Noisy Subgraph Isomorphisms on Multiplex Networks

Hui Jin*, Xie He†, Yanghui Wang*, Hao Li*, and Andrea L. Bertozzi*
* University of California, Los Angeles, Los Angeles, California
† University of North Carolina, Chapel Hill, Chapel Hill, North Carolina

***Abstract*—We address the problem of finding noisy subgraph isomorphisms on large multiplex networks. Our goal is to find as many subgraph matches as possible within a noise tolerance. We propose a novel approach based on the well-known A* search algorithm. Our approach employs new heuristics to estimate the number of missing edges of subgraph matches. This method is verified on one of the synthetic multiplex networks from the Modeling Adversarial Activity program of the Defense Advanced Research Projects Agency.**

I. Introduction

Multiplex networks are a mathematical framework for representing complicated abstract data with a set of nodes and various types of interactions between them. The interactions are encoded by different types of edges; each type stands for a channel in a multiplex network. The subgraph isomorphism problem looks for copys of a provided template graph inside a larger world graph. Solving this problem finds applications in many disciplines; one specific example is to find cells, proteins, or molecules with certain structures [1], [14], [13].

Real-world datasets are often noisy, meaning the edge information may be imprecise, either in the world or the template graph. From the application standpoint of view, one could consider noises such as node/edge insertions, deletions, and mismatches. In this paper, we address the problem of noisy subgraph matching of templates in large multiplex networks. Namely, given a small template and a large world graph, both of which are multiplex networks, we want to find the best possible matches of the template within the world graph. Our work builds upon previous work in [16], which introduces a suite of filtering algorithms. The algorithms assume all world nodes are initially candidates to each template node, and eliminate them according to certain properties of the template graph. For example, the statistics filter in [16] uses the in-degree and out-degree of each node in the template graph as a criterion; the filter eliminates world nodes with less in/out-degree than their corresponding template node, from the candidate list of this template node. We generalize these filters to the noisy case and develop an approach, based on the A* search algorithm, for finding all noisy subgraph isomorphisms, within a tolerance level. We focus on solving problems with missing edges in the world graph and extra edges in the template graph, in which no exact matches can be found. We implement our algorithm on the equivalence classes [18], which we introduce in details in Section V, to speed up the search. The experiments are performed on an Intel Xeon Gold 6136 processor with 3 GHz, 25 MB of cache, and 128 GB of memory.

This paper is structured as follows:

- In Section II and Section III, we discuss related work and our contributions, respectively;
- In Section IV, we talk about the problem setup of the noisy case;
- In Section V, we propose heuristics to estimate the number of missing edges for partial matches; we also explain how to use the A* search algorithm to find all Noisy Subgraph Isomorphisms (NSIs);
- We demonstrate experimentally in Section VI that our approach allows us to enumerate many NSIs for a world graph with 22K nodes and a template with 74 nodes.

II. Related Work

Many methods have been proposed to find exact subgraph isomorphisms, e.g. VF2 [4], VF3 [3], LAD [20], Graph-Indexing [10] and Ullmann's earlier papers such as [22]. While many prior works are based on constraint propagation on single channel networks, [16] presents a suite of filtering methods for finding subgraph isomorphisms on multiplex networks. Based on the filters, [16] aims to understand the entire solution space — that is the set of all subgraph isomoprhisms — rather than focusing on finding one of them. However, real-world datasets can have noise, requiring methods for finding noisy subgraph isomorphisms; these are subgraphs of the given world graph that are a few edges short of an exact match. Recent works such as [25] and [24] explore the noisy subgraph isomorphism problem on single channel networks with small world graphs.

The paper [6] focuses on datasets with 10^4 edges; the work [7] uses a graph of 100 nodes; the paper [11] has a world graph with at most 4×10^3 edges. Some inexact subgraph matching methods such as [2] have been proposed to handle large single channel networks, but none of these has been extended to large multiplex networks. Very few works consider the noisy subgraph matching on large multiplex datasets with more than 10^6 edges. A recent paper [21] proposes an optimization-based method for large multiplex networks, and the authors demonstrate the effectiveness of their method for a certain

This material is based on research sponsored by the Air Force Research Laboratory and DARPA under agreement number FA8750-18-2-0066. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon.

