'
\end{aligned} \tag{26}$$

which completes the proof. □

C. Scale-up computation. Simply calculating the influence by Eq. (26) will take $O(r^2(n_1+n_2))$ time complexity for *each* rating. Therefore, the amortized time complexity will still be superlinear w.r.t. the network size ($O(r^2(n_1+n_2)m)$), which makes it not scalable to large networks. However, if we take one more look at Eq. (26), we can find out that $\mathbf{C}_i^{-1}$ for each user i, $\mathbf{D}_j^{-1}$ for each item j, $\mathbf{U}'\mathbf{U}$ and $\mathbf{V}'\mathbf{V}$ are globally shared by all elements in $\mathbf{A}$. Thus, we can precompute them for all users and items once we have trained the optimal model output $\mathbf{U}$ and $\mathbf{V}$. With the help of precomputation, for each edge $\mathbf{A}(i,j)$ in the input bipartite network $\mathbf{A}$, we can calculate its influence as $\mathbf{B}(i,j) = 2\mathbf{U}(i,:)\mathbf{V}'\mathbf{V}\mathbf{C}_i^{-1}\mathbf{V}(j,:)' + 2\mathbf{V}(j,:)\mathbf{U}'\mathbf{U}\mathbf{D}_j^{-1}\mathbf{U}(i,:)'$ in $O(r)$ time. Since we only compute $\mathbf{B}(i,j)$ for observed rating $\mathbf{A}(i,j)$, we efficiently reduce the overall time complexity to be linear w.r.t. the input network size.

Lemma 7. *(Time and space complexities). The amortized time complexity to generate the derivative network* $\mathbf{B}$ *is* $O(r^3(n_1+n_2) + r^2m)$*, where* n_1 *and* n_2 *are numbers of users and items,* r *is the dimension of latent factors, and* m *is the total number of observed ratings. It takes* $O(r^2(n_1+n_2) + m)$ *complexity in space.*

Proof. It takes $O(r^3(n_1+n_2)+r^2m)$ to train the model by ALS. It takes $O(r^2|\Omega_i|)$ and $O(r^2|\Omega_j|)$ to precompute $\mathbf{C}_i$ and $\mathbf{D}_j$, where $|\Omega_i|$ and $|\Omega_j|$ are the number of observed feedback for user i and item j, respectively. And it takes $O(r^3)$ complexity to inverse each $\mathbf{C}_i$ and $\mathbf{D}_j$ for all $1 \le i \le n_1$ and $1 \le j \le n_2$. Thus, the amortized complexity in precomputation stage is $O(r^3(n_1+n_2) + r^2m)$. And the time complexities to precompute $\mathbf{U}'\mathbf{U}$ and $\mathbf{V}'\mathbf{V}$ are $O(r^2n)$ and $O(r^2m)$ respectively. Then, to calculate the influence for one element $\mathbf{A}(i,j)$, it takes $O(r^2)$ time complexity. Thus, the overall amortized time complexity to calculate the influence for all observed element in $\mathbf{A}$ is $O(r^3(n_1+n_2) + r^2m)$. Regarding space complexity, it takes $O(m)$ space to save the bipartite network $\mathbf{A}$. Each $\mathbf{C}_i$ and $\mathbf{D}_j$ for all $1 \le i \le n_1$ and $1 \le j \le n_2$ takes $O(r^2)$ space. And we also need to save the precomputed $\mathbf{U}'\mathbf{U}$ and $\mathbf{V}'\mathbf{V}$ which also requires $O(r^2)$ space. Thus, it takes $O(r^2(n_1+n_2) + m)$ complexity in space. □

5 EXPERIMENTS

In this section, we perform empirical evaluations on the proposed N2N framework. All experiments are designed to answer the following questions:

- **Effectiveness.** How effective is the proposed N2N framework with respect to the corresponding mining task?

- **Efficiency.** How efficient and scalable is the proposed N2N framework to generate the derivative network?

5.1 Setup

A - Datasets. We test our algorithms on a diverse set of real-world datasets, all of which are publicly available. The statistics of these datasets is summarized in Table 3.

