Adversarial Attacks on Multi-Network Mining: Problem Definition and Fast Solutions*

Qinghai Zhou, Liangyue Li, Nan Cao, Lei Ying, Hanghang Tong

Abstract—Multi-sourced networks naturally appear in many application domains, ranging from bioinformatics, social networks, neuroscience to management. Although state-of-the-art offers rich models and algorithms to find various patterns when input networks are given, it has largely remained nascent on how vulnerable the mining results are due to the adversarial attacks. In this paper, we address the problem of attacking multi-network mining through the way of deliberately perturbing the networks to alter the mining results. The key idea of the proposed method (ADMIRING) is effective and efficient influence functions on the Sylvester equation defined over the input networks, which plays a central and unifying role in various multi-network mining tasks. The proposed algorithms bear three main advantages, including (1) *effectiveness,* being able to accurately quantify the rate of change of the mining results in response to attacks; (2) *efficiency,* scaling *linearly* with more than $100\times$ speed-up over the straight-forward implementation without any quality loss; and (3) *generality,* being applicable to a variety of multi-network mining tasks (e.g., graph kernel, network alignment, cross-network node similarity) with different attacking strategies (e.g., edge/node removal, attribute alteration).

Index Terms—Adversarial attack, Sylvester equation, Multi-network mining.

1 INTRODUCTION

Multi-sourced networks naturally appear in many high-impact application domains, ranging from bioinformatics, social networks, neuroscience to management. For example, for protein function prediction, a classic method is to assign similarity to protein pairs by applying graph kernel over multiple protein networks [1]. Another application is to leverage the attribute information and/or network topology to identify unique users across different networks (i.e., network alignment) [2]. For team management, [3] proposes to replace the unavailable individual in the team by recommending the best candidate who maximizes the similarity of the team networks before and after the replacement (i.e., team-context aware similarity). For financial fraud detection, [4] resorts to interactive subgraph matching to identify complex fraud schema, e.g., syntheticIDs, money laundry, etc.

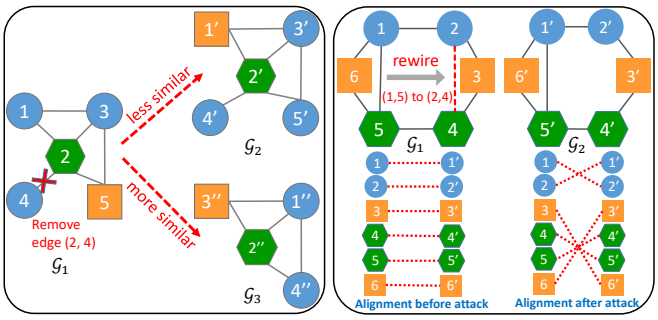
To date, many sophisticated multi-network mining models and algorithms have been proposed (See Section 5 for a review). Although these methods are quite effective in identifying various patterns when the input networks are given, less is known on how the mining results would be affected by the perturbation of the underlying networks, due to either random noise (i.e., sensitivity analysis) or malicious attacks (i.e., adversarial learning). For example, although graph kernel is effective in predicting the function (i.e., labels) of a protein, it is not clear how sensitive the prediction result is due to the measurement error for certain molecule-molecule interactions (i.e., edge error). For team management, it remains opaque on how the replacement results might be misled by the falsely claimed skills by certain users (i.e., manipulation of node attributes). For financial fraud detection, it is still largely an open question on how the fraudulent users might intentionally create some legitimate transactions to bypass the current subgraph matching based detectors.

In this paper, we address the problem of attacking multi-network mining to alter its results, which we formulate as an optimization problem. We propose a family of algorithms (ADMIRING) to achieve effective and efficient attacks. Figure 1 presents two illustrative examples of adversarial multi-network mining. The key idea behind our method is to quantitatively characterize how the mining result will change if we deliberately perturb the networks, e.g., editing the network topology by removing edges/nodes or modifying attributes. To be specific, given the central and unifying role of the Sylvester equation defined over the input networks in a variety of multi-network mining tasks (e.g., graph kernel, network alignment, cross-network node similarity, etc.) [5], [6], we measure the influence of network elements (i.e., edge, node and attribute) as the rate of change of the mining results induced by the underlying Sylvester equation. We further propose efficient algorithms to speed up and scale up the computation. We summarize the main contributions of this paper as follows,

- **Problem Formulation.** We formally define the adversarial multi-network mining problem and formulate it as an

*This paper is an extended version of an earlier paper at IEEE ICDM 2019 with the title of "ADMIRING: Adversarial Multi-Network Mining"

- *Qinghai Zhou and Hanghang Tong are with the Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL 61801 Email: {qinghai2, htong}@illinois.edu*
- *Liangyue Li is with Amazon, Palo Alto, CA Email: liliangy@amazon.com*
- *Nan Cao is with the joint appointment at both College of Design and Innovation and College of Software Engineering, Tongji University, Shanghai, China. Email: nan.cao@gmail.com*
- *Lei Ying is with the Electrical Engineering and Computer Science Department of the University of Michigan, Ann Arbor, MI 48109 Email: leiying@umich.edu*



(a) Attack graph kernel. (b) Attack network alignment.

Fig. 1: Two illustrative examples of adversarial multi-network mining. (a) Attacking graph kernel: given three input networks $\mathcal{G}_1$, $\mathcal{G}_2$ and $\mathcal{G}_3$, $\mathcal{G}_1$ is much more similar to $\mathcal{G}_2$ than $\mathcal{G}_3$. By removing edge $(2, 4)$ in $\mathcal{G}_1$, the new network $\mathcal{G}_1'$ becomes more similar to $\mathcal{G}_3$ than it is to $\mathcal{G}_2$. (b) Attacking network alignment: given two input networks $\mathcal{G}_1$ and $\mathcal{G}_2$, Node-i in $\mathcal{G}_1$ is aligned to Node-i' in $\mathcal{G}_2$ $(i = 1, ..., 6)$. If we attack edge $(1, 5)$ in $\mathcal{G}_1$ by rewiring it to $(2, 4)$, the alignment result between the new network $\mathcal{G}_1'$ and $\mathcal{G}_2$ will be completely changed.

optimization problem. The key idea is to measure the influence of network elements as the rate of change of the mining results induced by the underlying Sylvester equation.

- **Algorithms and Analysis.** We propose a family of algorithms (ADMIRING) to effectively solve the adversarial multi-network mining problem, which are applicable to a variety of multi-network mining tasks. We further propose efficient algorithms to speed up and scale up computations, with a linear complexity.
- **Empirical Evaluations.** We perform extensive experimental evaluations on real-world datasets to test the efficacy of our proposed algorithm in a variety of multi-network mining tasks. Our evaluations demonstrate that (1) *effectiveness*, the algorithms can significantly alter the similarity between networks, the accuracy of network classification and that of network alignment; (2) *efficiency*, our algorithms are able to scale linearly w.r.t. the input network size, achieving more than $100\times$ speed-up over the straight-forward implementation without any quality loss.

The rest of the paper is organized as follows. In Section 2, we define the problem of adversarial multi-network mining. Section 3 introduces our proposed algorithms. We present experimental results in Section 4, and review related work in Section 5. We conclude the paper in Section 6.

2 Problem Definitions

In this section, we formally define adversarial multi-network mining problem, after we introduce notations and preliminaries on multi-network mining as well as influence function.

2.1 Notations

Table 1 summarizes the main symbols and notations used in this paper. We use bold uppercase letters for matrices (e.g., $\mathbf{A}$), bold lowercase letters for vectors (e.g., $\mathbf{q}$) and lowercase letters for scalars (e.g., c). For matrix indexing, we use $\mathbf{A}(i, j)$ to represent the entry at the i^{th} row and the j^{th} column of matrix $\mathbf{A}$, $\mathbf{A}(i, :)$ to denote the i^{th} row of $\mathbf{A}$ and $\mathbf{A}(:, j)$ to denote the j^{th} column of $\mathbf{A}$.

In this paper, we focus on a pair of node attributed networks, represented as $\mathcal{G}_1 = \{\mathbf{A}_1, \mathbf{N}_1\}$ and $\mathcal{G}_2 = \{\mathbf{A}_2, \mathbf{N}_2\}$, where $\mathbf{A}_t$ (t= 1, 2) represents the adjacency matrices of the input networks. $\mathbf{N}_t^j = \text{diag}(\mathbf{N}_t(:, j))(t = 1, 2 \text{ and } j = 1, \ldots, d)$ represents the strength of all nodes having the j^{th} attribute in network $\mathcal{G}_t$. The uppercase bold letter $\mathbf{N}_\times$ is the combined node attribute matrix of the two networks $\mathbf{N}_\times = \sum_{j=1}^{d} \mathbf{N}_1^j \otimes \mathbf{N}_2^j$. For simplicity, we assume the input networks are (a) unweighted, (b) undirected and (c) of the same size. The generalization of the proposed method to weighted and/or directed networks of different sizes is straightforward.

Symbols	Definitions
$\mathcal{G} = \{\mathbf{A}, \mathbf{N}\}$	an attributed network
$\mathbf{A}$	adjacency matrix
$\mathbf{A}_\times$	Kronecker product of $\mathbf{A}_1$ and $\mathbf{A}_2$
$\mathbf{N}^l$	diagonal matrix of the l^{th} node attribute
$\mathbf{N}_\times$	combined node attribute matrix
$\mathbf{A}^{-1}$	inverse of matrix $\mathbf{A}$
$\mathbf{A}'$	transpose of matrix $\mathbf{A}$
$\mathbf{S}^{i,j}$	single entry matrix $\mathbf{S}^{i,j}(i, j) = 1$ and zeros elsewhere
$\mathbf{I}$	an identity matrix
c	a regularization parameter, $0 < c < 1$
α	damping factor, $0 < \alpha < 1$
$\mathbf{p}_\times, \mathbf{q}_\times$	initial and stopping probability distribution
n, m	number of nodes and edges, respectively
d	dimension of node attribute vector
$\mathcal{I}(g)$	influence function of network element g
$\otimes$	Kronecker product
$a = \text{vec}(\mathbf{A})$	vectorize a matrix $\mathbf{A}$ in column order
$\mathbf{X} = \text{mat}(\mathbf{x}, n, n)$	reshape $\mathbf{x}$ to an $n \times n$ matrix in column order
$\mathbf{Y} = \text{diag}(\mathbf{y})$	diagonalize a vector $\mathbf{y}$

TABLE 1: Symbols and Definition

2.2 Preliminaries

We briefly review (1) Sylvester equation for multi-network mining tasks, and (2) influence function for machine learning.

A – Sylvester equation for multi-network mining. A unifying cornerstone behind many multi-network mining tasks can be attributed to the Sylvester equation defined over the input networks. In detail, given two node-attributed networks $\mathcal{G}_1$ and $\mathcal{G}_2$, we have the following generalized Sylvester equation,

$$\mathbf{X} = \sum_{l=1}^{d} c\mathbf{M}_l\mathbf{X}\mathbf{T}_l' + \mathbf{B} \tag{1}$$

where c is a regularization parameter, $\mathbf{M}_l = \mathbf{N}_2^l\mathbf{A}_2, \mathbf{T}_l = \mathbf{N}_1^l\mathbf{A}_1$, and $\mathbf{B} \in \mathbb{R}^{n\times n}$ encodes the prior knowledge of the mining tasks. For instance, in network alignment [6], $\mathbf{B}$ is the preference matrix to encode anchor links; and in random walk graph kernel [7], $\mathbf{B}$ represents the initial probability distribution of the random walks on the direct product matrix. In Eq. (1), $\mathbf{X} \in \mathbb{R}^{n\times n}$ is the solution matrix.

