TINET: Learning Invariant Networks via Knowledge Transfer

Chen Luo[1,*,†], Zhengzhang Chen[2,*], Lu-An Tang[2], Anshumali Shrivastava[1],
Zhichun Li[2], Haifeng Chen[2], Jieping Ye[3]
[1]Rice University
[2]NEC Laboratory America
[3]University of Michigan
{cl67,anshumali}@rice.edu,{zchen,ltang,zhichun,haifeng}@nec-labs.com,jpye@umich.edu

ABSTRACT

The latent behavior of an information system that can exhibit extreme events, such as system faults or cyber-attacks, is complex. Recently, the *invariant network* has shown to be a powerful way of characterizing complex system behaviors. Structures and evolutions of the invariance network, in particular, the vanishing correlations, can shed light on identifying causal anomalies and performing system diagnosis. However, due to the dynamic and complex nature of real-world information systems, learning a reliable invariant network in a new environment often requires continuous collecting and analyzing the system surveillance data for several weeks or even months. Although the invariant networks learned from old environments have some common entities and entity relationships, these networks cannot be directly borrowed for the new environment due to the domain variety problem. To avoid the prohibitive time and resource consuming network building process, we propose **TINET**, a knowledge transfer based model for accelerating invariant network construction. In particular, we first propose an entity estimation model to estimate the probability of each source domain entity that can be included in the final invariant network of the target domain. Then, we propose a dependency construction model for constructing the unbiased dependency relationships by solving a two-constraint optimization problem. Extensive experiments on both synthetic and real-world datasets demonstrate the effectiveness and efficiency of **TINET**. We also apply **TINET** to a real enterprise security system for intrusion detection. **TINET** achieves superior detection performance at least 20 days lead-lag time in advance with more than 75% accuracy.

1 INTRODUCTION

Dynamic information systems, such as cyber-physical systems, enterprise systems, and cloud computing facilities, are inherently complex. These large-scale systems usually consist of a great variety of components/entities that work together in a highly complex and coordinated manner. For example, the cyber-physical system is typically equipped with a large number of wireless sensors that keep recording the running status of the local physical and software components.

Recently, a very promising means for studying complex systems has emerged through the concept of invariants [6, 8, 13, 15–17]. Such invariant models focus on discovering stable and significant dependencies between pairs of system entities that are monitored through surveillance data recordings, so as to profile the system status and perform subsequent reasoning. A strong dependency between a pair of entities is called invariant relationship. By combining the invariants learned from all monitoring entities, a global system dependency profile can be obtained. The significant practical value of such an invariant profile is that it provides important clues on abnormal system behaviors, and in particular on the source of anomalies, by checking whether existing invariants are broken [6, 13, 17, 20, 33].

For fully utilizing the invariant model, the first prerequisite is to construct the *invariant network* from the system streaming data. In the invariant network, a node represents a system component/entity and an edge indicates a stable, significant interaction between two system entities. During the construction process, we infer the network structure and invariant/dependency relations by continuously collecting and analyzing the surveillance data generated by the system.

Due to the dynamic and complex nature of the real-world information system, learning a robust invariant network often requires a very long training time. For instance, in enterprise security systems (See Fig. 1), the construction process needs to collect at least 30 days of streaming data to identify important entities and relationships reliably. However, it is often impractical and uneconomical to wait for such long time, especially for some mission-critical environments (*e.g.*, nuclear plants) and PoC (Proof of Concept) scenarios. Unfortunately, utilizing fewer days' data will lead to an unreliable network with poor performance. To illustrate this, we show, in Section 4.7, that if we utilize only three days of training data then we get a poor recall of 10% for intrusion detection, which is unacceptable in real-world systems. Thus, to deploy our models reliably in a new environment, we have to wait a long time (*e.g.*, 30 days) before we can get any reliable invariant network.

Enlightened by the cloud services [21], one way to "speed up" the learning process is by reusing a unified invariant network model in different domains/environments. However, due to the

*Both authors contributed equally.
†Work done during an internship at NEC Labs America.

KDD'18, August 19–23, 2018, London, United Kingdom

ACM ISBN 978-1-4503-5552-0/18/08…$15.00

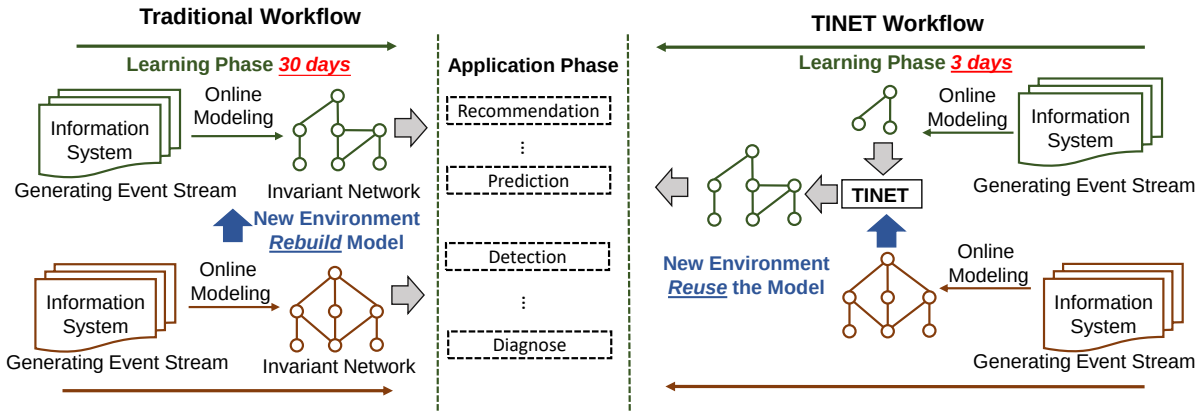


Figure 1: Comparison between a traditional work flow and TINET work flow of learning the invariant network. TINET extracts the knowledge from a well-trained invariant network to speed up the training process of a new invariant network.

domain/environment variety problem [27], directly apply the invariant network learned from an old environment to a new environment often cannot achieve good performance. In Section 4.7, we show that directly transferring the old invariant network can only get a 15% precision result, which is also unacceptable in real-world applications.

The good news is that it is easy and fast to compute a partial, significantly incomplete, invariant network of the new environment of interest. To avoid the prohibitive time and resource consuming network building process, the aim of this paper is to complete this partial information reliably by transferring knowledge from another invariant network. Formally, given a partial invariant network of the target domain and a complete invariant network of the source domain. How can we reliably compute the full invariant network of the target domain? There are two major challenges for achieving this:

- **Challenge 1: Identify the domain-specific/irrelevant entities between two environments.** As aforementioned, since the environments are different, not all entities of the source domain are related to the target domain. For instance, an invariant network from an electronic factory system will have entities such as energy-related program, which will not exist in an IT company enterprise system. Thus, we need to identify the right entities that can be transferred from the source domain to the target one.
- **Challenge 2: Constructing the invariant relationships on the new environment.** After transferring the entities from source to target, we also need to identify invariant relationships between the entities to complete the invariant network. The challenge is to extract the invariant information from the old environment, and then combine this knowledge with the partial invariant network of the new environment.