Table 3: Statistics of Datasets

Task	Domain	Type	Dataset	Nodes (Users/Items)	Edges
HITS	Cit	D	cit-hepth	27,770	352,807
		D	cit-dblp	12,590	49,759
		D	cora	23,166	91,500
		D	patent	3,774,768	16,518,948
	Social	D	gplus	23,628	39,242
		D	epinions	75,879	508,837
	Infra	D	gnutella	8,114	26,013
Spectral Clustering	CA	U	astroph	18,772	198,110
		U	condmat	23,133	93,497
		U	grqc	5,242	14,496
		U	ca-hepph	12,008	118,521
	Social	U	hamster	1,858	12,534
		U	douban	154,908	327,162
	Infra	U	twin	14,274	20,573
Matrix Completion	Rating	B	lastfm	1,892/17,632	92,834
		B	delicious	1,867/69,223	437,593
		B	movielens	610/9,724	100,836
		B	ml-20m	138,493/26,744	20,000,264

(In Type, D: directed; U: undirected; B: bipartite.)

There are three types of datasets, including directed uni-partite networks ('D') for the ranking task, undirected uni-partite networks ('U') for the spectral clustering task, and bipartite networks ('B') for the matrix completion task. Among them, the three largest ones (i.e., *patent*, *douban* and *ml-20m*) are used for scalability experiments. These datasets come from a variety of application domains, including social networks (Social), citation networks (Cit), collaboration networks (CA), physical infrastructure networks (Infra) and rating networks (Rating). The detailed descriptions of these datasets are as follows.

- Social Networks. Here, nodes are users and edges indicate social relationships. Among them, *gplus* [27] is a directed network of Google+ user-user links, which is a social network by Google. A directed edge indicates that one user has the other user in his/her circle. *epinions* [29] is a directed who-trust-whom network from consumer reviews website Epinions[2]. *hamster* [27] is a directed network of friendship among users of the website hamsterster.com. *douban* [27] is the social network of a Chinese online recommendation site Douban[3].
- Citation Networks. Here, each node is a research paper. If a paper i cites paper j, there is a directed edge from node i to node j. *cit-hepph* [29] is an ArXiv HEP-PH (High Energy Physics - Phenomenology) citation network. The data covers papers from January 1993 to April 2003. *cit-dblp* [27] is a directed network of citation data on DBLP[4], a database of computer science bibliography. *cora* [27] is the CORA citation network. *patent* [29] is a directed network of all the utility patents granted from 1963 to 1999.
- Collaboration Networks. Here, nodes are researchers and two individuals are connected if they have collaborated together. We use four collaboration networks in the field of Physics from arXiv preprint archive[5]: Astro Physics (*astroph*), Condense Matter Physics (*condmat*), General Relativity and Quantum Cosmology (*grqc*), and High Energy Physics - Phenomenology (*ca-hepph*).
- Physical Infrastructure Networks. This domain refers to the networks of physical infrastructure entities. Nodes correspond to physical infrastructure, and edges are connections. *gnutella* [29] is a snapshots of Gnutella peer-to-peer file sharing network collected on August 9, 2002. *twin* [27] is an undirected network of cities connected by sister city relationships. The dataset is extracted from WikiData[6].
- Rating Networks. It is a collection of bipartite networks that consist of feedback given to items by users, weighted by a rating value. Four different rating networks are used. Among them, *lastfm* is extracted from the music streaming service Last.fm[7]. If a user i listened to a song by artist j, its corresponding feedback $\mathbf{A}(i,j) = 1$, otherwise it is 0. *delicious* is extracted from the social bookmark sharing service website Delicious[8]. If a user i bookmarked a particular URL j, the feedback $\mathbf{A}(i,j) = 1$, otherwise it is 0. *movielens* and *ml-20m* are two rating networks of users to movies provided by GroupLens Research[9]. An edge between a user and a movie represents a rating of the movie by the user. Each rating ranges from 0.5 to 5.0.

B - Baseline Methods. We compare the proposed method N2N (Algorithm 1) with several baseline methods. We briefly summarize the baseline methods as below.