A numerical solution of the Sylvester equation usually costs at least $O(n^3)$ in time complexity [8], and some recent works [9], [10] are able to reduce the time complexity to be linear w.r.t. the input network size. By the Kronecker product properties, we have the following equivalent linear equation of Eq. (1),

$$\mathbf{x} = c\mathbf{N}_\times \mathbf{A}_\times \mathbf{x} + \mathbf{b} \tag{2}$$

where $\mathbf{x} = \text{vec}(\mathbf{X})$, $\mathbf{b} = \text{vec}(\mathbf{B})$ and $\mathbf{A}_\times = \mathbf{A}_1 \otimes \mathbf{A}_2$. The closed-form solution of $\mathbf{x}$ is given by $\mathbf{x} = (\mathbf{I} - c\mathbf{N}_\times \mathbf{A}_\times)^{-1}\mathbf{b}$.

Remarks. For clarity of the description of the proposed algorithms and analysis, we will mainly focus on a pair of node-attributed networks. Nonetheless, the proposed algorithms can be naturally generalized to handle other scenarios. For example, both Eq. (1) and Eq. (2) can be generalized to handle edge attributes as well [6], [10]. If the node and edge attribute information is absent, Eq. (1) degenerates to $\mathbf{X} = c\mathbf{A}_2\mathbf{X}\mathbf{A}_1' + \mathbf{B}$. If there are multiple (more than two) input networks, we can organize them into a combined network by matrix direct sum $\mathcal{G} = \{\mathbf{A}, \mathbf{N}\}$, where $\mathbf{A}$ and $\mathbf{N}$ are block diagonal matrices and each diagonal block represents one input network. Then Eq. (1) is defined w.r.t. the combined network $\mathcal{G}$ (i.e., $\mathcal{G} = \mathcal{G}_1 \bigoplus \mathcal{G}_2$).

It turns out the solution matrix $\mathbf{X}$ and its vectorization $\mathbf{x}$ encodes rich information of the input networks, and therefore have been used for a variety of mining tasks. For example, the random walk based graph kernel [7] is essentially a summation of all the entries of $\mathbf{X}$ linearly weighted by the stopping probability distribution $\mathbf{q}_\times'$ of random walks on the direct product matrix; the solution matrix $\mathbf{X}$ indicates the soft node-alignment between the input networks and the specific entries indicates the similarity or proximity between two nodes across networks (i.e., cross-network node proximity) [6]; the solution matrix $\mathbf{X}$ defined over a query network and a data network can be further fed into a goodness function [4] for subgraph matching. Conceptually, we can represent all the above multi-network mining results induced by the solution matrix $\mathbf{X}$ by a function f. For example, $f(\mathbf{X}) = \mathbf{q}_\times'\text{vec}(\mathbf{X})$ for random walk graph kernel [5]; $f(\mathbf{X}) = \mathbf{X}$ for (soft) network alignment [6]; $f(\mathbf{X}) = \mathbf{X}(s,t)$ for cross-network node similarity search; and $f(\mathbf{X}) = \text{argmin}_\mathbf{M}\ g(\mathbf{M}, \mathbf{X}) = -\|\mathbf{M}\mathbf{A}\mathbf{M}' - \mathbf{A}_q\|_F^2 + a \cdot \text{trace}(\mathbf{X}\mathbf{M}') - b \cdot \|\mathbf{M}\mathbf{M}' - \mathbf{I}\|_F^2$ for subgraph matching, where $\mathbf{M}$ is the matching indicator matrix, $\mathbf{A}$ is the adjacency matrix of the data network, a and b are additional parameters [4]. Table 2 presents a summary for different choices of $f(\cdot)$ for multi-network minignt tasks.

B – Influence function for machine learning. Influence function is a powerful analytical tool from robust statistics to evaluate the dependence of the estimator on the value of the data points [11]. The seminal work by Pang et al. [12] proposes to leverage influence function to assess the effect of each training example on the performance of the machine learning system, as a key step towards explainable machine learning. Its key idea is to trace the learning model's predictions back to the input training examples. In order to identify the training examples that are most responsible to model's behavior, they use influence function in accordance with the model that reflects how the learning model's parameters are affected if a training example is perturbed by an imperceptible amount.

2.3 Problem Definition

Generally speaking, adversarial learning aims to maximally alter the learning results by manipulating a small number of the input data points. In the case of multi-network mining, it translates to a small number of network elements (e.g., edges/nodes/attributes). Given the unifying role of the solution matrix $\mathbf{X}$ of Eq. (1), we formally define the adversarial multi-network mining problem as follows,

Problem 1. Adversarial Multi-network Mining

Given: (1) two input attributed networks $\mathcal{G}_1$ and $\mathcal{G}_2$, (2) the vectorization of the solution matrix $\mathbf{X}$ of Eq. (1), (3) a function $f(\mathbf{X})$ in Table 2 underlying the corresponding mining task, (4) an integer budget k, and (5) the specific network element type (i.e., `edge` vs. `node` vs. `attribute`);

Find: a set of k most influential network elements of the specified type so that $f(\mathbf{X})$ will change most if we attack (e.g., remove or alter) those elements.

Multi-network Mining Tasks	Function $f(\cdot)$
Random walk graph kernel [5]	$f(\mathbf{X}) = \mathbf{q}_\times'\text{vec}(\mathbf{X})$
Soft network alignment [6]	$f(\mathbf{X}) = \mathbf{X}$ or $f(\mathbf{X}) = \mathbf{vec}(\mathbf{X})$
Cross-network node similarity [6]	$f(\mathbf{X}) = \mathbf{X}(s,t)$
Subgraph matching [4]	$f(\mathbf{X}) = \text{argmin}_\mathbf{M}\ g(\mathbf{M}, \mathbf{X})$

TABLE 2: Choices of functions $f(\cdot)$ w.r.t. the solution $\mathbf{X}$ of Sylvester Equation underlying various multi-network mining tasks. In cross-network node similarity, the similarity between node s in $\mathcal{G}_2$ and node t in $\mathcal{G}_1$ is $\mathbf{X}(s,t)$.

3 Algorithms and Analysis

In this section, we first formally formulate Problem 1 from the optimization perspective. Next, we derive the influence functions with respect to various network elements (e.g., edges, nodes and attributes) for multi-network mining tasks. Based on that, we propose effective and efficient algorithms which leverage such influence functions to attack multi-network mining results, together with some analysis.

3.1 Admiring Formulation

The intuition behind adversarial multi-network mining is to find a set of key network elements (e.g., edges, nodes, attributes) whose perturbation (e.g., removal, alteration) would cause the largest change of the function $f(\cdot)$ underlying a given mining task. For example, for random walk graph kernel, the goal of an adversarial attack is to significantly change the similarity (i.e., graph kernel) between two input networks by deliberately perturbing a small set of influential network elements. For network alignment, we want to maximally alter the alignment vector $\mathbf{x}$ and for cross-network node similarity, we aim to considerably change the similarity between two nodes across input networks. To be specific, let $\mathbf{X}$ be the original solution of Eq. (1) for the input networks $\mathcal{G}_1$, $\mathcal{G}_2$ and $\mathbf{X}_\mathcal{P}$ be the new solution

matrix for networks $\mathcal{G}_{1\mathcal{P}}$ and $\mathcal{G}_2$ after we perturb the network elements in set $\mathcal{P}$. The corresponding mining results are $f(\mathbf{X})$ and $f(\mathbf{X}_{\mathcal{P}})$, respectively. We formally formulate Problem 1 as the following optimization problem,

$$\begin{aligned} \underset{\mathcal{P}}{\text{argmax}}\ \Delta f &= \left(f\left(\mathbf{X}\right) - f\left(\mathbf{X}_{\mathcal{P}}\right)\right)^2 \\ \text{s.t. } |\mathcal{P}| &= k \end{aligned} \tag{3}$$

Note that for network alignment, the squared loss in the above formulation will be replaced by the L_2 norm if $f(\mathbf{X}) = \text{vec}(\mathbf{X})$ or the Frobenius norm if $f(\mathbf{X}) = \mathbf{X}$. In order to solve the above optimization problem, there are two crucial questions that need to be answered: (Q1) how to quantitatively evaluate the influence of a specific network element w.r.t. the function $f(\cdot)$ over the solution $\mathbf{X}$ of the Sylvester equation; and (Q2) how to leverage the influence function to identify a set of network elements to attack various multi-network mining tasks. In the next two subsections, we present our solutions to Q1 and Q2 according to the specific type of network elements (i.e., edges, nodes and attributes), respectively.

3.2 Network Element Influence

In order to achieve effective attacks to multi-network mining tasks, it is desirable that the change to the network structure would significantly impact the mining results (i.e., $f(\mathbf{X})$ in Table 2). For simplicity, we only consider three types of attacks, including edge removal, node removal and node attribute alteration. For clarity of the presentation, we assume the attack always happens on the first network $\mathcal{G}_1$. In order to quantify how $f(\mathbf{X})$ changes (i.e., Δf in Eq. (3)) if we perturb a specific network element, we propose to use the influence function w.r.t. the corresponding multi-network mining tasks.

Definition 1. *Edge Influence in Multi-network Mining.* For a given multi-network mining task $f(\mathbf{X})$, the influence of a specific edge (e.g., $\mathbf{A}_1\left(i,j\right)$ in $\mathcal{G}_1$) w.r.t. the mining result is defined as the derivative of $f(\mathbf{X})$ w.r.t. this edge. Formally, the edge influence is defined as $\mathcal{I}\left(\mathbf{A}_1\left(i,j\right)\right) = \frac{\partial f(\mathbf{X})}{\partial \mathbf{A}_1(i,j)}$.

Definition 2. *Node Influence in Multi-network Mining.* The node influence is defined as the summation of influences of the incident edges, i.e., $\mathcal{I}\left(\mathbf{N}_1\left(i\right)\right) = \sum_{j|\mathbf{A}_1(i,j)=1} \mathcal{I}\left(\mathbf{A}_1\left(i,j\right)\right)$

Definition 3. *Node Attribute Influence in Multi-network Mining.* For node-attributed networks, the influence of the l^{th} attribute of node i (i.e., $\mathbf{N}_1^l(i,i)$) in network $\mathcal{G}_1$ is defined as the derivate of $f(\mathbf{X})$ w.r.t. this specific node attribute, i.e., $\mathcal{I}(\mathbf{N}_1^l(i,i)) = \frac{\partial f(\mathbf{X})}{\partial \mathbf{N}_1^l(i,i)}$.

Next, we present details on how to compute the influence of network elements (e.g., edges, nodes and attributes) w.r.t. the mining results in the various tasks (e.g., random walk graph kernel, network alignment and cross-network node similarity in Table 2). For clarity, we will mainly use random walk graph kernel as an example to illustrate the mathematical details to compute different network element influences, followed by the discussion on how the computation would differ for other multi-network mining tasks.

A – Edge influence. We first give Lemma 1 to compute the edge influence for random walk graph kernel.

Lemma 1. (Edge Influence for Random Walk Graph Kernel.) Given random walk graph kernel between two input networks: $f(\mathbf{X}) = \mathbf{q}_\times' \left(\mathbf{I} - c\mathbf{N}_\times\mathbf{A}_\times\right)^{-1}\mathbf{N}_\times\mathbf{p}_\times$, the influence of a specific edge (e.g., $\mathbf{A}_1(i,j)$ in $\mathcal{G}_1$) w.r.t. random walk graph kernel can be calculated as follows,

$$\mathcal{I}\left(\mathbf{A}_1\left(i,j\right)\right) = c\mathbf{q}_\times{}'\mathbf{Q}\mathbf{N}_\times[(\mathbf{S}^{i,j} + \mathbf{S}^{j,i}) \otimes \mathbf{A}_2]\mathbf{Q}\mathbf{N}_\times\mathbf{p}_\times \tag{4}$$

where $\mathbf{Q} = \left(\mathbf{I} - c\,\mathbf{N}_\times\mathbf{A}_\times\right)^{-1}$, $\mathbf{S}^{i,j}$ is a single-entry matrix of the same size as $\mathbf{A}_1$, with 1 at the $(i,j)^{\text{th}}$ position and 0 elsewhere.