To address the aforementioned two challenges, in this paper, we propose **TINET**, an efficient and effective method for transferring knowledge between *Invariant Networks*. **TINET** consists of two sub-models: **EEM** (Entity Estimation Model) and **DCM** (Dependency Construction Model). First, **EEM** filters out irrelevant entities from the source network based on entity embedding and manifold learning. Only the entities with statistically high correlations with the target domain are transferred. Then, after transferring the entities, **DCM** model effectively constructs invariant (dependency) relationships between different entities for the target network by solving a two-constraint optimization problem. Our approach can use an existing invariant network of an old environment to complete the partial invariant network of the new environment. As a result, the costly time and resource consuming re-building process of the invariant network from scratch can be avoided. We perform an extensive set of experiments on both synthetic and real-world data to evaluate the performance of **TINET**. The results demonstrate the effectiveness and efficiency of our proposed algorithm. We also apply **TINET** to real enterprise security systems for intrusion detection. By using **TINET**, we can achieve more than 75% accuracy after 3 days of training time, this performance is almost the same as 30 days of construction of invariant network without using **TINET**. On the contrary, building an invariant network using only 3 days of data can only get 10% accuracy. Thus, our method can achieve superior detection performance at least 20 days lead-lag time in advance with more than 75% accuracy.

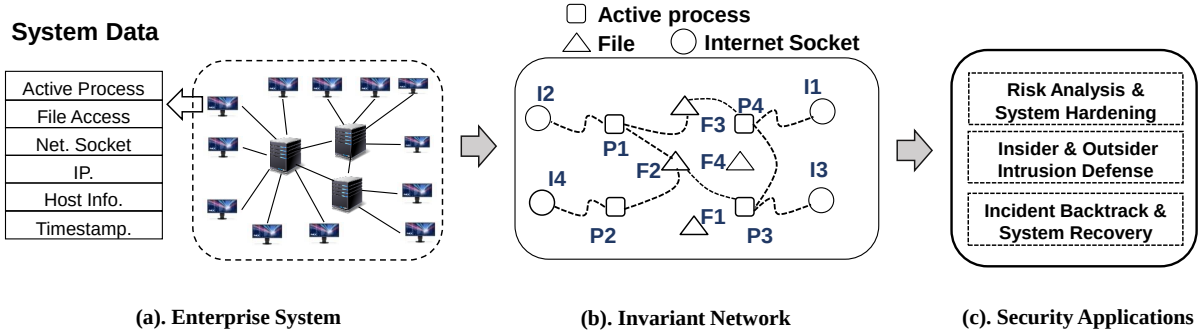
2 PRELIMINARIES AND PROBLEM STATEMENT

In this section, we first introduce the invariant network and then define our problem.

2.1 Invariant Network

We formally define an invariant network as an undirected weighted graph $G = \{V, E\}$, where $V = \{v_1, ..., v_n\}$ is the set of n heterogeneous system entities and $E = \{e_1, ..., e_m\}$ is the set of m edges between pairs of entities. The edges exist depending on whether there are invariant or dependency relationships between the corresponding pairs of system entities.

For example, in an enterprise security system (as illustrated in Fig. 2), an invariant network is a graph between different computer system entities such as processes, files, and Internet sockets. The



(a). Enterprise System **(b). Invariant Network** **(c). Security Applications**

Figure 2: An overview of an enterprise security system. It first uses the surveillance data of a long period of time (*e.g.*, one month) to learn an invariant network, then applies this invariant network for various forensic applications.

edges indicate the stable causal dependencies including a process accessing a file, a process forking another process, and a process connecting to an Internet socket.

Depending on the type of the collected system data, there are different ways to generate the invariant relationships/edges. For the time series data (*e.g.*, sensor readings from a cyber-physical system,), given two pairs of time series $x(t)$ and $y(t)$, where t is the timestamp, the relationship between $x(t)$ and $y(t)$ can be constructed by using the AutoRegressive eXogenous (ARX) model [6, 13, 17, 20]. For the categorical event data (*e.g.*, the process events from an enterprise system), a common system event can be presented as an edge between two nodes, each representing the initiator or the target of the interaction [8, 33].

A network including all the invariant links is referred to as the invariant network. Constructing the invariant network from the system monitoring or surveillance data is referred to as the model training. After the training, the learned complete invariant network, as the system profile, can be applied to many autonomic system management applications such as anomaly detection, system fault diagnose, incident backtrack, and etc [6, 26].

2.2 Problem Statement

Given two environments/domains: a source domain $\mathcal{D}_S$ and a target domain $\mathcal{D}_T$, an information system has been running in $\mathcal{D}_S$ for a long time, while the same information system has only been deployed in $\mathcal{D}_T$ for a short period of time. Let G_S be the well-trained invariant network constructed based on the collected data from $\mathcal{D}_S$. Let $\widehat{G}_T$ be the partial/incomplete invariant network constructed based on the collected data from $\mathcal{D}_T$. Our main goal is to transfer the knowledge from G_S to help construct a complete invariant network $\overline{G}_T$ of the domain $\mathcal{D}_T$.

In the rest of the paper, we will use invariant relationship and dependency interchangeably, and for simplicity, we will use source network (target network) as the short name for the invariant network of the source domain (target domain). The symbols used in the paper are listed in Table 1.

Table 1: Summary of notations

Notation	Description
$\mathcal{D}_S$	The source domain
$\mathcal{D}_T$	The target domain
G_S	The invariant network of source domain
G_T	The ground-truth invariant network of target domain
$\overline{G}_T$	The estimated invariant network of target domain
$\widehat{G}_T$	The partial incomplete invariant network of target domain
$\widehat{G}_S$	A sub-network of G_S, which has the same entity set as $\widehat{G}_T$
$\widetilde{G}_T$	The target domain invariant network after entity estimation
$\widetilde{G}_S$	The sub-network of G_S, which has the same entity set as $\widetilde{G}_T$
$\widehat{A}_T, \widehat{A}_S$	The adjacent matrix of $\widehat{G}_T, \widehat{G}_S$, respectively
$\widetilde{A}_T, \widetilde{A}_S$	The adjacent matrix of $\widetilde{G}_T, \widetilde{G}_S$, respectively
n_S	The number of entities in $\widetilde{G}_S$
u_S	The vector representation of entities in source domain
u_T	The vector representation of entities in target domain
P	A set of meta-paths
W	The weight for each meta-path in P
$F(G_1, G_2)$	The dynamic factor between G_1 and G_2
$\Omega(\cdot)$	The regularization term
λ, μ	The parameters $0 < \mu < 1, 0 < \lambda < 1$

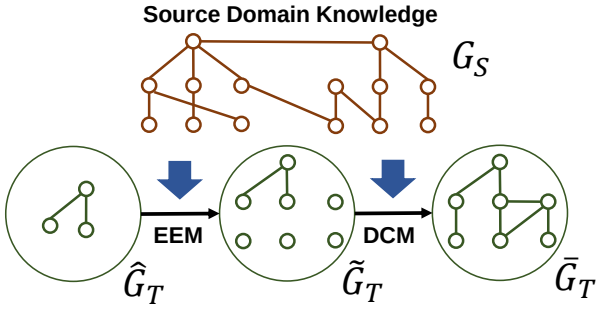


Figure 3: An overview of TINET model. TINET contains two sub-models: Entity Estimation Model (EEM) and Dependency Construction Model (DCM).