- *Top Degrees (Degree).* We define the degree score of an edge (u,v) as follows.

$$d(u,v) = \begin{cases} d(u) + d(v), & \text{if undirected} \\ (d(u) + d(v)) \times d(u), & \text{if directed} \end{cases} \tag{27}$$

  where $d(u)$ represents the degree of node u.
- *Top Eigenvector Centrality (EigenCentrality).* We define the eigenvector centrality score of an edge (u,v) as follows.

$$\mathbf{e}ig(u,v) = \begin{cases} \mathbf{e}ig(u) + \mathbf{e}ig(v), & \text{if undirected} \\ (\mathbf{e}ig(u) + \mathbf{e}ig(v)) \times \mathbf{e}ig(u), & \text{if directed} \end{cases} \tag{28}$$

  where $eig(u)$ is the eigenvector centrality score of node u.
- *HITS.* We define HITS score of an edge (u,v) as follows.

$$HITS(u,v) = hub(u) + hub(v) + auth(u) + auth(v) \tag{29}$$

  where $hub(u)$ and $auth(u)$ represent the hub score and authority score of node u, respectively.
- *Contain.* Contain is an algorithm proposed in [8] to optimize the network connectivity. It iteratively selects a network element (e.g. a node or an edge) with the highest impact score on a user-defined connectivity measurement.

[2] http://www.epinions.com/
[3] https://www.douban.com
[4] http://dblp.uni-trier.de/
[5] https://arxiv.org/
[6] https://www.wikidata.org
[7] https://www.last.fm
[8] https://del.icio.us
[9] https://grouplens.org/datasets/movielens/

- *Aurora.* Aurora is an algorithm proposed in [20] for PageRank auditing problem. It iteratively selects a network element (e.g. a node or an edge) with the highest influence on the PageRank ranking vector of a network.

Algorithm 2 describes the procedure to select edges for our proposed N2N method and baseline methods (including top degrees, top eigenvector centrality and HITS). The edge scoring function corresponds to the score we defined in Section 5. That is, $C(u,v) = \mathbb{I}(u,v)$ for N2N, $C(u,v) = d(u,v)$ for top degrees, $C(u,v) = eig(u,v)$ for top eigenvector centrality and $C(u,v) = HITS(u,v)$ for HITS.

Algorithm 2: Description of Comparison Methods

Input : The adjacency matrix $\mathbf{A}$, an edge scoring function $C(\cdot)$, integer budget k.

Output : A set of k edges $\mathcal{S}$ with highest edge scores.

initialize $\mathcal{S} = \emptyset$;
let $\tilde{\mathbf{A}} = \mathbf{A}$;
while $|\mathcal{S}| \neq k$ **do**
  find edge $u, v = \underset{(u,v)\in\tilde{\mathbf{A}}}{\text{argmax}}\, C(u,v)$;
  add edge (u,v) to $\mathcal{S}$;
  remove edge (u,v) from $\tilde{\mathbf{A}}$;
return $\mathcal{S}$;

C - Metrics. Generally speaking, we want to measure how the mining results would change if we perturb the network elements (e.g., nodes, edges) for each mining task (i.e. HITS, spectral clustering, matrix completion).

More specifically, for HITS, we measure how the hub and authority ranking change in total if we perturb the set of network elements. We define the distortion error metric for HITS as

$$err = ||\mathbf{u} - \tilde{\mathbf{u}}||_2 + ||\mathbf{v} - \tilde{\mathbf{v}}||_2 \tag{30}$$

where $\mathbf{u}$ and $\mathbf{v}$ are original hub and authority vectors, while $\tilde{\mathbf{u}}$ and $\tilde{\mathbf{v}}$ are the hub and authority vectors after perturbation.