Proof. According to [5], the random walk graph kernel for node-attributed networks is,

$$f(\mathbf{X}) = \mathbf{q}_\times'\text{vec}(\mathbf{X}) = \mathbf{q}_\times'\left(\mathbf{I} - c\mathbf{N}_\times\mathbf{A}_\times\right)^{-1}\mathbf{N}_\times\mathbf{p}_\times \tag{5}$$

where $\text{vec}(\mathbf{X}) = \left(\mathbf{I} - c\mathbf{N}_\times\mathbf{A}_\times\right)^{-1}\mathbf{N}_\times\mathbf{p}_\times = \mathbf{x}$.

Following Definition 1, we take the partial derivative of $f(X)$ w.r.t. a specific edge (e.g., $\mathbf{A}_1\left(i,j\right)$ in network $\mathcal{G}_1$),

$$\mathcal{I}(\mathbf{A}_1(i,j)) = \frac{\partial f\left(\mathbf{X}\right)}{\partial \mathbf{A}_1\left(i,j\right)} = \frac{\partial f(\mathbf{X})}{\partial \mathbf{x}}\frac{\partial \mathbf{x}}{\partial \mathbf{A}_1(i,j)} \tag{6}$$

According to the first row of Table 2, we have that $\frac{\partial f(\mathbf{X})}{\partial \mathbf{x}} = \mathbf{q}_\times'$. For the second partial derivative (i.e., $\frac{\partial \mathbf{x}}{\partial \mathbf{A}_1(i,j)}$), by taking the derivative of Eq. (2), we have that

$$\frac{\partial \mathbf{x}}{\mathbf{A}_1(i,j)} = c\mathbf{N}_\times\frac{\partial \mathbf{A}_\times}{\partial \mathbf{A}_1(i,j)}\mathbf{x} + c\mathbf{N}_\times\mathbf{A}_\times\frac{\partial \mathbf{x}}{\partial \mathbf{A}_1(i,j)} \tag{7}$$

From Eq. (7), we then have that

$$\frac{\partial \mathbf{x}}{\mathbf{A}_1(i,j)} = c\left(\mathbf{I} - c\mathbf{N}_\times\mathbf{A}_\times\right)^{-1}\mathbf{N}_\times\frac{\partial \mathbf{A}_\times}{\partial \mathbf{A}_1(i,j)}\mathbf{x} \tag{8}$$

By the property of the matrix derivative [13, Page 8], we further have that

$$\frac{\partial \mathbf{A}_\times}{\mathbf{A}_1(i,j)} = \frac{\partial \mathbf{A}_1}{\mathbf{A}_1(i,j)} \otimes \mathbf{A}_2 = \left(\mathbf{S}^{i,j} + \mathbf{S}^{j,i}\right) \otimes \mathbf{A}_2 \tag{9}$$

Recall the closed-form solution $\mathbf{x} = \left(\mathbf{I} - c\mathbf{N}_\times\mathbf{A}_\times\right)^{-1}\mathbf{N}_\times\mathbf{p}_\times$. Putting everything together, we obtain the solution for calculating the influence of edge $\mathbf{A}_1\left(i,j\right)$ w.r.t. random walk graph kernel as follows,

$$\mathcal{I}\left(\mathbf{A}_1\left(i,j\right)\right) = c\mathbf{q}_\times{}'\mathbf{Q}\mathbf{N}_\times[(\mathbf{S}^{i,j} + \mathbf{S}^{j,i}) \otimes \mathbf{A}_2]\mathbf{Q}\mathbf{N}_\times\mathbf{p}_\times$$

which completes the proof. □

B – Node influence. Based on the edge influence (Eq. (4)), it is straight-forward to compute the node influence, which is summarized in the following proposition.

Proposition 1. (Node Influence in Random Walk Graph Kernel.) Given random walk graph kernel between two input networks: $f(\mathbf{X}) = \mathbf{q}_\times'\left(\mathbf{I} - c\mathbf{N}_\times\mathbf{A}_\times\right)^{-1}\mathbf{N}_\times\mathbf{p}_\times$, the influence of a specific node (e.g., $\mathbf{N}_1(i)$ in $\mathcal{G}_1$) w.r.t. random walk graph kernel can be calculated as,

$$\mathcal{I}\left(\mathbf{N}_1\left(i\right)\right) = c\mathbf{q}_\times{}'\mathbf{Q}\mathbf{N}_\times[\sum_{j|\mathbf{A}_1(i,j)=1}(\mathbf{S}^{i,j}+\mathbf{S}^{j,i})\otimes\mathbf{A}_2]\mathbf{Q}\mathbf{N}_\times\mathbf{p}_\times \tag{10}$$

Proof. It directly follows Lemma 1. Omitted for brevity. □

C – Node attribute influence. Finally, we give Lemma 2 to compute the node attribute influence.

Lemma 2. (Node Attribute Influence for Random Walk Graph Kernel.) Given random walk graph kernel between two input networks: $f(\mathbf{X}) = \mathbf{q}'_\times \left(\mathbf{I} - c\mathbf{N}_\times \mathbf{A}_\times\right)^{-1} \mathbf{N}_\times \mathbf{p}_\times$, the influence of a specific node attribute (e.g., the l^{th} dimension of the attribute vector of node $\mathbf{N}_1(i)$ in $\mathcal{G}_1$, i.e., $\mathbf{N}_1^l(i,i)$) w.r.t. random walk graph kernel can be calculated as follows,

$$\mathcal{I}\left(\mathbf{N}_1^l(i,i)\right) = \mathbf{q}'_\times \mathbf{Q}[\mathbf{S}^{i,i} \otimes \mathbf{N}_2^l]\left(\mathbf{I} + c\mathbf{A}_\times \mathbf{Q}\mathbf{N}_\times\right)\mathbf{p}_\times \quad (11)$$

where $\mathbf{Q} = \left(\mathbf{I} - c\,\mathbf{N}_\times \mathbf{A}_\times\right)^{-1}$, $\mathbf{S}^{i,i}$ is a single-entry matrix of the same size as $\mathbf{A}_1$, with 1 at the $(i,i)^{\text{th}}$ position and 0 elsewhere. $\mathbf{N}_2^l$ represents the strength of all the nodes having the l^{th} attribute from the network $\mathcal{G}_2$.

Proof. Recall that $\mathbf{N}_\times = \sum_{j=1}^{d} \mathbf{N}_1^j \otimes \mathbf{N}_2^j$ is the combined node attribute matrix. According to Definition 3, we restrict the node attribute influence to be the l-th ($1 \leqslant l \leqslant d$) attribute of node i in $\mathcal{G}_1$, i.e., $\mathbf{N}_1^l(i,i)$. Following a similar derivation as the edge influence, we can compute the node attribute influence as

$$\mathcal{I}\left(\mathbf{N}_1^l(i,i)\right) = \frac{\partial f(\mathbf{X})}{\partial \mathbf{N}_1^l(i,i)} = \frac{\partial f(\mathbf{X})}{\partial \mathbf{x}} \frac{\partial \mathbf{x}}{\partial \mathbf{N}_1^l(i,i)} \quad (12)$$

To calculate the second derivative of Eq. (12), we take the derivative of Eq. (2) on both sides w.r.t $\mathbf{N}_1^l(i,i)$, where $\mathbf{b} = \mathbf{N}_\times \mathbf{p}_\times$. We have that

$$\frac{\partial \mathbf{x}}{\partial \mathbf{N}_1^l(i,i)} = c\frac{\partial \mathbf{N}_\times}{\partial \mathbf{N}_1^l(i,i)}\mathbf{A}_\times \mathbf{x} + c\mathbf{N}_\times \mathbf{A}_\times \frac{\partial \mathbf{x}}{\partial \mathbf{N}_1^l(i,i)} + \frac{\partial \mathbf{N}_\times}{\partial \mathbf{N}_1^l(i,i)}\mathbf{p}_\times$$

$$\Leftrightarrow \quad \frac{\partial \mathbf{x}}{\partial \mathbf{N}_1^l(i,i)} = \left(\mathbf{I} - c\mathbf{N}_\times \mathbf{A}_\times\right)^{-1} \frac{\partial \mathbf{N}_\times}{\partial \mathbf{N}_1^l(i,i)}\left(c\mathbf{A}_\times \mathbf{x} + \mathbf{p}_\times\right)$$

where $\mathbf{x} = \left(\mathbf{I} - c\mathbf{N}_\times \mathbf{A}_\times\right)^{-1} \mathbf{N}_\times \mathbf{p}_\times$.

Since $\mathbf{N}_\times = \sum_{j=1}^{d} \mathbf{N}_1^j \otimes \mathbf{N}_2^j$, by the property of the derivative of matrix [13, Page 8], we have that

$$\frac{\partial \mathbf{N}_\times}{\partial \mathbf{N}_1^l(i,i)} = \frac{\partial \mathbf{N}_1^l}{\partial \mathbf{N}_1^l(i,i)} \otimes \mathbf{N}_2^l = \mathbf{S}^{i,i} \otimes \mathbf{N}_2^l$$

Putting everything together, we have the closed-form solution $\mathcal{I}\left(\mathbf{N}_1^l(i,i)\right)$ as follows,

$$\mathcal{I}\left(\mathbf{N}_1^l(i,i)\right) = \mathbf{q}'_\times \mathbf{Q}[\mathbf{S}^{i,i} \otimes \mathbf{N}_2^l]\left(\mathbf{I} + c\mathbf{A}_\times \mathbf{Q}\mathbf{N}_\times\right)\mathbf{p}_\times$$

which completes the proof. □

D – Influence functions for other mining tasks. Lemma 1, Proposition 1 and Lemma 2 provide accurate ways to compute edge, node and attribute influence for random walk graph kernel, respectively. We can use a very similar procedure to compute influence for other multi-network mining tasks in Table 2. Let us take edge influence as an example, since node influence directly follows the edge influence and attribute influence can be computed in a similar way as the edge influence. For network alignment, since $f(\mathbf{X}) = \text{vec}(\mathbf{X}) = \mathbf{x}$, we take the partial derivative of the squared L_2 norm of $\mathbf{x}$ w.r.t. $\mathbf{A}_1(i,j)$, i.e., $\frac{\partial \|\mathbf{x}\|_2^2}{\partial \mathbf{A}_1(i,j)}$ as the influence of a given edge $\mathbf{A}_1(i,j)$. Likewise, for the cross-network node similarity between node-s in $\mathcal{G}_1$ and node-t in $\mathcal{G}_2$, the influence of a given edge $\mathbf{A}(i,j)$ is the $((s-1)n+t)^{\text{th}}$entry of $\frac{\partial \mathbf{x}}{\partial \mathbf{A}_1(i,j)}$ in Eq. (8). For subgraph matching, it involves another level of optimization over the matching indicator matrix $\mathbf{M}$ which is non-differential due to the binary constraint. Nonetheless, it is reasonable to expect the matching indicator matrix $\mathbf{M}$ to be highly dependent on a few top-ranked entries in the solution vector $\mathbf{x}$. This suggests to adopt the ordered weighted L_1 norm (OWL) [14] of $\frac{\partial \mathbf{x}}{\mathbf{A}_1(i,j)}$ in Eq. (8) as the influence of a given edge $\mathbf{A}_1(i,j)$.

3.3 Proposed Algorithms

A – A generic algorithm for adversarial multi-network mining. Based on Lemma 1, 2 and proposition 1, we propose Algorithm 1 to identify the most influential network elements (i.e., edges, nodes, attributes) for random walk graph kernel. Note that Algorithm 1 provides a family of attacking algorithms based on the specific network element. We use different suffix to differentiate different attacking scenarios, i.e., ADMIRING-E, ADMIRING-N, ADMIRING-A for attacking edges, nodes and node attributes respectively. The proposed algorithms for other tasks in Table 2 follow the same procedure as Algorithm 1 except for the specific ways for computing influence functions as outlined in the previous subsection.