3 THE TINET MODEL

To address the two key challenges introduced in Section 1, we propose a knowledge transfer algorithm with two sub-models: **EEM** (Entity Estimation Model) and **DCM** (Dependency Construction Model) as illustrated in Fig. 3. We first introduce these two sub-models separately in details and then analyze the whole algorithm including the parameters and complexity.

3.1 EEM: Entity Estimation Model

For the first sub-model, Entity Estimation Model, our goal is to filter out the entities in the source network G_S that are irrelevant to the target domain. To achieve this, we need to deal with two main challenges: (1) the lack of intrinsic correlation measures among heterogeneous entities, and (2) heterogeneous relations among different entities in the invariant network.

3.1.1 Objective Function. To overcome the lack of intrinsic correlation measures among heterogeneous entities, we embed entities into a common latent space where their semantics can be preserved. More specifically, each entity, such as a user, or a process in computer systems, is represented as a d-dimensional vector and will be automatically learned from the data. In the embedding space, the correlation of entities can be naturally computed by distance/similarity measures in the space, such as Euclidean distances, vector dot product, and so on. Compared with other distance/similarity metrics defined on sets, such as Jaccard similarity, the embedding method is more flexible and it has nice properties such as transitivity [35].

To address the challenge of heterogeneous relations among different entities, we use the *meta-path* [31] to model the heterogeneous relations. A meta-path is a path that connects entity types (labels) via a sequence of relations over a heterogeneous network [31]. For example, in a computer system, a *meta-path* can be a "Process-File-Process", or a "File-Process-Internet Socket". "Process-File-Process" denotes the relationship of two processes load the same file, and "File-Process-Internet Socket" denotes the relationship of a file loaded by a process who opened an Internet Socket. Notice that, the potential meta-paths induced from the heterogeneous network G_S can be infinite, but not every one is relevant and useful for the specific task of interest. Fortunately, there are some algorithms [5] proposed recently for automatically selecting the meta-paths for specific tasks.

Given a set of meta-paths $P = \{p_1, p_2, ...\}$, where p_i denotes the i-th meta-path and let $|P|$ be the number of meta-paths. We can construct $|P|$ graphs G_{p_i} by each time only extracting the corresponding meta-path p_i from the invariant network [31]. Let u_S be the vector representation of the entities in G_S. Then, we can model the relationship between two entities using their vector representations $u_S(i)$ and $u_S(j)$: $\|u_S(i) - u_S(j)\|_F^2 \approx S_G(i,j)$, where S_G is a weighted average of all the similarity matrices S_{p_i}: $S_G = \sum_{i=1}^{|P|} w_i S_{p_i}$, where w_i's are non-negative coefficients, and S_{p_i} is the similarity matrix constructed by calculating the pairwise shortest path between every two entities in A_{p_i}. Here, A_{p_i} is the adjacent matrix of the invariant network G_{p_i}. By using the shortest path in the graph, one can capture the long-term relationship between different entities [1]. Then, the objective function of the **EEM** model can be defined as:

$$\mathcal{L}_1^{(u_S,W)} = \sum_{i,j}^{n} \left(\|u_S(i) - u_S(j)\|_F^2 - S_G \right)^{\theta} + \Omega(u_S, W), \quad (1)$$

where $W = \{w_1, w_2, ..., w_{|P|}\}$, and $\Omega(u_S, W) = \lambda \|u_S\| + \lambda \|W\|$ is the generalization term [11], which prevents the model from over-fitting. And λ is the trade-off factor of the generalization term. In practice, we can choose θ as 1 or 2, which bears the resemblance to Hamming distance and Euclidean distance, respectively.

Putting everything together, we get:

$$\begin{aligned}\mathcal{L}_1^{(u_S,W)} &= \sum_{i,j}^{n} \left(\|u_S(i) - u_S(j)\|_F^2 - S_G \right)^{\theta} + \Omega(u_S, W) \\ &= \sum_{i,j}^{n} \left(\|u_S(i) - u_S(j)\|_F^2 - \sum_{i=0}^{|P|-1} w_i S_{p_i} \right)^{\theta} + \lambda \|u_S\| + \lambda \|W\|. \end{aligned} \quad (2)$$

Then, the optimized value $\{u_S, W\}^{opt}$ can be obtained by:

$$\{u_S, W\}^{opt} = \arg\min_{u_S, W} \mathcal{L}_1^{(u_S,W)}.$$

3.1.2 Inference Method. The objective function in Eq. 2 contains two sets of parameters: (1) u_S and (2) W. Then, we propose a two-step iterative method for optimizing $\mathcal{L}_1^{(u_S,W)}$, where the entity vector matrix u_S and the weight vector W for meta-paths mutually enhance each other. In the first step, we fix the weight vectors W and learn the best entity vector matrix u_S. In the second step, we fix the entity vector matrix u_S and learn the best weight vector W. Note that, based on the empirical experience, here we fix $\theta = 2$.

Fix W and learn u_S: when we fix W, then the problem is reduced to $\|u_S(i) - u_S(j)\|_F^2 \approx S_G(i,j)$, where S_G is a constant similarity matrix. Then, the optimization process becomes a traditional manifold learning problem. Fortunately, we can have a closed form to solve this problem, via the multi-dimensional scaling technique[11]. More specifically, to obtain such an embedding, we compute the eigenvalue decomposition of the following matrix: $-\frac{1}{2}HS_GH = U\Lambda U$, where H is the double centering matrix, U has columns as the eigenvectors and Λ is a diagonal matrix with eigenvalues. Then, u_S can be computed as:

$$u_S = U\sqrt[2]{\Lambda}. \quad (3)$$

Fix u_S and learn W: When fixing u_S, the problem is reduced to:

$$\begin{aligned}\mathcal{L}_1^{W} &= \sum_{i,j}^{n} \left(\|u_S(i) - u_S(j)\|_F^2 - \sum_{i=1}^{|P|} w_i S_{p_i} \right)^{\theta} + \lambda \|u_S\| + \lambda \|W\| \\ &= \sum_{i,j}^{n} \left(C_1 - \sum_{i=0}^{|P|} w_i S_{p_i} \right)^{\theta} + \lambda \|W\| + C_2, \end{aligned} \quad (4)$$

where $C_1 = \|u_S(i) - u_S(j)\|_F^2$ is a constant matrix, and $C_2 = \lambda \|E_S\|$ is a constant. Then, this function becomes a linear regression. So, we also have the close form solution for W: $W = (S_G^T S_G)^{-1} S_G C_1$.

After we get the embedding vectors u_S, then the relevance matrix $\mathbb{R}$ between different entities can be obtained as $\mathbb{R} = u_S u_S^T$. We can use a user-defined threshold to select the entities with high correlation with target domain for transferring. But this thresholding scheme is often suffered by the lack of domain knowledge. So here, we introduce a hypothesis test based method for automatically thresholding the selection of the entities.

For each entity in $\widehat{G}_T$, we first normalize all the scores by: $\mathbb{R}(i,:)_{norm} = (\mathbb{R}(i,:) - \mu)/\delta$, where $\mu = \overline{\mathbb{R}(i,:)}$ is the average value of $\mathbb{R}(i,:)$, and δ is the standard deviation of $\mathbb{R}(i,:)$. This standardized scores can be approximated with a Gaussian distribution. Then, the

threshold will be 1.96 for $P = 0.025$ (or 2.58 for $P = 0.001$) [11]. By using this threshold, one can filter out all the statistically irrelevant entities from the source domain, and transfer highly correlated entities to the target domain.