For spectral clustering, we measure changes in cluster assignments after we perturb the set of network elements. We use the normalized mutual information (NMI) to measure the agreement between two cluster assignments before and after the perturbation. We define the distortion error metric as

$$err = 1 - NMI(C, \tilde{C}) = 1 - \frac{2MI(C, \tilde{C})}{H(C) + H(\tilde{C})} \tag{31}$$

where C and $\tilde{C}$ are the cluster assignments before and after perturbation, $\text{MI}(C, \tilde{C})$ is the mutual information between C and $\tilde{C}$, and $H(C)$ is the entropy of assignment C. Notice that a larger err means a bigger difference between the clustering assignment before and after perturbation, which implies more effective attacking on clustering.

Regarding matrix completion, we measure how the model prediction changes after we perturb the set of network elements. We measure its change as follows,

$$\text{RMSE} = \sqrt{\sum_{(i,j)\in\Omega^C} (\hat{\mathbf{A}}(i,j) - \tilde{\mathbf{A}}(i,j))^2 / |\Omega^C|} \tag{32}$$

where $\hat{\mathbf{A}}(i,j) = \mathbf{U}(i,:)\mathbf{V}(j,:)'$ is the prediction made by the original model output, $\tilde{\mathbf{A}}(i,j) = \tilde{\mathbf{U}}(i,:)\tilde{\mathbf{V}}(j,:)'$ is the prediction made by model after perturbation, and $|\Omega^C|$ is the cardinality of the complementary set of Ω. Notice that a larger RMSE means a bigger difference in predicting a user's preference, which implies more effective attacking on the recommender system.

D - Machine Configuration and Reproducibility. All experiments are performed on a Windows PC with 3.8GHz Intel Core i7-9800X CPU and 64 GB RAM. All datasets are publicly available. The methods are programmed in Python 3.6. We will release the source code upon publication of this paper.

Regarding detailed parameter settings in the experimental evaluations, we set the budget size $k = 50$ for ranking and spectral clustering, and $k = 10$ for matrix completion. For *Contain* method, we use the leading eigenvalue as the connectivity measurement and set its rank parameter to 80 (see details in [8]). For spectral clustering, the number of clusters is set to 5. Regarding matrix completion, we set the latent dimension $r = 10$, the regularization parameters $\lambda_u = 0.5$ and $\lambda_v = 0.5$, and the number of training iterations to 10. Regarding the random initialization of cluster centroids in spectral clustering and low-rank matrices $\mathbf{U}$, $\mathbf{V}$ in matrix completion, the random seed is uniformly selected from 1 to 50 for each dataset.

5.2 Effectiveness Results

We perform quantitative effectiveness comparison with baseline methods. The experiments are designed to explore the potential of the proposed N2N framework in attacking the corresponding network mining task, i.e. the mining results could be distorted greatly by removing a set of influential network edges in the derivative network.

Quantitative Comparison Results. The quantitative comparison results for HITS, spectral clustering and matrix completion across 15 different datasets are shown from Figure 1 to Figure 3, respectively. From those figures, we can see that the proposed N2N framework (the leftmost red solid bar in Figures 1, 2 and 3) consistently outperforms other baseline methods in all datasets, which indicates that the derivative network generated by our proposed N2N framework can indeed effectively attacking the network mining tasks by a few edge deletion.

Effect of Budget Size k. We explore the power of N2N on different budget size. Note that we only show the results of HITS ranking here. We set the budget size k from 1 to 50. From the results shown in Figure 5, we can observe that our proposed N2N method (red solid line in Figure 5) outperforms other baseline methods on different budget size k.

5.3 Efficiency Results

We show the running time vs. the input network size for HITS, spectral clustering and matrix completion in Figure 4. We can see that the proposed N2N framework scales linearly with respect to the input network size $m+n$ in all three instantiations. These results are consistent with our complexity analysis in Lemma 3, 5 and 7, which states that the derivative network $\mathbf{B}$ can be computed in linear time with respect to the number of edges m and the number of nodes n.

6 RELATED WORK

In this section, we review the related work from the following two perspectives: (1) network mining and (2) network interventions.