The key idea of the proposed ADMIRING algorithm is that we iteratively attack one network element with the highest influence value (Step-5, Step-9, Step-16), remove or alter it from the network (Step-6, Step-10, Step-17) and recompute the influence functions for the remaining network elements. In Algorithm 1, the two column vectors $\mathbf{p}_\times$ and $\mathbf{q}_\times$ are set as uniform unit vectors, i.e., $\mathbf{p}_\times = \mathbf{q}_\times = \mathbf{1} \times \frac{1}{n^2}$, and c is chosen as a small positive number to ensure convergence of the Sylvester Eq. (1) with fixed-point methods (e.g., $c = 1/(\max(\text{eigenvalue}(\mathbf{N}_1\mathbf{A}_1)) \times \max(\text{eigenvalue}(\mathbf{N}_2\mathbf{A}_2)) + 1)$).

B – Speed-up and Scale-up. The main computational bottleneck of Algorithm 1 is that in each iteration we need to (re-)compute the influence for each existing network element. For example, with fixed-point methods to solve Eq. (1), the time complexity of Algorithm 1 is at least $O(k(m^2 + n^2 + mn))$ and its space complexity is at least $O(m^2 + n^2)$.)

In order to address this issue, we propose a fast and exact solution to speed up the computation of influence function. Again, we take edge influence as an example, since the node influence directly follows the edge influence, and we can develop a very similar fast algorithm for attribute influence.

Our proposed fast solution is based on two key ideas. First, we observe that there are lots of overlaps in terms of computing the influence of different edges. That is, $\mathbf{q}'_\times \mathbf{Q}\mathbf{N}_\times$ term and $\mathbf{Q}\mathbf{N}_\times \mathbf{p}_\times$ term in Eq. (4) are the same for different edges, and each of these two terms correspond to the solution vector of Eq. (2) with different bias vector $\mathbf{b}$ respectively. Second, some recent progress suggests that Sylvester equation in Eq. (1) can be solved in linear time *without* quality loss [10], as shown in the following proposition:

Proposition 2. (Kronecker Krylov subspace based Sylvester equation solver.) By implicit Kronecker Krylov subspace method [10], Eq. (1) can be solved in $O(m)$ time, where m is the number of edges in input networks, and its solution matrix $\mathbf{X}$ can be represented in low-rank form, i.e., $\mathbf{X} = \mathbf{U}\mathbf{V}'$ where $\mathbf{U}, \mathbf{V} \in \mathbb{R}^{n \times r}$ are two low-rank matrices.

Algorithm 1 ADMIRING: Adversarial Multi-network Mining

Input: (1) Two attributed networks $\mathcal{G}_1$ and $\mathcal{G}_2$, (2) an integer budget k, (3) a mining task denoted by $f(\cdot)$ in Table 2, (4) network element type (i.e., `edge` vs `node` vs `node attribute`), (5) $\mathbf{q}_\times$, $\mathbf{p}_\times$, and (6) parameter c;
Output: A set of k network elements $\mathcal{P}$ to attack, and the residual network $\mathcal{G}_{1_\mathcal{P}}$.

Initialize $\mathcal{P} = \varnothing$;
while $|\mathcal{P}| < k$ **do**
  if element type is `edge` **then**
    Calculate influence for all edges using Eq. (4)
    Add edge $(i,j) = \underset{(i,j)}{\operatorname{argmax}} \mathcal{I}(\mathbf{A}_1(i,j))$ to $\mathcal{P}$;
    Remove edge (i,j) and (j,i) from $\mathcal{G}_1$;
  else if element type is `node` **then**
    Calculate influence for all nodes in $\mathcal{G}_1$ using Eq. (10);
    Add node $i = \underset{i}{\operatorname{argmax}} \mathcal{I}(\mathbf{N}(i))$ to $\mathcal{P}$;
    Remove node i from $\mathcal{G}_1$;
  else if element type is `node attribute` **then**
    Set the value of damping factor $\alpha \in (0,1)$;
    for node i in $\mathcal{G}_1$ **do**
      Calculate influence of each attribute using Eq. (11);
    end for
    Add node attribute $\mathbf{N}_1^l(i,i) = \underset{\mathbf{N}_1^l(i,i)}{\operatorname{argmax}} \mathcal{I}(\mathbf{N}_1^l(i,i))$ to $\mathcal{P}$;
    Reduce the selected attribute by a ratio α in $\mathcal{G}_1$;
  else
    return Error
  end if
end while
return $\mathcal{P}$ and $\mathcal{G}_{1_\mathcal{P}}$.

Based on Proposition 2, we propose ADMIRING-E-Fast to speedup the computation of edge influence in Algorithm 2. In Step-1, since $\mathbf{q}_\times$ is a unit vector, we can reshape it into rank-1 matrix, i.e., $\mathbf{Q}_\times = \frac{1}{n^2}\mathbf{1}_{n\times 1}\mathbf{1}_{1\times n}$. We solve two Sylvester equations in Step-2 and Step-3 by Kronecker Krylov subspace method and save low-rank representations of the solution matrices, respectively. Then, the influence of all edges can be computed by Step-8.

We give the following Theorem, which states that the edge influences computed by the proposed ADMIRING-E-Fast is exactly the same as those computed by Eq. (4).

Theorem 1. (Correctness of ADMIRING-E-Fast.) Given two input networks, $\mathcal{G}_1$ and $\mathcal{G}_2$, the influence of edges in $\mathcal{G}_1$ to random walk graph kernel computed from ADMIRING-E-Fast is the same as it is calculated by Eq. (4).

Proof. Recall that the influence of edge (i,j) in $\mathcal{G}_1$ to random walk graph kernel is computed as,

$$\mathcal{I}(\mathbf{A}_1(i,j)) = c\mathbf{q}_\times{}'\mathbf{Q}\mathbf{N}_\times[(\mathbf{S}^{i,j} + \mathbf{S}^{j,i}) \otimes \mathbf{A}_2]\mathbf{Q}\mathbf{N}_\times\mathbf{p}_\times$$

We denote $\mathbf{x} = \mathbf{Q}\mathbf{N}_\times\mathbf{p}_\times$, $\mathbf{X} = \mathrm{mat}(\mathbf{x}, n, n)$, $\mathbf{y} = [\mathbf{q}_\times'\mathbf{Q}\mathbf{N}_\times]' = \mathbf{N}_\times\mathbf{Q}'\mathbf{q}_\times$ and $\mathbf{Y} = \mathrm{mat}(\mathbf{y}, n, n)$. We have,

$$\begin{aligned}\mathbf{N}_\times\mathbf{p}_\times &= [\sum_{i=1}^{d}\mathbf{N}_1^i \otimes \mathbf{N}_2^i]\cdot[\mathbf{p}_1 \otimes \mathbf{p}_2] \\ &= \sum_{i=1}^{d}[\mathbf{N}_1^i\cdot\mathbf{p}_1] \otimes [\mathbf{N}_2^i\cdot\mathbf{p}_2] = \sum_{i}^{d}\widetilde{\mathbf{p}}_{1,i} \otimes \widetilde{\mathbf{p}}_{2,i}\end{aligned}$$

where $\widetilde{\mathbf{p}}_{1,i} = \mathbf{N}_1^i\cdot\mathbf{p}_1$ and $\widetilde{\mathbf{p}}_{2,i} = \mathbf{N}_2^i\cdot\mathbf{p}_2$ ($i = 1,\ldots,d$). We then reshape $\mathbf{N}_\times\mathbf{p}_\times$ as,

$$\widetilde{\mathbf{P}}_\times = \mathrm{mat}(\mathbf{N}_\times\mathbf{p}_\times, n, n) = \begin{bmatrix}\widetilde{\mathbf{p}}_{2,1} & \cdots & \widetilde{\mathbf{p}}_{2,d}\end{bmatrix}\begin{bmatrix}\widetilde{\mathbf{p}}_{1,1}' \\ \vdots \\ \widetilde{\mathbf{p}}_{1,d}'\end{bmatrix}$$

and $\widetilde{\mathbf{P}}_\times$ is in low-rank. Therefore, based on Proposition 2, by solving the linear system at Step-2 in Algorithm 2, we have an implicit representation of $\mathbf{X} = \mathbf{U}_1\mathbf{V}_1$, where $\mathbf{U}_1, \mathbf{V}_1 \in \mathbb{R}^{n\times r}$.

We solve the linear system in Step-3 for an implicit representation for $\widetilde{\mathbf{X}} = \mathrm{mat}(\mathbf{Q}'\mathbf{q}_\times, n, n) = \mathbf{F}\mathbf{G}'$, where $\mathbf{F}, \mathbf{G} \in \mathbb{R}^{n\times r'}$ because $\mathbf{Q}_\times = \mathrm{mat}(\mathbf{q}_\times, n, n)$ is a rank-1 matrix. By vectorizing $\widetilde{\mathbf{X}}$, we get $\mathrm{vec}(\widetilde{\mathbf{X}}) = \sum_{j=1}^{r'}\mathbf{g}_j \otimes \mathbf{f}_j$, where $\mathbf{g}_j, \mathbf{f}_j$ are the j^{th} column of $\mathbf{G}$ and $\mathbf{F}$, respectively. We further have,

$$\begin{aligned}\mathbf{N}_\times\mathrm{vec}(\widetilde{\mathbf{X}}) &= [\sum_{i=1}^{d}\mathbf{N}_1^i \otimes \mathbf{N}_2^i][\sum_{j=1}^{r'}\mathbf{g}_j \otimes \mathbf{f}_j] \\ &= \sum_{i=1}^{d}\sum_{j=1}^{r'}(\mathbf{N}_1^i\mathbf{g}_j) \otimes (\mathbf{N}_2^i\mathbf{f}_j) = \sum_{i=1}^{d}\sum_{j=1}^{r'}\widetilde{\mathbf{g}}_{i,j} \otimes \widetilde{\mathbf{f}}_{i,j}\end{aligned}$$

where $\widetilde{\mathbf{g}}_{i,j} = \mathbf{N}_1^i\mathbf{g}_j$ and $\widetilde{\mathbf{f}}_{i,j} = \mathbf{N}_2^i\mathbf{f}_j$. After reshaping, we get

Algorithm 2 ADMIRING-E-Fast

Input: (1) Two attributed networks $\mathcal{G}_1$ and $\mathcal{G}_2$, (2) $\mathbf{q}_\times$, $\mathbf{p}_\times$, and (3) the parameter c;
Output: Influence of all edges in $\mathcal{G}_1$;