By combining the transferred entities and the original incomplete target network $\widehat{G}_T$, we get $\widetilde{G}_T$, a network that contains all the transferred entities, but missing the dependencies among them. Then, the next step is to construct the missing dependencies in $\widetilde{G}_T$.

3.2 DCM: Dependency Construction Model

To construct the missing dependencies/invariants in $\widetilde{G}_T$, there are two constraints need to be considered:

- *Smoothness Constraint*: The predicted dependency structure in $\overline{G}_T$ needs to be close to the dependency structure of the original incomplete target network $\widehat{G}_T$. The intuition behind this constraint is that the learned dependencies should keep the original dependencies of $\widehat{G}_T$ as intact as possible. This constraint guarantees that the constructed dependencies follow the behaviors of the target domain.
- *Consistency Constraint*: The inconsistency between $\widetilde{G}_T$ and $\widetilde{G}_S$ should be similar to the inconsistency between $\widehat{G}_T$ and $\widehat{G}_S$. Here, $\widetilde{G}_S$ and $\widehat{G}_S$ are the sub-graphs of G_S, which have the same entity set with $\widetilde{G}_T$ and $\widehat{G}_T$, respectively. This constraint guarantees that the target network learned by our model can keep the original domain difference with the source network.

Before we model the above two constraints, we first need a measure to evaluate the inconsistency between different domains. As aforementioned, invariant networks are normal profiles of their corresponding domains. So, we use the distance between different invariant networks to denote the domain inconsistency.

In this work, we propose a novel metric, named dynamic factor $F(\widetilde{G}_S, \widetilde{G}_T)$ between two invariant networks $\widetilde{G}_S$ and $\widetilde{G}_T$ from two different domains as:

$$F(\widetilde{G}_S, \widetilde{G}_T) = \frac{\left\|\widetilde{A}_S - \widetilde{A}_T\right\|}{|\widetilde{G}_S| * (|\widetilde{G}_S| - 1)/2} = \frac{2\left\|\widetilde{A}_S - \widetilde{A}_T\right\|}{n_S(n_S - 1)}, \tag{5}$$

where $n_s = |\widetilde{G}_S|$ is the number of entities in $\widetilde{G}_S$, $\widetilde{A}_S$ and $\widetilde{A}_T$ denote the adjacent matrix of $\widetilde{G}_S$ and $\widetilde{G}_T$, respectively, and $n_S(n_S - 1)/2$ denotes the number of edges of a fully connected graph with n_S entities [1]. Next, we introduce the Dependency Construction Model in details.

3.2.1 Modeling Smoothness Constraint. We first model the smoothness constraint as follows:

$$\begin{aligned}\mathcal{L}_{2.1}^{u_T} &= \left\|\sum_{i=1}^{n_S}\sum_{j=0}^{n_S-1}\left(u_T(i)u_T(j)^T - \widetilde{A}_T(i,j)\right)\right\|_F^2 + \lambda\left\|u_T\right\| \\ &= \left\|u_T u_T^T - \widetilde{A}_T\right\|_F^2 + \Omega(u_T),\end{aligned} \tag{6}$$

where u_T is the vector representation of the entities in $\overline{G}_T$, and $\Omega(u_T) = \lambda\left\|u_T\right\|$ is the regularization term.

Algorithm 1: TINET Algorithm

Input : G_S, $\widehat{G}_T$
Output: $\overline{G}_T$

```
Select a set of meta-paths from G_S.;
Extract |P| networks from G_S;
Calculate all the similarity matrix S_{p_i};
\* Entity Estimation Process as introduced in Section 3.1*\;
while Convergence do
    Calculate U and Λ;
    u_S = U ∛Λ;
    Calculate S_G and C_1;
    W = (S_G^T S_G)^{-1} S_G C_1;
end
Construct G̃_T;
\* Dependency Construction Process as introduced in Section 3.2*\;
while Convergence do
    Update u_T using the gradient of Eq. 10;
end
Construct Ḡ_T;
```

3.2.2 Modeling Consistency Constraint. We then model the consistency constraint as follows:

$$\mathcal{L}_{2.2}^{(u_T)} = \left\|F(u_T u_T^T, \widetilde{A}_S) - F(\widehat{A}_S, \widehat{A}_T)\right\|_F^2 + \Omega(u_T), \tag{7}$$

where $F(*,*)$ is the dynamic factor as we defined before. Then, putting Eq. 5 and $\Omega(u_T)$ into Eq. 7, we get:

$$\begin{aligned}\mathcal{L}_{2.2}^{E_T} &= \left\|F(u_T u_T^T, \widetilde{G}_S) - F(\widehat{G}_S, \widehat{G}_T)\right\|_F^2 + \Omega(u_T) \\ &= \left\|\frac{2\left\|u_T u_T^T - \widetilde{A}_S\right\|}{n_s(n_S - 1)} - F(\widehat{G}_S, \widehat{G}_T)\right\|_F^2 + \Omega(u_T) \\ &= \left\|\frac{2\left\|u_T u_T^T - \widetilde{A}_S\right\|}{n_s(n_s - 1)} - C_3\right\|_F^2 + \Omega(u_T),\end{aligned} \tag{8}$$

where $C_3 = F(\widehat{G}_S, \widehat{G}_T)$.

3.2.3 Unified Model. By putting the two constraints together, we propose the unified model for dependency construction as follows:

$$\begin{aligned}\mathcal{L}_2^{u_T} &= \mu\mathcal{L}_{2.1}^{u_T} + (1-\mu)\mathcal{L}_{2.2}^{u_T} \\ &= \mu\left\|u_T u_T^T - \widetilde{A}_T\right\|_F^2 + (1-\mu)\left\|\frac{2\left\|u_T u_T^T - \widetilde{A}_S\right\|}{n_S(n_S-1)} - C_3\right\|_F^2 + \Omega(u_T)\end{aligned} \tag{9}$$

The first term of the model incorporates the *smoothness constraint* component, which keeps the u_T closer to the target domain knowledge existed in $\widetilde{G}_S$. The second term considers the *consistency constraint*, that is the inconsistency between $\widetilde{G}_T$ and $\widetilde{G}_S$ should be similar to the inconsistency between $\widehat{G}_T$ and $\widehat{G}_S$. μ and λ are important parameters, which capture the importance of each term, and we will discuss these parameters in Section 3.3.1.

To optimize the model as in Eq. 9, we use stochastic gradient descent method [11]. The derivative on u_T is given as:

$$\frac{1}{2}\frac{\partial \mathcal{L}_2^{u_T}}{\partial E_T} = \mu u_T(u_T u_T^T - \widetilde{A}_T) + (1-\mu)u_T \left\| \frac{2\left\|u_T u_T^T - \widetilde{A}_S\right\|}{n_S(n_S-1)} - C_3 \right\| + u_T \tag{10}$$

3.3 Overall Algorithm

The overall algorithm is then summarized as Algorithm 1. In the algorithm, line 5 to line 11 implements the Entity Estimation Model, and line 13 to 16 implements the Dependency Construction Model.