Network mining aims to find interesting patterns from the underlying network data. Classic network mining tasks include

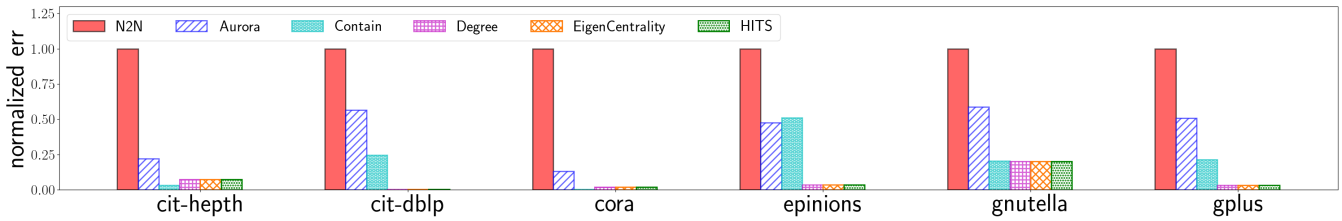


Figure 1: HITS ranking attacking results. Higher is better (i.e., more effective attacking). Best viewed in color.

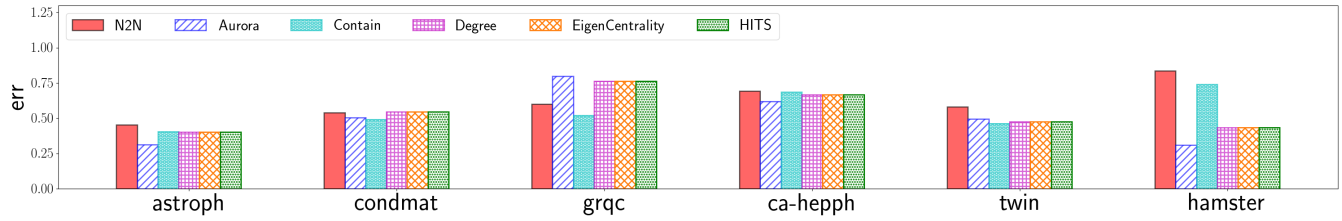


Figure 2: Spectral clustering attacking results. Higher is better (i.e., more effective attacking). Best viewed in color.

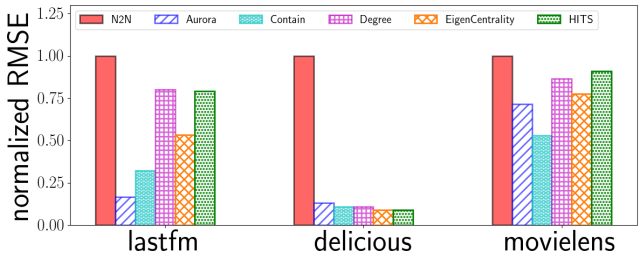


Figure 3: Matrix completion attacking results. Higher is better (i.e., more effective attacking). Best viewed in color.

ranking, clustering, completion, and many more. Regarding ranking on networks, PageRank [36] and HITS [23] are two most well-known algorithms to measure the importance of nodes. Many variants have been developed in the literature, including personalized PageRank [17], random walk with restart [42], randomized HITS and subspace HITS [35]. As for clustering methods, spectral clustering [34] is one of the most well-known and widely-used algorithms. Other representative clustering methods on networks include hierarchical clustering [2, 13], cut and conductance based methods [18, 38, 44], and online clustering method [4]. Matrix completion is another important network mining task [3, 19, 25, 26]. To name a few, Yao et. al. apply collaborative filtering to learn the hidden relationships between nodes [43]. Another example is [37], which uses the side information to help complete the network.