1: Reshape $\widetilde{\mathbf{P}}_\times = \mathrm{mat}(\mathbf{N}_\times\mathbf{p}_\times, n, n)$, $\mathbf{Q}_\times = \mathrm{mat}(\mathbf{q}_\times, n, n)$
2: Solve the linear system $\mathbf{X} = \sum_{l=1}^{d} c\mathbf{N}_2^l\mathbf{A}_2\mathbf{X}(\mathbf{N}_1^l\mathbf{A}_1)' + \widetilde{\mathbf{P}}_\times$ and save its implicit representation $\mathbf{X} = \mathbf{U}_1\mathbf{V}_1'$, where $\mathbf{U}_1, \mathbf{V}_1 \in \mathbb{R}^{n\times r}$;
3: Solve the linear system $\widetilde{\mathbf{X}} = \sum_{l=1}^{d} c\mathbf{A}_2\mathbf{N}_2^l\widetilde{\mathbf{X}}(\mathbf{A}_1\mathbf{N}_1^l)' + \mathbf{Q}_\times$ and save its implicit representation $\widetilde{\mathbf{X}} = \mathbf{F}\mathbf{G}' = [\mathbf{f}_1\ \mathbf{f}_2\ \ldots\mathbf{f}_{r'}][\mathbf{g}_1\ \mathbf{g}_2\ldots\mathbf{g}_{r'}]'$, where $\mathbf{f}_i, \mathbf{g}_i (i = 1,\ldots,r') \in \mathbb{R}^{n\times 1}$ are the i^{th} column of $\mathbf{F}$ and $\mathbf{G}$, respectively;
4: Vectorize $\widetilde{\mathbf{X}}$ as $\mathrm{vec}(\widetilde{\mathbf{X}}) = \sum_{i=1}^{r'}\mathbf{g}_i \otimes \mathbf{f}_i$;
5: Represent $\mathbf{Y} = \mathrm{mat}(\mathbf{N}_\times\mathrm{vec}(\widetilde{\mathbf{X}}), n, n)$ as $\mathbf{U}_2\mathbf{V}_2'$, where $\mathbf{U}_2 = [\widetilde{\mathbf{f}}_{1,1}\ \widetilde{\mathbf{f}}_{1,2}\ldots\widetilde{\mathbf{f}}_{d,r'-1}\ \widetilde{\mathbf{f}}_{d,r'}]$, $\mathbf{V}_2 = [\widetilde{\mathbf{g}}_{1,1}\ \widetilde{\mathbf{g}}_{1,2}\ldots\widetilde{\mathbf{g}}_{d,r'-1}\widetilde{\mathbf{g}}_{d,r'}]$, $\widetilde{\mathbf{f}}_{i,j} = \mathbf{N}_2^j\mathbf{f}_i$ and $\widetilde{\mathbf{g}}_{i,j} = \mathbf{N}_1^j\mathbf{g}_i$ for $i = 1,\ldots,r'$ and $j = 1,\ldots,d$.
6: Compute and save $\mathbf{P} = \mathbf{U}_2'\mathbf{A}_2\mathbf{U}_1$;
7: **for** edge(i,j) in $\mathcal{G}_1$ **do**
8: Calculate the influence as, $\mathcal{I}(\mathbf{A}_1(i,j)) = \mathbf{V}_2(i,:)\mathbf{P}\mathbf{V}_1'(:,j) + \mathbf{V}_2(j,:)\mathbf{P}\mathbf{V}_1'(:,i)$;
9: **end for**
10: **return** Influences of all edges in $\mathcal{G}_1$;

$$\mathbf{Y} = \mathrm{mat}(\mathbf{N}_\times \mathrm{vec}(\widetilde{\mathbf{X}}), n, n) = \overbrace{\begin{bmatrix}\widetilde{\mathbf{f}}_{1,1} & \widetilde{\mathbf{f}}_{1,2} & \dots & \widetilde{\mathbf{f}}_{d,r'}\end{bmatrix}}^{\mathbf{U}_2 \in \mathbb{R}^{n \times r'd}} \underbrace{\begin{bmatrix}\widetilde{\mathbf{g}}'_{1,1} \\ \widetilde{\mathbf{g}}'_{1,2} \\ \vdots \\ \widetilde{\mathbf{g}}'_{d,r'}\end{bmatrix}}_{\mathbf{V}'_2 \in \mathbb{R}^{r'd \times n}}$$

Then Eq. (4) can be re-written as,

$$\mathcal{I}(\mathbf{A}_1(i,j)) = c \cdot (\mathrm{vec}(\mathbf{U}_2\mathbf{V}'_2))'[(\mathbf{S}^{i,j} + \mathbf{S}^{j,i}) \otimes \mathbf{A}_2]\mathrm{vec}(\mathbf{U}_1\mathbf{V}'_1)$$

Since $(\mathbf{S}^{i,j} + \mathbf{S}^{j,i}) \otimes \mathbf{A}_2$ is a sparse matrix with $\mathbf{A}_2$ at its $(i,j)^{\text{th}}$ and $(j,i)^{\text{th}}$ position if it is represented as a block matrix, the computation of the influence of edge (i,j) can be further simplified as,

$$\mathcal{I}(\mathbf{A}_1(i,j)) = c\mathbf{V}_2(i,:)\mathbf{P}\mathbf{V}'_1(:,j) + c\mathbf{V}_2(j,:)\mathbf{P}\mathbf{V}'_1(:,i)$$

where $\mathbf{P} = \mathbf{U}'_2\mathbf{A}_2\mathbf{U}_1$ and we only compute $\mathbf{P}$ for once. This completes the proof. □

With ADMIRING-E-Fast, we can reduce the complexity of ADMIRING-E to be linear with the number of edges and nodes in networks (i.e., m, n respectively) in both time and space, which is summarized in Lemma 4.

Lemma 3. (Time and space complexity of ADMIRING-E, ADMIRING-N and ADMIRING-A). The time and space complexity of ADMIRING-E are $O(k(m^2 + n^2 + mn))$ and $O(n^2 + m)$, respectively. The time and space complexity of ADMIRING-N are $O(k(m^2 + n^2 + d_{max}mn + d_{max}n^2))$ and $O(n^2+m^2)$ (d_{max} is the largest node degree in the network), respectively. The time and space complexity of ADMIRING-A are $O(k(m^2 + n^2 + dn^2))$ and $O(n^2 + m^2)$, respectively.

Proof. **a –** In ADMIRING-E, for each of k iterations, we need to first compute two vectors, $\mathbf{q}_\times{}'\mathbf{Q}\mathbf{N}_\times$ and $\mathbf{Q}\mathbf{N}_\times\mathbf{p}_\times$ each with $O(m^2 + n^2)$ time complexity as mentioned in Subsection 3.2. With the two vectors computed, the complexity to calculate influence w.r.t. one specific edge is $O(m + n)$ because of the sparse structure of $\mathbf{S}^{i,j} \otimes \mathbf{A}_2$ with only $O(m)$ elements in it. Therefore, to compute the influences of all edges in one network costs $O(m(m+n)) = O(m^2 + mn)$. Overall, the time complexity of ADMIRING-E is $O(k(m^2 + n^2 + mn))$ and $O(m^2 + n^2)$ space is required to save the $n^2 \times 1$ vectors and the kronecker product of two adjacency matrices of $O(m^2)$ edges.

$$\mathcal{I}(\mathbf{A}_1(i,j)) = \overbrace{c\underbrace{\mathbf{q}_\times{}'\mathbf{Q}\mathbf{N}_\times}_{O(m^2+n^2)}[(\mathbf{S}^{i,j} + \mathbf{S}^{j,i}) \otimes \mathbf{A}_2]\underbrace{\mathbf{Q}\mathbf{N}_\times\mathbf{p}_\times}_{O(m^2+n^2)}}^{O(m+n)\text{with two vectors pre-computed}}$$

b – For ADMIRING-N, by firstly computing the two vectors ($O(m^2 + n^2)$), $\mathbf{q}_\times{}'\mathbf{Q}\mathbf{N}_\times$ and $\mathbf{Q}\mathbf{N}_\times\mathbf{p}_\times$, the time complexity to calculate influences of all nodes costs $O(n(d_{max}m + d_{max}n))$ since there are $O(d_{max}m)$ entries in $\sum_{j|\mathbf{A}_1(i,j)=1}(\mathbf{S}^{i,j}+\mathbf{S}^{j,i})\otimes\mathbf{A}_2$. Therefore, the time complexity of ADMIRING-N is $O(k(m^2 + n^2 + d_{max}mn + d_{max}n^2))$ and the space complexity is $O(m^2 + n^2)$.

c - In ADMIRING-A, it costs $O(m^2+n^2)$ to compute two vectors, $\mathbf{q}'_\times\mathbf{Q}$ and $(\mathbf{I} + c\mathbf{A}_\times\mathbf{Q}\mathbf{N}_\times)\,\mathbf{p}_\times$, and $O(nd(n))$ to iteratively calculate influence of all node attributes ($\mathbf{S}^{i,i} \otimes \mathbf{N}_2^l$ has $O(n)$ elements). Therefore, we can get the time and space complexity of ADMIRING-A, which are $O(k(m^2+n^2+dn^2))$ and $O(n^2 + m^2)$, respectively. □

Lemma 4. (Complexity of ADMIRING-E-Fast.) The time complexity of ADMIRING-E-Fast is $O(kdrr'(m+n))$ and the space complexity is $O(m + nr + nr'd)$ where n, m are the number of nodes and edges of the network respectively, r is the dimension of the low-rank matrices $\mathbf{U}_1$ and $\mathbf{U}_2$, r' is the dimension of the implicit representation matrices, $\mathbf{F}$ and $\mathbf{G}$ in Algorithm 2, k is the size of $\mathcal{P}$ as mentioned in ADMIRING formulation, and d is the dimension of node attribute vector.

Proof. In ADMIRING-E-Fast, for each one of k iterations, it first takes $O(m)$ time to get the implicit representation of $\mathbf{X} = \mathbf{U}_1\mathbf{V}'_1$ and $\mathbf{Y} = \mathbf{U}_2\mathbf{V}'_2$. Then we first compute $\mathbf{P} = \mathbf{U}'_2\mathbf{A}_2\mathbf{U}_1$, since there are m edges in $\mathbf{A}_2$, we need $O(mr'd)$ time to compute $\mathbf{U}'_2\mathbf{A}_2$; it takes another $O(nrr'd)$ to compute the matrix multiplication of the previous result and $\mathbf{U}_1$. After we have $\mathbf{P}$, for every edge in $\mathbf{A}_1$, we need $O(drr')$ time to compute the influence for this specific edge. By adding them together and ignore the low-order terms, the final time complexity of ADMIRING-E-Fast is $O(kdrr'(m + n))$. The space we need to save $\mathbf{U}_1, \mathbf{V}_1 \in \mathbb{R}^{n\times r}$, $\mathbf{U}_2, \mathbf{V}_2 \in \mathbb{R}^{n\times r'd}$, $\mathbf{P} \in \mathbb{R}^{r'd\times r}$ and $\mathbf{A}_2$ are $O(nr)$, $O(nr'd)$, $O(drr')$ and $O(m)$, respectively. Therefore the space complexity for ADMIRING-E-Fast is $O(m + nr + nr'd)$. □

4 EXPERIMENTAL EVALUATION

In this section, we conduct experiments to evaluate the proposed algorithms in terms of the following two questions:

- **Effectiveness.** How effective are the proposed algorithms in identifying key network elements to attack multi-network mining tasks?
- **Efficiency.** How scalable and efficient are the proposed ADMIRING algorithms?

4.1 Experimental Setup

Dataset	Class	Avg. #nodes	Avg. #edges	Attribute
MUTAGENICITY	2	30.32	19.79	0
ENZYMES	6	32.63	62.14	18
PROTEINS	2	39.06	72.82	1
DHFR	2	42.43	44.54	3
REDDIT-BINARY	2	429.63	197.75	0
IMDB-BINARY	2	19.77	96.53	0
IMDB-MULTI	3	13	65.94	0
COLLAB	3	74.49	2,457.78	0
ACM Citation	-	9,872	79,122	17
DBLP Citation	-	9,916	89,616	17

TABLE 3: Statistics of datasets. The 'Class' column is the number of network labels, and '-' means such labels are unavailable. '0' in the 'Attribute' column means the corresponding networks do not have node attributes.

A – Datasets. We use ten real-world datasets, which are publicly available. Table 3 summarizes the statistics of these datasets. The first four datasets are bioinformatics networks, and the remaining six are social and collaboration networks. The detailed descriptions of these datasets are as follows.