3.3.1 Parameter Setting. There are two parameters, λ and μ, in our model. For λ, as in [11, 31], it is always assigned manually based on the experiments and experience. For μ, when a large number of entities are transferred to the target domain, a large μ can improve the transferring result, because we need more information to be added from the source domain. On the other hand, when only a small number of entities are transferred to the target domain, then a larger μ will bias the result. Therefore, the value of μ depends on how many entities are transferred from the source domain to the target domain. In this sense, we can use the proportion of the transferred entities in $\widetilde{G}_T$ to calculate μ. Given the entity size of $\widetilde{G}_T$ as $|\widetilde{G}_T|$, the entity size of $\widehat{G}_T$ as $|\widehat{G}_T|$, then μ can be calculated as:

$$\mu = (|\widetilde{G}_T| - |\widehat{G}_T|)/|\widetilde{G}_T|. \tag{11}$$

The experimental results in Section 4.6 demonstrate the effectiveness of the proposed parameter selection method.

3.3.2 Complexity Analysis. As shown in Algorithm 1, the time for learning our model is dominated by computing the objective functions and their corresponding gradients against feature vectors. For the Entity Estimation Model (**EEM**), the time complexity of computing the u_S in Eq. 3 is bounded by $O(d_1 n)$, where n is the number of entities in G_S, and d_1 is the dimension of the vector space of u_S. The time complexity for computing W is also bounded by $O(d_1 n)$. So, suppose the number of training iterations for **EEM** is t_1, then the overall complexity of **EEM** model is $O(t_1 d_1 n)$. For the Dependency Construction Model (**DCM**), the time complexity of computing the gradients of $\mathcal{L}_2$ against u_T is $O(t_2 d_2 n)$, where t_2 is the number of iterations and d_2 is the dimensionality of feature vector. As shown in our experiment (see Section 4.5), t_1, t_2, d_1, and d_2 are all small numbers, so we can regard them as a constant, say C. Thus, the overall complexity of our method is $O(Cm)$, which is linear with the size of the entity set. This makes our algorithm practicable for large-scale datasets.

4 EXPERIMENTS

In this section, we evaluate **TINET** using synthetic data and real system surveillance data collected from enterprise networks.

4.1 Comparing Methods

We compare **TINET** with the following methods:

NT: This method directly uses the original incomplete target network without knowledge transfer. In other words, the estimated target network $\overline{G}_T = \widehat{G}_T$.

DT: This method directly combines the source network and the incomplete target network. In other words, the estimated target network $\overline{G}_T = \widetilde{G}_S + \widehat{G}_T$.

RW-DCM: This method uses random walk, instead of the **EEM** model (see Section 3.1)), to evaluate the correlations between entities and perform entity estimation, and then apply the **DCM** model to perform the perform the dependency construction. The random walk method has shown to be a powerful way for relevance search in a graph [19].

EEM-CMF: Collective matrix factorization (**CMF**) is a widely-used transfer learning framework for constructing links/dependencies in graphs [28]. However, it can't be directly applied for transferring system entities, which is required in this work. Thus, we modify **CMF** by adding the entity estimation component. **EEM-CMF** first uses **EEM** model (see Section 3.1) to perform the entity estimation, and then uses **CMF** to perform the dependency construction.

4.2 Evaluation Metrics

Since in **TINET** algorithm, we use hypothesis-test for thresholding the selection of entities and dependencies, similar to [11, 24], we use F1-score to evaluate the hypothesis-test accuracy of all the methods. F1-score is the harmonic mean of precision and recall. In our experiment, the final F1-score is calculated by averaging the entity F1-score and dependency/edge F1-score.

To calculate the precision (recall) of the entity (edge), we compare the estimated entity (edge) set with the ground-truth. The precision and recall can be calculated as: $Precision = \frac{N_C}{N_E}, Recall = \frac{N_C}{N_T}$, where N_C is the number of correctly estimated entities (edges), N_E is the number of total estimated entities (edges), and N_T is the number of the ground-truth entities (edges).

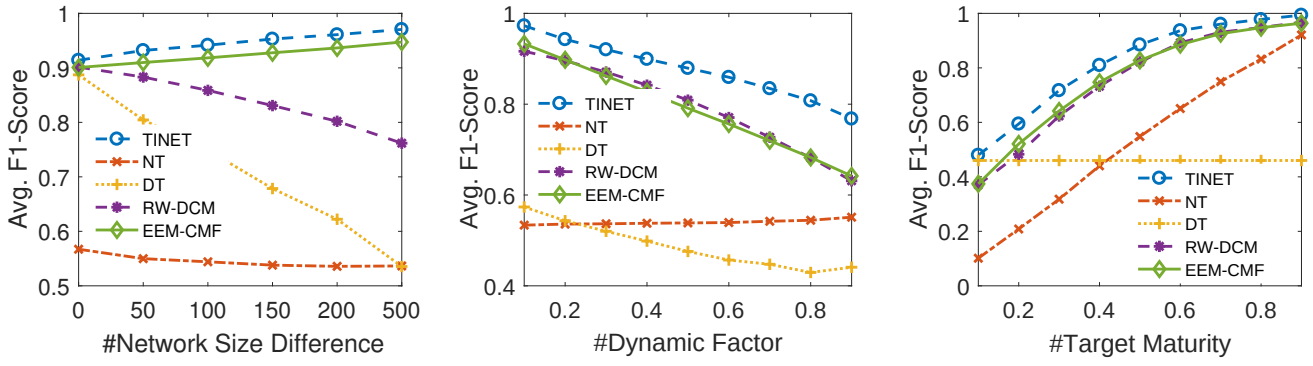
4.3 Synthetic Experiments

We first evaluate the **TINET** on synthetic graph datasetsto have a more controlled setting for assessing algorithmic performance.

4.3.1 Synthetic Data Generation. According to our problem statement (see Section 2), we need a source network G_S and an incomplete target network $\widehat{G}_T$ as the input, and a ground-truth target network G_T to evaluate the performance in each synthetic data. Based on the relationships among these three networks, we define the following parameters to guide our synthetic data generation:

- **Graph size**: It is defined as the number of entities for an invariant network. In particular, we use $|G_S|$ to control the size of the source network and $|\widehat{G}_T|$ to control the size of the incomplete target network.
- **Dynamic factor** F: It controls the difference between the networks G_S and G_T (see Eq. 5 for definition).
- **Graph maturity score** M: It is defined as the percentage of entities and edges of the ground-truth graph G_T that have already been learned in $\widehat{G}_T$. Here, graph maturity score is used for simulating the period of learning time of $\widehat{G}_T$ to reach the maturity in the real system.

For a given $|G_S|$, $|\widehat{G}_T|$, F, and M, we generate the synthetic data as follows: We first randomly generate an undirected graph [32] as the source network G_S with the size of $|G_S|$; Then, to get a heterogeneous graph, we randomly assign one of the three different



(a) Varying graph size **(b) Varying dynamic factor** **(c) Vary graph maturity**

Figure 4: Performance on synthetic data. TINET outperforms all the other methods in all the datasets.

labels to each entity[2]; We further construct the target network G_T by randomly adding/deleting $F = d\%$ of the edges and entities from G_S. Finally, we randomly select $M = c\%$ of entities and edges from G_T to form $\widehat{G}_T$.