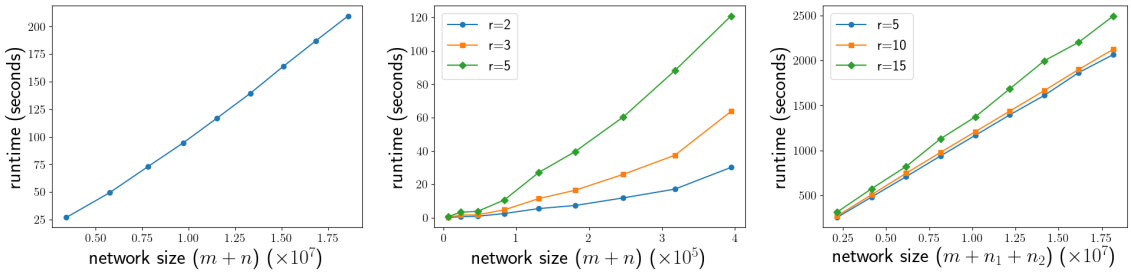
Network intervention has been attracting a lot of research interests. Here, we review several research lines that are most relevant to this work. First, finding influential nodes to maximize the spread of influence in a social network (i.e. influence maximization) is a very active research area [14, 16]. Kempe et. al. [21, 22] discover its diminishing returns property. Another important line of research lies in optimizing network structure. In [1], Backstrome and Leskovec study the problem of learning the optimal edge weights with node and edge attributes/features. In [41], the authors propose a method to use side information to refine the network topology. In [31], Li et. al. learn the query-specific optimal networks for random walk with restart. Regarding network connectivity optimization, Chen et. al. explore the diminishing returns property and propose several methods to optimize the network connectivity with both edge-level and node-level manipulations for various connectivity measurements on plain networks [8–10] and multi-layered networks [7]. In [28, 32], the authors aim to optimize the network connectivity by studying the triangle minimization problem. Chan et. al. study the optimization problem for network robustness in [5, 6]. As for adversarial network mining, several research work has been proposed to study the adversarial attacks on networks for different tasks, e.g. classification [12, 45], link prediction [11] and node embeddings [40]. Li et. al. [30] propose a poisoning attack strategy using gradients for collaborative filtering problem. The proposed N2N framework bears subtle difference from this work since we face different objective functions to calculate the gradients and propose a much more efficient algorithm with a linear complexity to speed up the computational process. In [20], Kang et. al. study the problem of auditing PageRank, which can be conceptually seen as an instantiation of the network derivative mining problem for the task of PageRank.

7 CONCLUSION

In this paper, we introduce the network derivative mining problem. It finds a derivative network which measures the influence of the corresponding edges of the input network given a specific mining algorithm. We formulate the network derivative network problem as an optimization problem and measure the influence of edges as the rate of change in a task-specific scalar function with respect to the corresponding edges of the input network. We further instantiate the proposed framework (N2N), with three classic network mining tasks (i.e., HITS ranking, spectral clustering and matrix completion). We propose effective and efficient ways to construct the corresponding derivative network linearly w.r.t. the input network size. The extensive experimental evaluations on more than 10 datasets demonstrate that our proposed N2N framework is effective in adversarial network mining, consistently outperforms baseline methods, and scale linearly to large networks. In the future, we would like to explore efficient ways to update the derivative network in the dynamic setting.

ACKNOWLEDGEMENT

This work is supported by NSF (IIS-1651203, IIS-1715385), and DHS (2017-ST-061-QA0001).



(a) HITS Ranking **(b) Spectral Clustering** **(c) Matrix Completion**

Figure 4: The scalability of the proposed N2N on different network mining tasks.

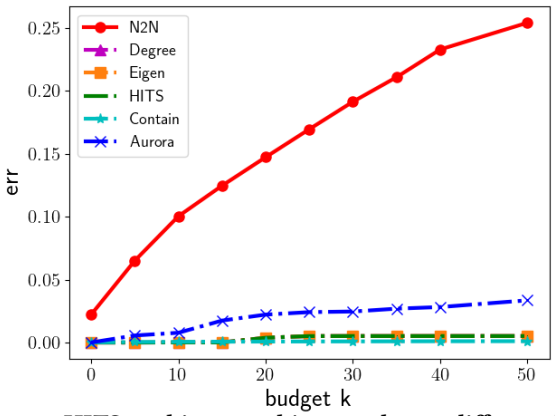


Figure 5: HITS ranking attacking results on different budget size. Higher is better (i.e., more effective attacking). Best viewed in color.

REFERENCES

[1] Lars Backstrom and Jure Leskovec. 2011. Supervised random walks: predicting and recommending links in social networks. In *WSDM*. ACM, 635–644.