- **MUTAGENICITY** is a dataset of 4,337 networks of molecular structures and it is divided into two classes *mutagen* (2,401 networks) and *nonmutagen* (1,936 networks) according to whether they have a property of mutagenicity [15].
- **PROTEINS** is a dataset of 1,113 protein structures [1]. Each protein is represented by a network. The task is to classify the protein structures into enzymes vs. non-enzymes.
- **ENZYMES** is a dataset of protein tertiary structures consisting of 600 enzymes [1]. This dataset includes 100 protein structures from each of the 6 enzyme classes and the goal is to correctly predict enzyme class for these proteins.
- **DHFR** is a dataset of 756 networks of inhibitors of dihydrofolate reductase and it is divided into two classes. The goal is to predict the correct class for each network structure [16].
- **REDDIT-BINARY** is a dataset where each network corresponds to an online discussion thread. The task is to identify whether a network belongs to a *question/answer*-based community or a *discussion*-based community [17].
- **COLLAB** is a scientific collaboration dataset from 3 public collaboration datasets [18] and the task is to determine whether the ego-network of a researcher belongs to High Energy, Condensed Matter or Astro Physics field.
- **IMDB-BINARY** is a movie collaboration dataset which collects cast and genre information of two types of movies, *romance* and *action* on IMDB. For each network, nodes represent actors/actresses and there is an edge between them if they appear in the same movie, and the task is to predict the genre of the movie the network represents [17].
- **IMDB-MULTI** is multi-class version of IMDB-BINARY and contains a balanced set of ego-networks derived from Comedy, Romance and Sci-Fi genres [17].
- **ACM/DBLP** are two co-authorship networks and are used for evaluating network alignment tasks. Nodes represent authors and there is an edge between two authors if they have published a paper together. Node attributes represents the areas of research the authors are working on [6].

B – Evaluation Metrics. We quantify the effectiveness of the proposed algorithms by measuring the following two aspects, including (1) the relative change of $f(\cdot)$ function (i.e., $\frac{\Delta f}{f}$), and (2) the accuracy of a multi-network mining task (e.g., network classification, network alignment) before and after performing adversarial attacks.

We perform the evaluations on two multi-network mining tasks, including random walk graph kernel and network alignment. For the former, (1) on each selected dataset, we randomly selected 100 pairs of networks of the same class, then for every pair of networks, we apply ADMIRING or comparison methods to attack one of them, and re-compute the graph kernel to compare the relative change; (2) we also trained a SVM classifier with graph kernel for each of the three datasets, *MUTAGENICITY, IMDB-BINARY and PROTEINS*, respectively; for every network (i.e., $\mathcal{G}_1$) in the testing set, we attacked it with ADMIRING or comparison methods to reduce its similarity with one random training network (i.e., $\mathcal{G}_2$) . The new classification result is from applying the trained SVM classifier on the attacked test networks. We report the average classification accuracy by repeating the above attacking strategy for 10 times. For the latter (network alignment), we first randomly sample 20 pairs of subgraphs from *ACM*, of which the size uniformly varies from 100 nodes to 2,000 nodes. The sampling results form ACM-ACM pairs (i.e., ACM-ACM). Similarly, we generate 20 DBLP-DBLP subgraph pairs (i.e., DBLP-DBLP) from *DBLP* and ACM-DBLP subgraph pairs (i.e., ACM-DBLP) from *ACM* and *DBLP*, respectively. Then we apply ADMIRING or baseline methods to every pair of subgraphs and report the average cross-network alignment accuracy before and after the attack.

C – Comparison Methods. We evaluate the proposed ADMIRING method with different attacking strategies, i.e., ADMIRING-E, ADMIRING-N, ADMIRING-A for attacking edges, nodes and node attributes respectively. We also evaluate a batch-mode variant of ADMIRING. That is, in Algorithm 1, instead of selecting one network element at each iteration, we select the top-k elements with the highest influence scores in one iteration. We use a suffix 'v' to denote such a variant. If the proposed fast algorithm (Algorithm 2) is used to compute the influence score, we add another suffix 'fast'. For example, ADMIRING-N-v means attacking networks by nodes in the batch mode; ADMIRING-E-fast means attacking networks by edges using fast solution in Algorithm 2, etc. For comparison, we use the following methods to select the top-k influential network elements, including,

(1) *Random,* which randomly selects k network elements in the first network to attack and for a specific type of network elements, we randomly select k elements and perturb (remove or alter) the selected network elements from $\mathcal{G}_1$.

(2) *Bruteforce,* which re-computes f after attacking each element and selects the one with the highest Δf in each iteration. At each of the k iterations, we calculate the change of mining results, i.e., Δf, by iterating through all the network elments of the specified type and add the element which causes the largest change of $f(\mathbf{X})$ to $\mathcal{P}$;

(3) *Bruteforce-v,* which uses *Bruteforce* in the batch mode. At the beginning, we compute Δf by iterating through all network elements of the specified type, and add the network elements with the top-k largest Δf to $\mathcal{P}$.

(4) *Q-Matrix,* which selects the top-k network elements based on $\mathbf{Q} = (\mathbf{I} - c\mathbf{N}_\times\mathbf{A}_\times)^{-1}$. Recall that $\mathbf{Q} = (\mathbf{I} - c\mathbf{N}_\times\mathbf{A}_\times)^{-1}$ is an $n^2 \times n^2$ matrix and we can use a block-matrix representation (i.e., $\mathbf{Q} = [\mathbf{W}^{ij}]_{i,j=1,\ldots,n}$, where $\mathbf{W}^{ij}$ is at the $(i,j)^{\text{th}}$ position of $\mathbf{Q}$ with size $n \times n$). In edge attack, the aggregation of the entries in block matrix $\mathbf{W}^{ij}$ can be considered as the contribution of the edge $\mathbf{A}_1(i,j)$ to the mining result (i.e., $f(\mathbf{X})$), then we compute the influence of edge $\mathbf{A}_1(i,j)$ by the aggregation of the entries in $\mathbf{W}^{ij}$ and select the top-k edges. In node attack, we calculate the influence of node i in $\mathcal{G}_1$ by summing up all blocks $\mathbf{W}^{ij}|_{\mathbf{A}_1(i,j)=1}$ and select the top-k nodes. While in node attribute attack, we compute the influence of $\mathbf{N}_1^l(i,i)$ as $\sum_{j=1}^{n}\mathbf{W}^{ij}(l,:)$ and select the top-k attributes.

(5) *PageRank* which measures the importance of nodes in a given network [19] and here we select top-k network elements based on PageRank ranking results. In $\mathcal{G}_1$, we define the PageRank value of an edge $\mathbf{A}_1(i,j)$ (i.e., $v(\mathbf{A}_1(i,j))$ as follows,

$$v(\mathbf{A}_1(i,j)) = (\mathbf{r}(i) + \mathbf{r}(j)) \times \max_{m\in\{i,j\}} \mathbf{r}(m)$$

where $\mathbf{r}(i)$ is the PageRank value of node i in $\mathcal{G}_1$. In ADMIRING-E, we choose the top k edges with the highest PageRank edge values; in ADMIRING-N, we select the k nodes in $\mathcal{G}_1$ with the largest PageRank values; and in ADMIRING-A, we first select the node with the the highest PageRank value and then select the attribute dimension with the largest value if the corresponding node attribute vector has multiple dimensions.

D – Repeatability and Machine Configuration. The experiments are performed on a Red Hat Linux Server - 6.9 with Intel Xeon E7-4820 at 2.00 GHz and 32GB RAM. Random walk graph kernel is implemented in Python 3.6 and network alignment in Matlab. All datasets are publicly available. We will release the code upon the publication of this paper.

4.2 Effectiveness Results

A – Attacking random walk graph kernel. We first compare the proposed ADMIRING with comparison methods in the task of attacking random walk graph kernel with different types of network elements, i.e., edges, nodes and attributes. The results of $\frac{\Delta f}{f}$ are summarized in Figures 2, 3, and 4, respectively. We can see that the proposed ADMIRING and its batch-mode variant ADMIRING-v (two red bars) achieve a very similar performance as their *bruteforce* counterparts (two yellow bars). As we will show in the next subsection, the *bruteforce* methods are much more expensive in terms of computation. In the meanwhile, the proposed methods (two red bars) consistently outperform all the remaining methods by a large margin in the three attacking scenarios across all datasets. For example, on *REDDIT-BINARY* dataset, the proposed method ADMIRING-E is 14.5 times better than the best competitor method (i.e., ADMIRING-E vs. *Q-Matrix*) by attacking edges; on *DHFR* dataset, by attacking nodes, the proposed ADMIRING-N is 259% better than *Q-Matrix*, and for node attribute attack, our proposed method ADMIRING-A is 1.93 times better than *Q-Matrix* on *ENZYMES* dataset. For the reported results in Figures 2, 3 and 4, the budget k is set to be 10, and the damping factor α is 0.2 for result in Figure 4.

Second, we evaluate and compare the impact of different attacking methods on the network classification results, summarized in Figure 5. We can see both the ADMIRING method and the *bruteforce* method can significantly impact the classification results. For example, on *PROTEINS* dataset, our proposed method can reduce the classification accuracy by 8.82%, 10.52%, and 6.98% by attacking edges, nodes and attributes, respectively. On the other hand, the attacking effect by other methods is quite marginal. For example, the best competitor method (*Q-Matrix*), can only achieve a reduction of 3.35% in classification accuracy by attacking nodes.

B – Attacking network alignment. Here, we evaluate different attacking methods for the task of network alignment. The results on *ACM* and *DBLP* datasets are presented in Figure 6. We have the following observations: (1) both the proposed ADMIRING method and the *bruteforce* method can considerably impact the network alignment accuracy, and outperform all other baseline methods. For example, on *ACM* dataset, our proposed method can reduce the alignment accuracy by 8.53%, 10.50%, and 4.89% by attacking edges, nodes and attributes, respectively; (2) the impact on the alignment accuracy by other remaining methods are all very marginal. For instance, on *DBLP* dataset with the Q-Matrix method, the alignment accuracy only decreases 2.14%, 3.63%, and 0.84% by attacking edges, nodes and attributes, respectively.

C - Parameter studies. There are three main parameters in the proposed ADMIRING algorithms, including (1) the budget k, (2) the damping factor α for attacking node attribute, and (3) the regularization parameter c . Figure 7a-7c present some sensitivity results w.r.t. these three parameters (c_0 is set as $c_0 = 1/(\max(\text{eigenvalue}(\mathbf{N}_1\mathbf{A}_1)) \times \max(\text{eigenvalue}(\mathbf{N}_2\mathbf{A}_2)) + 1)$ in 7a and 7b, α is set as 0 in 7a and 7c.). We can observe that our proposed method ADMIRING has a very close performance to the brute force method w.r.t all the three parameters in a wide range. In the meanwhile, the performance increases monotonically w.r.t. α, which is consistent with our intuition (i.e., the more we alter the attributes, the more the mining results are influenced). Another observation w.r.t c is that $\frac{\Delta f}{f}$ decreases significantly as we reduce the value of c. A possible explanation for this is as follows. Recall that $\text{vec}(\mathbf{X}) = \mathbf{Q}\mathbf{N}_\times\mathbf{q}_\times$. By expanding the matrix inverse $\mathbf{Q}$ ($(\mathbf{I} - c\mathbf{A})^{-1} = \sum_{i=0}^{\infty} c^i\mathbf{A}^i$), we can see that as c decreases, $\mathbf{Q}$ is approaching $\mathbf{I}$, therefore the effect of changing $\mathbf{A}$ (i.e., attacking network in our problem) is infinitesimal.

D - Additional Experiment Result. Figure 9a-9c present the tendency of Δf as we increase k (i.e., Δf vs. k) on *DHFR* dataset in attacking three types of network elements. We have the following observations. First, as we increase the number of attacks (i.e., removal of edges/nodes or alteration of attributes), the random walk graph kernel is monotonically decreasing, which is consistent with our intuition. Second, both the proposed method and the *bruteforce* method can achieve very similar results in terms of altering f, while significantly outperforming other comparison methods. For example, in edge attacks, our proposed method is 3.54 times better than the best competitor (*Q-Matrix* method) at each step of attacks on average.