*4.3.2 How Does **TINET**'s Performance Scale with Graph Size?* We first explore how the **TINET**'s performance changes with the graph size difference between $|G_S|$ and $|\widehat{G}_T|$. Here, we set the maturity score M to be 50%, the dynamic factor F to be 10%, and the size of the incomplete target network $|\widehat{G}_T|$ to be $0.9K$. Then, we increase the source network size $|G_S|$ from $0.9K$ to $1.4K$. From Fig. 4a, we observe that with the increase of the size difference $|G_S| - |\widehat{G}_T|$, the performances of **DT** and **RW-DCM** are getting worse. This is due to the poor ability of **DT** and **RW-DCM** for extracting useful knowledge from the source domain. In contrast, the performance of **TINET** and **EEM-CMF** increases with the size differences. This demonstrates the great capability of the **EEM** model for entity knowledge extraction.

*4.3.3 How Does **TINET**'s Performance Scale with Domain Dynamic Factor?* We now vary the dynamic factor F to understand its impact on the **TINET**'s performance. Here, the graph maturity score M is set to be 50%, and two network sizes $|G_S|$ and $|\widehat{G}_T|$ are set to be $1.2K$ and $0.6K$, respectively. Fig. 4b shows that the performances of all the methods go down with the increase of the dynamic factor. This is expected because transferring the invariant network from a very different domain will not work well. On the other hand, the performances of **TINET**, **RW-DCM**, and **EEM-CMF** only decrease slightly with the increase of the dynamic factor. Since **RW-DCM** and **EEM-CMF** are variants of the **TINET** method, this demonstrates that the two sub-models of the **TINET** method are both robust to large dynamic factors.

*4.3.4 How Does **TINET**'s Performance Scale with Graph Maturity?* Third, we explore how the graph maturity score M impacts the performance of **TINET**. Here, the dynamic factor F is fixed to be 0.2. The graph sizes $|G_S|$ and $|\widehat{G}_T|$ are set to be $1.2K$ and $0.6K$, respectively. Fig. 4c shows that with the increase of the M, the performances of all the methods are getting better. The reason is straightforward: with the maturity score increases, the challenge of domain difference for all the methods is becoming smaller. In addition, our **TINET** and its variants **RW-DCM**, and **EEM-CMF** perform much better than **DT** and **NT**. This demonstrates the great ability of the sub-models of **TINET** for knowledge transfer. Furthermore, **TINET** still achieves the best performance.

[2]Similar results have been achieved in synthetic networks with more than three labels.

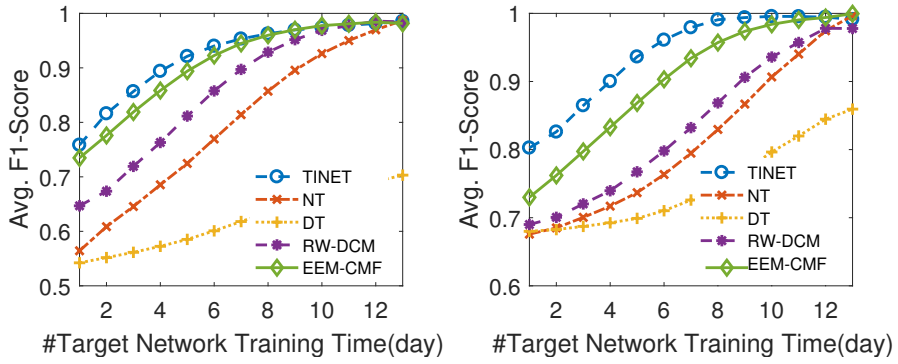
4.4 Real-World Experiments

Two real-world system monitoring datasetsare used in this experiment. The data is collected from an enterprise network system composed of 47 Linux machines and 123 Windows machines from two departments, in a time span of 14 consecutive days. In both datasets, we collect two types of system events: (1) communications between processes, and (2) system activity of processes sending or receiving Internet connections to/from other machines at destination ports. Three different types of system entities are considered: (1) processes, (2) Unix domain sockets, and (3) Internet sockets. The sheer size of the Windows dataset is around 7.4 Gigabytes, and the Linux dataset is around 73.5 Gigabytes. Both Windows and Linux datasets are split into a source domain and a target domain according to the department name. Knowledge transferring between a Linux system and a Windows system is not considered in this work because the behaviors of these two systems are very different.

In this experiment, we construct one target network $\widehat{G}_T$ per day by increasing the learning time daily. The final graph is the one learned for 14 days. From Fig. 5, we observe that for both Windows and Linux datasets, with the increase of the training time, the performances of all the algorithms are getting better. On the other hand, compared with all the other methods, **TINET** achieves the best performance on both Windows and Linux datasets. In addition, our proposed **TINET** algorithm can make the invariant network deplorable in less than four days, instead of two weeks or longer by directly learning on the target domain.

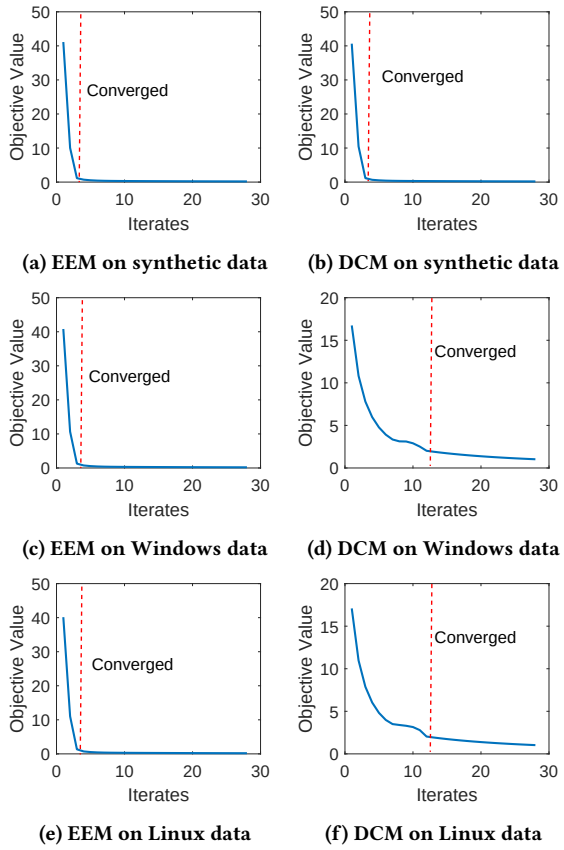
4.5 Convergence Analysis

As described in Section 3.3.2, the performance bottleneck of **TINET** model is the learning process of the two sub-models: **EEM** (Entity Estimation Model) and **DCM** (Dependency Construction Model). In



(a) Windows dataset (b) Linux dataset

Figure 5: Performance on real-world data. TINET outperforms all the other methods in all datasets.



(a) EEM on synthetic data (b) DCM on synthetic data

(c) EEM on Windows data (d) DCM on Windows data

(e) EEM on Linux data (f) DCM on Linux data

Figure 6: Performance on convergence. TINET converges very fast, which makes it applicable for large-scale systems.

this section, we use both synthetic and real-world data to validate the convergence speed of both sub-models.

For the synthetic data, we choose the one with the dynamic factor $F = 0.2$, the invariant network sizes $|G_S = 1.2K|$ and $|\widehat{G}_T| = 0.6K$, and the graph maturity $M = 50\%$. For the two real-world datasets, we fix the target network learning time as 4 days.