[2] Ulrik Brandes, Marco Gaertler, and Dorothea Wagner. 2003. Experiments on graph clustering algorithms. In *ESA*. Springer.

[3] Emmanuel J Candès and Benjamin Recht. 2009. Exact matrix completion via convex optimization. *FoCM* 9, 6 (2009), 717.

[4] Deepayan Chakrabarti, Ravi Kumar, and Andrew Tomkins. 2006. Evolutionary clustering. In *KDD*. ACM, 554–560.

[5] Hau Chan, Leman Akoglu, and Hanghang Tong. 2014. Make it or break it: Manipulating robustness in large networks. In *SDM*. SIAM, 325–333.

[6] Hau Chan, Shuchu Han, and Leman Akoglu. 2015. Where graph topology matters: the robust subgraph problem. In *SDM*. SIAM, 10–18.

[7] Chen Chen, Jingrui He, Nadya Bliss, and Hanghang Tong. 2017. Towards optimal connectivity on multi-layered networks. *TKDE* 29, 10 (2017), 2332–2346.

[8] Chen Chen, Ruiyue Peng, Lei Ying, and Hanghang Tong. 2018. Network Connectivity Optimization: Fundamental Limits and Effective Algorithms. In *KDD*. ACM, 1167–1176.

[9] Chen Chen, Hanghang Tong, B Prakash, Charalampos Tsourakakis, Tina Eliassi-Rad, Christos Faloutsos, and Duen Chau. 2016. Node immunization on large graphs: Theory and algorithms. *TKDE* (2016).

[10] Chen Chen, Hanghang Tong, B Aditya Prakash, Tina Eliassi-Rad, Michalis Faloutsos, and Christos Faloutsos. 2016. Eigen-optimization on large graphs by edge manipulation. *TKDD* 10, 4 (2016), 49.

[11] Jinyin Chen, Ziqiang Shi, Yangyang Wu, Xuanheng Xu, and Haibin Zheng. 2018. Link Prediction Adversarial Attack. *arXiv preprint arXiv:1810.01110* (2018).

[12] Hanjun Dai, Hui Li, Tian Tian, Xin Huang, Lin Wang, Jun Zhu, and Le Song. 2018. Adversarial Attack on Graph Structured Data. *arXiv preprint arXiv:1806.02371* (2018).

[13] Luca Donetti and Miguel A Munoz. 2004. Detecting network communities: a new systematic and efficient algorithm. *JSTAT* 2004, 10 (2004), P10012.

[14] Manuel Gomez Rodriguez, Jure Leskovec, and Andreas Krause. 2010. Inferring networks of diffusion and influence. In *KDD*. ACM, 1019–1028.

[15] Marco Gori and Augusto Pucci. 2007. ItemRank: A Random-walk Based Scoring Algorithm for Recommender Engines. In *IJCAI*. 2766–2771.

[16] Daniel Gruhl, Ramanathan Guha, David Liben-Nowell, and Andrew Tomkins. 2004. Information diffusion through blogspace. In *WWW*. ACM, 491–501.

[17] Taher H Haveliwala. 2002. Topic-sensitive pagerank. In *WWW*. ACM, 517–526.

[18] Xiaofeng He, Hongyuan Zha, Chris HQ Ding, and Horst D Simon. 2002. Web document clustering using hyperlink structures. *CSDA* 41, 1 (2002), 19–45.

[19] Yifan Hu, Yehuda Koren, and Chris Volinsky. 2008. Collaborative filtering for implicit feedback datasets. In *ICDM*. Ieee, 263–272.

[20] Jian Kang, Hanghang Tong, Yinglong Xia, and Wei Fan. 2018. AURORA: Auditing PageRank on Large Graphs. In *BigData*. IEEE.

[21] David Kempe, Jon Kleinberg, and Éva Tardos. 2003. Maximizing the spread of influence through a social network. In *KDD*. ACM, 137–146.

[22] David Kempe, Jon M Kleinberg, and Éva Tardos. 2005. Influential Nodes in a Diffusion Model for Social Networks. Springer.