4.3 Efficiency Results

The scalability result of the proposed methods on *COLLAB* dataset is in Figure 8a. We can see that the proposed fast solutions (ADMIRING-E-Fast and ADMIRING-N-Fast) scale linearly w.r.t. the input network size(i.e., the number of edges), which is consistent with Lemma 4.

The quality vs. speed trade-off of the proposed methods on *COLLAB* dataset is in Figure 8b. The proposed fast solutions (ADMIRING-E-Fast and ADMIRING-N-Fast) achieve a good balance between the Δf and running time. For instance, they are $6,209\times$ faster than the *bruteforce* counterparts, while maintaining a similar attacking effect. Compared with the straight-forward implementations (ADMIRING-E and ADMIRING-N), their fast solutions (ADMIRING-E-Fast and ADMIRING-N-Fast) are $133\times$ and $139\times$ faster respectively, with the same attacking effect (i.e., the horizontal dashed lines). The results are consistent with our analysis in Theorem 1. That is, due to the quadratic complexity, the straightforward implementation is computationally inefficient, even on moderate-sized networks with up to thousands of nodes and edges. For the baseline methods (PageRank-E/N and Q-

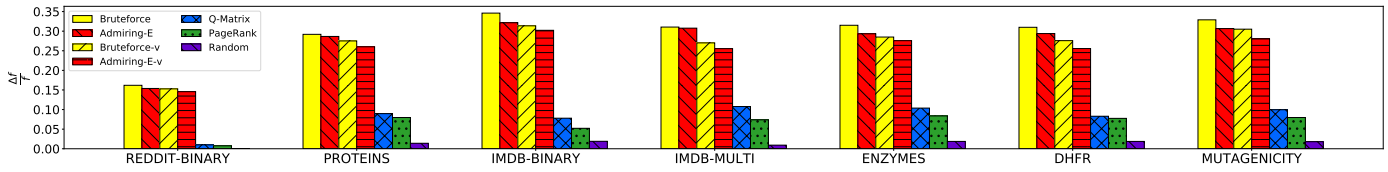


Fig. 2: $\frac{\Delta f}{f}$ on seven datasets by removing $k = 10$ influential edges. Higher is better. Best viewed in color.

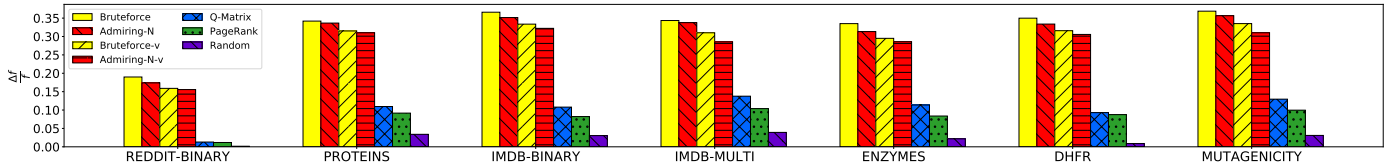


Fig. 3: $\frac{\Delta f}{f}$ on seven datasets by removing $k = 10$ influential nodes. Higher is better. Best viewed in color.

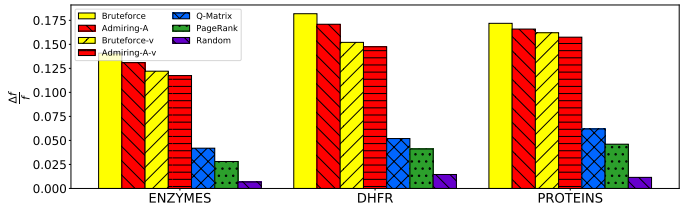


Fig. 4: $\frac{\Delta f}{f}$ on three datasets by altering $k = 10$ influential node attributes ($\alpha = 0.2$). Higher is better. Best viewed in color.

Matrix-E/N), although they are fast (x-axis), they are much less effective in attacking the networks (y-axis).

5 RELATED WORK

In this section, we review the related work in terms of (a) multi-network mining, and (b) adversarial learning.

Multi-network Mining aims to collectively leverage the relationship among multiple networks for a better mining outcome or reveal patterns that would have been invisible if we analyze each individual network separately. Graph kernel is a family of methods that measure the similarity between two input networks based on walks [5], [7], limited-sized subgraphs [20], [21] or subtree patterns [22], [23]. Network alignment aims to identify node correspondence across different networks. A fundamental assumption is topology consistency [24], which might be violated in some applications. Some recent work suggests that the alignment can be enhanced by augmenting the topology consistency with the node and edge attributes [6]. Furthermore, network alignment task is found to be complementary with network completion in a joint optimization framework [9]. Multiple networks are often not independent of each other, but are manifested as a network of networks, where each node of the main network itself is another domain network [25], [26]. By leveraging the intrinsic relationship among these networks, it often brings significant performance gain to the mining tasks. The main network can contextualize the mining tasks in each domain-specific network by providing the consistency constraints across networks for both ranking [25] and clustering [27]. With the increased complexity of the networked system, some efforts have been towards explainable networked prediction [28], [29]. More recently, the compatible and complementary information from multiple networks is exploited to refine the learned node representations to be robust against the defects in each individual network [30].

Adversarial Learning studies the learning process in the adversarial setting, where an adversary can attack the learning model at the training phase (*poisoning*) [31] or at the test phase (*evasion*) [32]. In white-box evasion attack, the adversary has the full knowledge of the learning model (i.e., parameters and hyperparameters). In this scenario, the construction of the adversarial examples often lends itself to a constrained optimization problem, where the objective is to maximally increase the expected loss or lead to misclassification with a small amount of perturbation. Since the optimization problem is often non-convex and intractable, gradient based approach on an approximation function is often used. In particular, the fast gradient sign method [32] employs the first-order Taylor approximation with a low computation cost. The Jacobian-based saliency map [33] computes a saliency map from gradient that reflects the impact of each feature dimension of the classification result. The adversarial example is crafted by perturbing the most salient features. On the other hand, the attacker has no access to the learning models in black-box attack. One approach is by training a subtitute model by querying the black-box model and then crafting the adversarial examples using the white-box attack techniques [34]. Apart from leveraging the tranferablility of subtitute models, the zeroth order optimization based attacks directly estimate the gradient of the black-box model [35] and the decision based attacks gradually reduce the perturbation relying solely on the final decision of the model [36]. More recently, adversarial attacks on network mining algorithms start to receive attentions. The perturbations to the network attributes and structure can be constructed to mislead the graph convolution models [37]. Based on the idea of meta-gradients from meta-learning to solve the bilevel optimization problem, an effective algorithm is proposed to alter the structure of a network by adding/deleting edges so as to degrade the global performance of node classification [38], [39].

6 CONCLUSION

In this paper, we study the problem of adversarial multi-network mining and formulate it as an optimization prob-

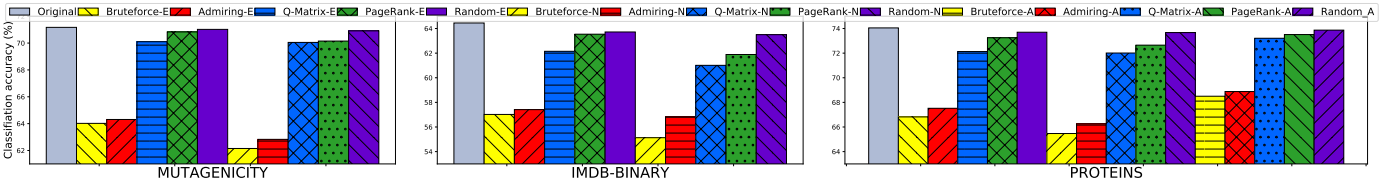


Fig. 5: Network classification accuracy after attacking by different methods ($k = 10, \alpha = 0.2$). Best viewed in color.

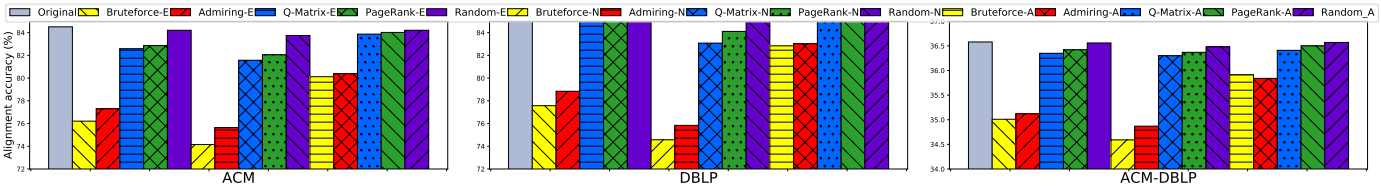
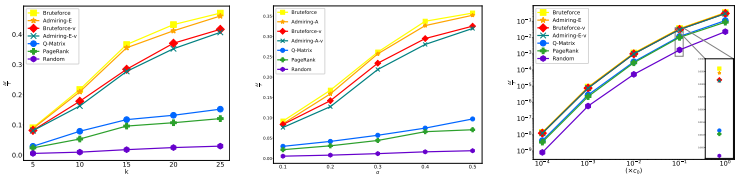
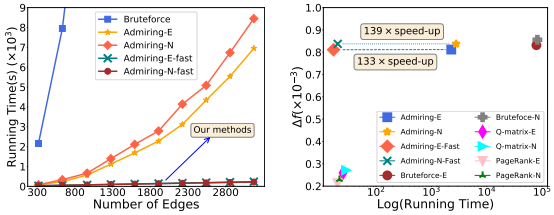


Fig. 6: Network alignment accuracy after attacking by different methods ($k = 10, \alpha = 0.2$). Best viewed in color.



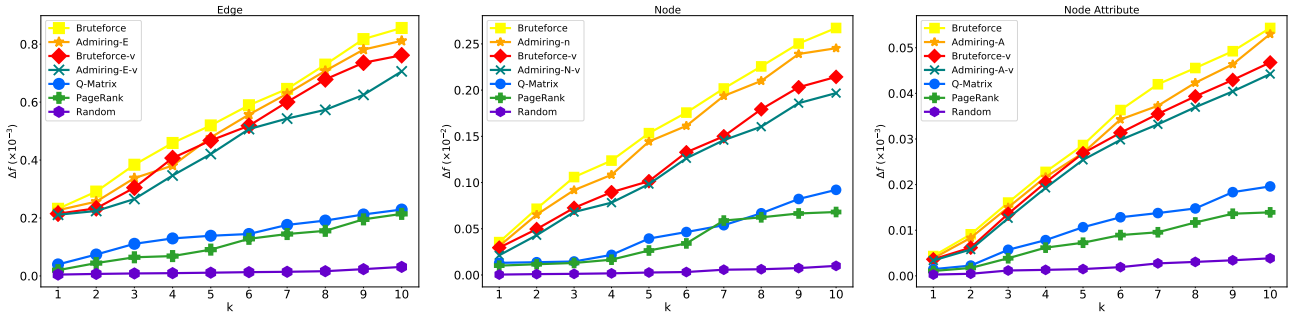
(a) $\frac{\Delta f}{f}$ vs. k. (b) $\frac{\Delta f}{f}$ vs. α.($k = 10$) (c) $\frac{\Delta f}{f}$ vs. c. ($k = 10$)

Fig. 7: Parameter analysis.



(a) Running time vs. m. (b) Δf vs. running time.

Fig. 8: Running time comparison (left), and Δf vs. running time of six methods (right). ($k = 10$).



(a) Attack edges ($k = 10$). (b) Attack nodes ($k = 10$). (c) Attack node attributes ($\alpha = 0.2$).