From Fig. 6, we can see that in all three datasets, **TINET** converges very fast (*i.e.*, with less than 10 iterations). This makes our model applicable for the real-world large-scale systems.

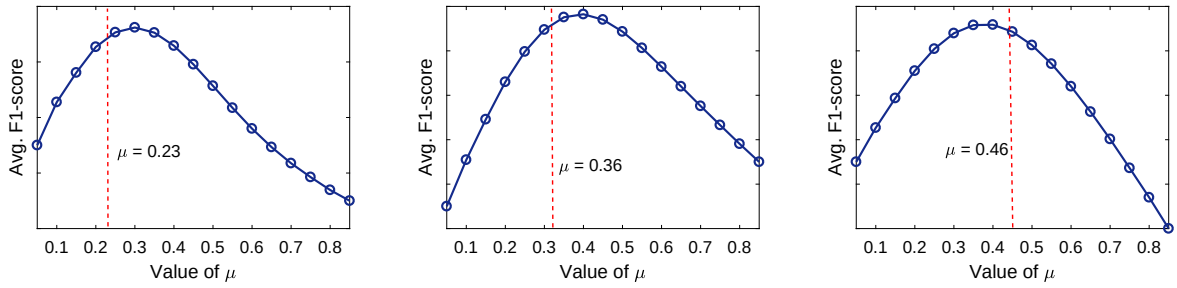
4.6 Parameter Study

In this section, we study the impact of parameter μ (see Eq. 9) by using both synthetic and real-world data. As shown in Fig. 7, when the value of μ is too small or too large, the results are not good, because μ controls the leverage between the source domain information and target domain information. The extreme value of μ (too large or too small) will bias the result. On the other hand, the μ value calculated by Eq. 11 is 0.23 for the synthetic dataset, 0.36 for the Windows dataset, and 0.46 for the Linux dataset. And Fig. 7 shows the best results just appear around these three values. This demonstrates that our proposed method for setting the μ value is very effective, which successfully addresses the parameter pre-assignment issue.

4.7 Case Study on Intrusion Detection

In this section, we evaluate the **TINET**'s performance in a real commercial enterprise security system (see Fig. 2) for intrusion detection. The same security system has been deployed in two companies: a Japanese electric company and an American IT company. We obtain one invariant network from the IT company after 30 days' training, and an incomplete invariant network from the electric company after only 3 days' training. Then, we apply **TINET** and other baseline methods to learn the final invariant network by leveraging the well-trained invariant network from the IT company. After learning the invariant networks, the same graph-based intrusion detection approach [8] is applied during the testing period to identify the causal anomalies (intrusions) via analyzing patterns in the invariant networks.

In the one-day testing period, we try 10 different types of attacks [18], including Snowden attack, ATP attack, botnet attack, Sniffer Attack and *etc.*, which resulted in 30 ground-truth alerts. All other alerts reported during the testing period are considered as false positives.



(a) Synthetic dataset **(b) Windows dataset** **(c) Linux dataset**

Figure 7: Sensitivity analysis on parameter μ. Red dashed lines denote the values that computed by Eq. 11.

Table 2 shows the intrusion detection results in the electric company using the invariant networks generated by different transfer learning methods and the real invariant network generated after 30 days' training from the electric company. From the results, we can clearly see that **TINET** outperforms all the other baseline methods by at least 15% in precision and 16% in recall. On the other hand, the performance of the invariant network (3 days' model) constructed by **TINET** is very close to the ground truth model (30 days' model). This means, by using **TINET**, we can achieve similar performance in one-tenth training time, which is of great importance to mission-critical environments such as nuclear plants.

Table 2: Intrusion detection performance

Method	Precision	Recall
NT	0.01	0.10
DT	0.15	0.30
RW-DCM	0.48	0.57
EEM-CMF	0.53	0.60
TINET	0.68	0.76
Real 30 days' invariant network	**0.70**	**0.76**

5 RELATED WORK

5.1 Transfer Learning

Transfer learning [2, 27] has been widely studied in recent years. Most of the traditional transfer learning methods focus on numerical data [3, 7, 30]. When it comes to graph (network) structured data, there is less existing work. In [10], the authors presented *TrGraph*, a novel transfer learning framework for network node classification. *TrGraph* leverages information from the auxiliary source domain to help the classification process of the target domain. In one of their earlier work, a similar approach was proposed [9] to discover common latent structure features as useful knowledge to facilitate collective classification in the target network. In [12], the authors proposed a framework to propagates the label information from the source domain to the target domain via the example-feature-example tripartite graph. Transfer learning has also been applied to the deep neural network structure. In [4], the authors introduced *Net2Net*, a technique for rapidly transferring the information stored in one neural net into another. *Net2Net* utilizes function preserving transformations to transfer knowledge from neural networks. Different from existing methods, we aim to expedite the invariant network learning process through knowledge transfer.

5.2 Link Prediction and Relevance Search

Graph link prediction is a well-studied research topic [14, 22]. In [34], Ye *et al.* presented a transfer learning algorithm to address the edge sign prediction problem in signed social networks. Because edge instances are not associated with a pre-defined feature vector, this work was proposed to learn the common latent topological features shared by the target and source networks, and then adopt an AdaBoost-like transfer learning algorithm with the instance weighting to train a classifier. Collective matrix factorization [28] is another popular technique that can be applied to detect mission links by combining the source domain and target domain graphs. However, all the existing link prediction methods cannot deal with dynamics between different domains in our problem.

Finding relevant nodes or similarity search in graphs is also related to our work. Many different similarity metrics have been proposed such as Jaccard coefficient, Pearson correlation coefficient [1], and random walks [19, 29]. However, none of these similarity measures consider the multiple relations exist in the data. Recent advances in heterogeneous information networks [31] have offered several similarity measures for heterogeneous relations, such as meta-path and relation path [23, 25]. However, these methods cannot deal with domain variety problems.

6 CONCLUSION

In this paper, we introduce an important and challenging problem of transfer learning on invariant networks. We propose **TINET**, a transfer learning framework for accelerating invariant network learning. By leveraging entity embedding and constrained optimization techniques, **TINET** can effectively extract useful knowledge (*e.g.*, entity and dependency relations) from the source domain, and transfer it to the target network. We evaluate the proposed algorithm using extensive experiments on both synthetic and real-world datasets. The experiment results convince us of the effectiveness and efficiency of our approach. We also apply **TINET** to a real enterprise security system for intrusion detection. Our method can

achieve superior detection performance at least 20 days lead-lag time in advance with more than 75% accuracy.

ACKNOWLEDGMENTS

This work was supported by National Science Foundation IIS-1652131, RI-1718478, AFOSR-YIP FA9550-18-1-0152, Amazon Research Award, and a GPU grant from NVIDIA.

REFERENCES

[1] John Adrian Bondy and Uppaluri Siva Ramachandra Murty. 1976. *Graph theory with applications*. Vol. 290. New York: Elsevier.

[2] Bin Cao, Nathan N Liu, and Qiang Yang. 2010. Transfer learning for collective link prediction in multiple heterogenous domains. In *Proceedings of the 27th International Conference on Machine Learning*. 159–166.