[23] Jon M Kleinberg. 1999. Authoritative sources in a hyperlinked environment. *Journal of the ACM (JACM)* 46, 5 (1999), 604–632.

[24] Pang Wei Koh and Percy Liang. 2017. Understanding black-box predictions via influence functions. *arXiv preprint arXiv:1703.04730* (2017).

[25] Yehuda Koren. 2008. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *KDD*. ACM, 426–434.

[26] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix factorization techniques for recommender systems. *Computer* 8 (2009), 30–37.

[27] Jérôme Kunegis. 2013. Konect: the koblenz network collection. In *WWW*. ACM.

[28] Long T Le, Tina Eliassi-Rad, and Hanghang Tong. 2015. MET: A fast algorithm for minimizing propagation in large graphs with small eigen-gaps. In *SDM*. SIAM.

[29] Jure Leskovec and Andrej Krevl. 2014. SNAP Datasets: Stanford Large Network Dataset Collection. http://snap.stanford.edu/data.

[30] Bo Li, Yining Wang, Aarti Singh, and Yevgeniy Vorobeychik. 2016. Data poisoning attacks on factorization-based collaborative filtering. In *Advances in neural information processing systems*. 1885–1893.

[31] Liangyue Li, Yuan Yao, Jie Tang, Wei Fan, and Hanghang Tong. 2016. QUINT: on query-specific optimal networks. In *KDD*. ACM.

[32] Rong-Hua Li and Jeffrey Xu Yu. 2015. Triangle minimization in large networks. *KAIS* 45, 3 (2015), 617–643.

[33] Shike Mei and Xiaojin Zhu. 2015. Using Machine Teaching to Identify Optimal Training-Set Attacks on Machine Learners.. In *AAAI*. 2871–2877.

[34] Andrew Y Ng, Michael I Jordan, and Yair Weiss. 2002. On spectral clustering: Analysis and an algorithm. In *NIPS*. 849–856.

[35] Andrew Y Ng, Alice X Zheng, and Michael I Jordan. 2001. Link analysis, eigenvectors and stability. In *IJCAI*. 903–910.

[36] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. 1999. *The PageRank citation ranking: Bringing order to the web.* Technical Report. Stanford InfoLab.

[37] Natali Ruchansky, Mark Crovella, and Evimaria Terzi. 2015. Matrix completion with queries. In *KDD*. ACM, 1025–1034.

[38] Jianbo Shi and Jitendra Malik. 2000. Normalized cuts and image segmentation. *TPAMI* 22, 8 (2000), 888–905.

[39] Rohit Singh, Jinbo Xu, and Bonnie Berger. 2007. Pairwise global alignment of protein interaction networks by matching neighborhood topology. In *RECOMB*. Springer, 16–31.

[40] Mingjie Sun, Jian Tang, Huichen Li, Bo Li, Chaowei Xiao, Yao Chen, and Dawn Song. 2018. Data Poisoning Attack against Unsupervised Node Embedding Methods. *arXiv preprint arXiv:1810.12881* (2018).

[41] Lei Tang and Huan Liu. 2010. Graph mining applications to social network analysis. In *Managing and Mining Graph Data*. Springer, 487–513.

[42] Hanghang Tong, Christos Faloutsos, and Jia-Yu Pan. 2006. Fast random walk with restart and its applications. In *ICDM*. IEEE, 613–622.

[43] Yuan Yao, Hanghang Tong, Guo Yan, Feng Xu, Xiang Zhang, Boleslaw K Szymanski, and Jian Lu. 2014. Dual-regularized one-class collaborative filtering. In *CIKM*. ACM, 759–768.

[44] Dawei Zhou, Si Zhang, Mehmet Yigit Yildirim, Scott Alcorn, Hanghang Tong, Hasan Davulcu, and Jingrui He. 2017. A local algorithm for structure-preserving graph cut. In *KDD*. ACM, 655–664.

[45] Daniel Zügner, Amir Akbarnejad, and Stephan Günnemann. 2018. Adversarial attacks on neural networks for graph data. In *KDD*. ACM, 2847–2856.