Fig. 9: Δf vs. budget k on *DHFR* Dataset. Higher is better. Best viewed in color.

lem. The key idea is to effectively characterize the change of mining results w.r.t. the perturbations to the network. We propose a family of algorithms (ADMIRING) that are able to measure the influence of network elements to the mining results, along with its fast solver of a linear complexity. The empirical evaluations on real-world datasets demonstrate the efficacy of the proposed algorithms. The proposed method is specifically designed for Sylvester equation defined over the input networks, which applies to a variety of multi-network mining tasks. Future work includes generalizing the current method beyond Sylvester equation based solution in the black-box model setting as well as designing effective defensive strategies.

REFERENCES

[1] K. M. Borgwardt, C. S. Ong, S. Schönauer, S. Vishwanathan, A. J. Smola, and H.-P. Kriegel, "Protein function prediction via graph kernels," *Bioinformatics*, vol. 21, no. suppl_1, pp. i47–i56, 2005.

[2] X. Kong, J. Zhang, and P. S. Yu, "Inferring anchor links across multiple heterogeneous social networks," in *CIKM*. ACM, 2013, pp. 179–188.

[3] L. Li, H. Tong, N. Cao, K. Ehrlich, Y.-R. Lin, and N. Buchler, "Replacing the irreplaceable: Fast algorithms for team member recommendation," in *WWW*, 2015, pp. 636–646.

[4] B. Du, S. Zhang, N. Cao, and H. Tong, "First: Fast interactive attributed subgraph matching," in *KDD*. ACM, 2017, pp. 1447–1456.

[5] S. V. N. Vishwanathan, N. N. Schraudolph, R. Kondor, and K. M. Borgwardt, "Graph kernels," *JMLR*, no. Apr, pp. 1201–1242, 2010.

[6] S. Zhang and H. Tong, "Final: Fast attributed network alignment," in *KDD*. ACM, 2016, pp. 1345–1354.

[7] K. M. Borgwardt, N. N. Schraudolph, and S. Vishwanathan, "Fast computation of graph kernels," in *NIPS*, 2007, pp. 1449–1456.

[8] J. D. Gardiner, A. J. Laub, J. J. Amato, and C. B. Moler, "Solution of the sylvester matrix equation axb t+ cxd t= e," *ACM Transactions on Mathematical Software (TOMS)*, vol. 18, no. 2, pp. 223–231, 1992.

[9] S. Zhang, H. Tong, J. Tang, J. Xu, and W. Fan, "ineat: Incomplete network alignment," in *ICDM*. IEEE, 2017, pp. 1189–1194.

[10] B. Du and H. Tong, "Fasten: Fast sylvester equation solver for graph mining," in *KDD*, ser. KDD '18. New York, NY, USA: ACM, 2018.

[11] P. J. Huber *et al.*, "The 1972 wald lecture robust statistics: A review," *The Annals of Mathematical Statistics*, pp. 1041–1067, 1972.

[12] P. W. Koh and P. Liang, "Understanding black-box predictions via influence functions," *arXiv preprint arXiv:1703.04730*, 2017.

[13] K. B. Petersen and M. S. Pedersen, "The matrix cookbook," nov 2012, version 20121115.

[14] X. Zeng and M. A. T. Figueiredo, "The ordered weighted l_1 norm: Atomic formulation, dual norm, and projections," *CoRR*, vol. abs/1409.4271, 2014. [Online]. Available: http://arxiv.org/abs/1409.4271

[15] J. Kazius, R. McGuire, and R. Bursi, "Derivation and validation of toxicophores for mutagenicity prediction," *Journal of Medicinal Chemistry*, vol. 48, no. 1, pp. 312–320, 2005.

[16] J. J. Sutherland, L. A. O'brien, and D. F. Weaver, "Spline-fitting with a genetic algorithm: A method for developing classification structure- activity relationships," *Journal of chemical information and computer sciences*, vol. 43, no. 6, 2003.

[17] P. Yanardag and S. Vishwanathan, "Deep graph kernels," in *KDD*, 2015.

[18] J. Leskovec, J. Kleinberg, and C. Faloutsos, "Graphs over time: densification laws, shrinking diameters and possible explanations," in *KDD*. ACM, 2005.

[19] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the web." Stanford InfoLab, Tech. Rep., 1999.

[20] T. Horváth, T. Gärtner, and S. Wrobel, "Cyclic pattern kernels for predictive graph mining," in *KDD*, 2004, pp. 158–167.

[21] N. Shervashidze, S. V. N. Vishwanathan, T. Petri, K. Mehlhorn, and K. M. Borgwardt, "Efficient graphlet kernels for large graph comparison," *JMLR - Proceedings Track*, vol. 5, pp. 488–495, 2009.

[22] P. Mahé, N. Ueda, T. Akutsu, J.-L. Perret, and J.-P. Vert, "Graph kernels for molecular structure-activity relationship analysis with support vector machines," *Journal of Chemical Information and Modeling*, vol. 45, no. 4, pp. 939–951, 2005.

[23] S. Hido and H. Kashima, "A linear-time graph kernel," in *ICDM*, 2009.

[24] L. Akoglu, H. Tong, and D. Koutra, "Graph based anomaly detection and description: a survey," *Data mining and knowledge discovery*, vol. 29, no. 3, pp. 626–688, 2015.

[25] J. Ni, H. Tong, W. Fan, and X. Zhang, "Inside the atoms: ranking on a network of networks," in *KDD*, 2014, pp. 1356–1365.

[26] C. Chen, J. He, N. Bliss, and H. Tong, "On the connectivity of multi-layered networks: Models, measures and optimal control," in *ICDM*, 2015.

[27] J. Ni, H. Tong, W. Fan, and X. Zhang, "Flexible and robust multi-network clustering," in *KDD*, 2015, pp. 835–844.

[28] L. Li, H. Tong, and H. Liu, "Towards explainable networked prediction," in *CIKM*, ser. CIKM '18, 2018.

[29] Q. Zhou, L. Li, N. Cao, N. Buchler, and H. Tong, "Extra: Explaining team recommendation in networks," in *RecSys*. ACM, 2018.

[30] J. Ni, S. Chang, X. Liu, W. Cheng, H. Chen, D. Xu, and X. Zhang, "Co-regularized deep multi-network embedding," in *WWW*, 2018.

[31] B. Li, Y. Wang, A. Singh, and Y. Vorobeychik, "Data poisoning attacks on factorization-based collaborative filtering," in *NIPS*, 2016.

[32] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *CoRR*, vol. abs/1412.6572, 2014.

[33] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, "The limitations of deep learning in adversarial settings," in *Security and Privacy (EuroS&P), 2016 IEEE European Symposium*.

[34] N. Papernot, P. McDaniel, and I. Goodfellow, "Transferability in machine learning: from phenomena to black-box attacks using adversarial samples," *arXiv preprint arXiv:1605.07277*, 2016.

[35] P.-Y. Chen, H. Zhang, Y. Sharma, J. Yi, and C.-J. Hsieh, "Zoo: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models," in *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*. ACM, 2017.

[36] W. Brendel, J. Rauber, and M. Bethge, "Decision-based adversarial attacks: Reliable attacks against black-box machine learning models," *arXiv preprint arXiv:1712.04248*, 2017.

[37] D. Zügner, A. Akbarnejad, and S. Günnemann, "Adversarial attacks on neural networks for graph data," in *KDD*. ACM, 2018, pp. 2847–2856.

[38] D. Zgner and S. Gnnemann, "Adversarial attacks on graph neural networks via meta learning," in *ICLR*, 2019.

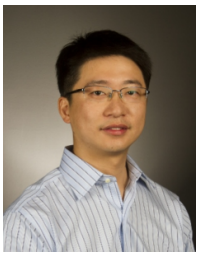
[39] H. Dai, H. Li, T. Tian, X. Huang, L. Wang, J. Zhu, and L. Song, "Adversarial attack on graph structured data," in *ICML*. Stockholmsmssan, Stockholm Sweden: PMLR, 10–15 Jul 2018, pp. 1115–1124.

Qinghai Zhou received the B.S. degree in Physics from Nanjing University in 2014 and the M.S. degree in Computer Engineering from the University of Florida in 2016. He is currently working toward the Ph.D. degree in the Department of Computer Science, University of Illinois at Urbana-Champaign. His research interests include large-scale data mining and machine learning, especially graph mining.

PLACE PHOTO HERE

Liangyue Li is now an applied scientist with Amazon search. He received the Ph.D in Computer Science under the supervision of Prof. Hanghang Tong from CIDSE, Arizona State University and my B.Eng from Tongji University in China. His research interests are in large scale data mining and machine learning, especially for graph data with application to social network analysis.

Nan Cao is a professor at TongJi University in China, with the joint appointment at both College of Design and Innovation and College of Software Engineering. He is the founding director of Tongji Intelligent Big Data Visualisation Lab (iDVx Lab) . Dr. Cao is also an adjunct professor at NYU ShangHai and NYU Tandon. Before joining Tongji and NYU Shanghai, he was a research staff member at IBM T.J. Watson Research Center and he has worked for IBM for 10 years during 2005 - 2015. Dr. Nan Cao received his PhD degree from the Hong Kong University of Science and Technology. He led many research projects related to data visualization in domains of information security, healthcare informatics, and urban computing. His primary expertise and research interests are information visualization and visual analysis. He is specialized in producing novel visualization techniques to represent and analysis complex (big, heterogenous, multi-dimensional, dynamic) real world data. He has received several awards, including ACM CHI Honorable Mention of the Best Paper Award (2018), ACM Rising Start Award (Shanghai) (2016), ACM IUI Best Paper Award (2016), IBM Outstanding Technical Achievement Award (2015), IEEE VAST Honorable Mention (2014), Microsoft Global MVP Award (Oct. 2015), etc.



Lei Ying is currently a professor at the Electrical Engineering and Computer Science Department of the University of Michigan, Ann Arbor, and an Associate Editor of the IEEE Transactions on Information Theory and the Elsevier Performance Evaluation. He received his B.E. degree from Tsinghua University, Beijing, China, and his M.S. and Ph.D in Electrical and Computer Engineering from the University of Illinois at Urbana-Champaign. His research is broadly in the interplay of complex stochastic systems and big-data, including large-scale communication/computing systems for big-data processing, private data marketplaces, and large-scale graph mining. He won the Young Investigator Award from the Defense Threat Reduction Agency (DTRA) in 2009 and NSF CAREER Award in 2010. He was the Northrop Grumman Assistant Professor in the Department of Electrical and Computer Engineering at Iowa State University from 2010 to 2012. His papers have received the best paper award at IEEE INFOCOM 2015, the Kenneth C. Sevcik Outstanding Student Paper Award at ACM SIGMETRICS/IFIP Performance 2016, and the WiOpt18 Best Student Paper Award; his papers have also been selected in ACM TKDD Speical Issue Best Papers of KDD 2016, Fast-Track Review for TNSE at IEEE INFOCOM 2018 (7 out of 312 accepted papers were invited), and Best Paper Finalist at MobiHoc 2019.

Hanghang Tong is an associate professor with the Department of Computer Science at University of Illinois at Urbana-Champaign. Before that he was an associate professor at School of Computing, Informatics, and Decision Systems Engineering (CIDSE), Arizona State University. He received his M.Sc. and Ph.D. degrees from Carnegie Mellon University in 2008 and 2009, both in Machine Learning. His research interest is in large scale data mining for graphs and multimedia. He has received several awards, including SDM/IBM Early Career Data Mining Research award (2018), NSF CAREER award (2017), ICDM 10-Year Highest Impact Paper award (2015) and four best paper awards (TUP'14, CIKM'12, SDM'08, ICDM'06). He is the Editor-in-Chief of SIGKDD Explorations (ACM), an action editor of Data Mining and Knowledge Discovery (Springer), and an associate editor of Knowledge and Information Systems (Springer) and Neurocomputing Journal (Elsevier); and has served as a program committee member in multiple data mining, database and artificial intelligence venues (e.g., SIGKDD, SIGMOD, AAAI, WWW, CIKM, etc.).