[3] Rita Chattopadhyay, Wei Fan, Ian Davidson, Sethuraman Panchanathan, and Jieping Ye. 2013. Joint transfer and batch-mode active learning. In *International Conference on Machine Learning*. 253–261.

[4] Tianqi Chen, Ian Goodfellow, and Jonathon Shlens. 2016. Net2net: Accelerating learning via knowledge transfer. In *Proceedings of the 4th International Conference on Learning Representations*.

[5] Ting Chen and Yizhou Sun. 2017. Task-Guided and Path-Augmented Heterogeneous Network Embedding for Author Identification. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*. ACM, 295–304.

[6] Wei Cheng, Kai Zhang, Haifeng Chen, Guofei Jiang, Zhengzhang Chen, and Wei Wang. 2016. Ranking causal anomalies via temporal and dynamical analysis on vanishing correlations. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 805–814.

[7] Wenyuan Dai, Qiang Yang, Gui-Rong Xue, and Yong Yu. 2007. Boosting for transfer learning. In *Proceedings of the 24th International Conference on Machine Learning*. ACM, 193–200.

[8] Boxiang Dong, Zhengzhang Chen, Hui (Wendy) Wang, Lu-An Tang, Kai Zhang, Ying Lin, Zhichun Li, and Haifeng Chen. 2017. Efficient discovery of abnormal event sequences in enterprise security systems. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management (CIKM'17)*. 707–715.

[9] Meng Fang, Jie Yin, and Xingquan Zhu. 2013. Transfer learning across networks for collective classification. In *2013 IEEE 13th International Conference on Data Mining*. IEEE, 161–170.

[10] Meng Fang, Jie Yin, Xingquan Zhu, and Chengqi Zhang. 2015. TrGraph: Cross-network transfer learning via common signature subgraphs. *IEEE Transactions on Knowledge and Data Engineering* 27, 9 (2015), 2536–2549.

[11] Jiawei Han, Jian Pei, and Micheline Kamber. 2011. *Data mining: concepts and techniques*. Elsevier.

[12] Jingrui He, Yan Liu, and Richard Lawrence. 2009. Graph-based transfer learning. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management*. ACM, 937–946.

[13] Miao He and Junshan Zhang. 2011. A dependency graph approach for fault detection and localization towards secure smart grid. *IEEE Transactions on Smart Grid* 2, 2 (2011), 342–351.

[14] Jake M Hofman, Amit Sharma, and Duncan J Watts. 2017. Prediction and explanation in social systems. *Science* 355, 6324 (2017), 486–488.

[15] Guofei Jiang, Haifeng Chen, and Kenji Yoshihira. 2006. Discovering likely invariants of distributed transaction systems for autonomic system management. In *IEEE International Conference on Autonomic Computing*. IEEE, 199–208.

[16] Guofei Jiang, Haifeng Chen, and Kenji Yoshihira. 2006. Modeling and tracking of transaction flow dynamics for fault detection in complex systems. *IEEE Transactions on Dependable and Secure Computing* 3, 4 (2006), 312–326.

[17] Guofei Jiang, Haifeng Chen, and Kenji Yoshihira. 2007. Efficient and scalable algorithms for inferring likely invariants in distributed systems. *IEEE Transactions on Knowledge and Data Engineering* 19, 11 (2007).

[18] Anita K Jones and Robert S Sielken. 2000. Computer system intrusion detection: A survey. *Computer Science Technical Report* (2000), 1–25.

[19] U Kang, Hanghang Tong, and Jimeng Sun. 2012. Fast random walk graph kernel. In *Proceedings of the 2012 SIAM International Conference on Data Mining*. SIAM, 828–838.

[20] Samuel T King, Z Morley Mao, Dominic G Lucchetti, and Peter M Chen. 2005. Enriching intrusion alerts through multi-host causality. In *Proceedings of the 2005 Network and Distributed System Security Symposium (NDSS)*.

[21] Ronald L Krutz and Russell Dean Vines. 2010. *Cloud security: A comprehensive guide to secure cloud computing*. Wiley Publishing.

[22] David Liben-Nowell and Jon Kleinberg. 2007. The link-prediction problem for social networks. *Journal of the Association for Information Science and Technology* 58, 7 (2007), 1019–1031.

[23] Chen Luo, Renchu Guan, Zhe Wang, and Chenghua Lin. 2014. HetPathMine: A novel transductive classification algorithm on heterogeneous information networks. In *Proceedings of the 36th European Conference on Information Retrieval*. Springer, 210–221.

[24] Chen Luo, Jian-Guang Lou, Qingwei Lin, Qiang Fu, Rui Ding, Dongmei Zhang, and Zhe Wang. 2014. Correlating events with time series for incident diagnosis. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 1583–1592.

[25] Chen Luo, Wei Pang, Zhe Wang, and Chenghua Lin. 2014. Hete-cf: Social-based collaborative filtering recommendation using heterogeneous relations. In *2014 IEEE International Conference on Data Mining*. IEEE, 917–922.

[26] Jingchao Ni, Wei Cheng, Kai Zhang, Dongjin Song, Tan Yan, Haifeng Chen, and Xiang Zhang. 2017. Ranking causal anomalies by modeling local propagations on networked systems. In *2017 IEEE International Conference on Data Mining (ICDM)*. IEEE, 1003–1008.

[27] Sinno Jialin Pan and Qiang Yang. 2010. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering* 22, 10 (2010), 1345–1359.

[28] Ajit P Singh and Geoffrey J Gordon. 2008. Relational learning via collective matrix factorization. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 650–658.

[29] Jimeng Sun, Huiming Qu, Deepayan Chakrabarti, and Christos Faloutsos. 2005. Relevance search and anomaly detection in bipartite graphs. *ACM SIGKDD Explorations Newsletter* 7, 2 (2005), 48–55.

[30] Qian Sun, Mohammad Amin, Baoshi Yan, Craig Martell, Vita Markman, Anmol Bhasin, and Jieping Ye. 2015. Transfer learning for bilingual content classification. In *Proceedings of the 21st ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2147–2156.

[31] Yizhou Sun and Jiawei Han. 2012. Mining heterogeneous information networks: principles and methodologies. *Synthesis Lectures on Data Mining and Knowledge Discovery* 3, 2 (2012), 1–159.

[32] Douglas Brent West. 2001. *Introduction to graph theory*. Vol. 2. Prentice hall Upper Saddle River.

[33] Zhang Xu, Zhenyu Wu, Zhichun Li, Kangkook Jee, Junghwan Rhee, Xusheng Xiao, Fengyuan Xu, Haining Wang, and Guofei Jiang. 2016. High fidelity data reduction for big data security dependency analyses. In *Proceedings of ACM SIGSAC Conference on Computer and Communications Security*. ACM, 504–516.

[34] Jihang Ye, Hong Cheng, Zhe Zhu, and Minghua Chen. 2013. Predicting positive and negative links in signed social networks by transfer learning. In *Proceedings of the 22nd International Conference on World Wide Web*. ACM, 1477–1488.

[35] Kai Zhang, Qiaojun Wang, Zhengzhang Chen, Ivan Marsic, Vipin Kumar, Guofei Jiang, and Jie Zhang. 2015. From categorical to numerical: Multiple transitive distance learning and embedding. In *Proceedings of the 2015 SIAM International Conference on Data Mining*. SIAM, 46–54.