978-1-7281-0858-2/19/$31.00 ©2019 IEEE

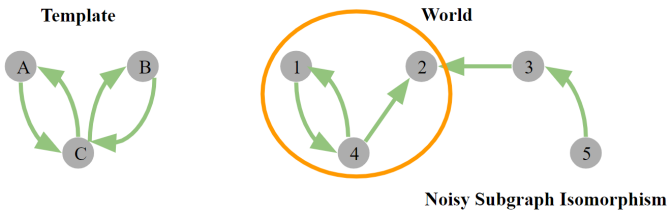

Fig. 1: Given the template and world networks above, the noisy subgraph isomorphism is the subgraph of the world induced by *nodes* $(1, 2, 4)$. Note that there is one missing edge in the NSI compared to the template.

noise model. However, their method is only suited for finding one subgraph isomorphism rather than identifying all subgraph matches within a certain noise level.

The A* search algorithm is a well-known heuristic shortest-path algorithm proposed by [9]. It uses an evaluation function for each node and avoids expanding paths that are already expensive. References [26], [15], [23], [8], [17] and many others have been applying A* search algorithms to calculate graph edit distance and thus solve both exact and inexact graph isomorphism problems. However, these methods focus on graph isomorphisms instead of subgraph isomorphisms; also, they only focus on single channel networks and have only been tested on small datasets, usually with around 100 nodes in the world graph. Thus, their heuristics are not designed for large multiplex networks.

We develop an algorithm for large multiplex networks and the noisy subgraph isomorphism problem while tracking the number of missing edges, with the goal of finding all the noisy subgraph isomorphisms in the world graph. Our approach is based on the A* search algorithm and we borrow the idea of the statistics filter and topology filter from [16], modifying them to best adjust to the noise.

III. Contribution

We develop novel heuristics and apply the A* search algorithm to find noisy subgraph isomorphisms on large multiplex networks. As in [18] our ultimate goal is to understand the solution space (within some tolerance level) rather than just finding one or several isomorphisms. We also propose two different noise models, one with missing edges in the world graph and another one with added edges in the template graph. In both cases we can identify at least one noisy subgraph isomorphism on a large multiplex network.

IV. Problem Set Up

A *graph* G is an ordered triple $\big(V(G), E(G)\big)$ where $V(G)$ is the set of *vertices* or *nodes* of a graph, $E(G) \subseteq V(G) \times V(G)$ the set of *edges*. Graphs of this form are referred to as *simple directed graphs*. We only consider directed graphs in this paper, so if $(u, v) \in E(G)$ does not necessarily imply $(v, u) \in E(G)$.

In this paper, we focus on multiplex graphs. A multiplex graph is defined by $\big(V(G), C_G\big)$ where $V(G)$ is the set of *vertices* and $C_G^c(u, v)$ describes the number of edges from vertex u to vertex v in a specific channel c. The central topic of interest in this paper is the subgraph isomorphism.

Definition 1 (*Subgraph isomorphism*)**.** *Given graphs* $T = (V(T), E(T))$ *and* $W = (V(W), E(W))$*, and a map* $F : V(T) \longrightarrow V(W)$*, we say that* F *is a subgraph isomorphism if it is injective and edge-preserving, that is, for each* $(u, v) \in E(G)$*, we have that* $(F(u), F(v)) \in E(H)$*. In the case of multiplex graphs, we need* $C_T^c(u, v) \leq C_W^c(F(u), F(v))$ *in each channel* c*.*

We refer to the graph T in the above definition as the *template graph* and the graph W as the *world graph*. Any subgraph of W that is isomorphic to T is denoted as a *signal* of T in W.

However, in the noisy case, we may not find an exact match of a template graph in a world graph; edges in the signals may be missing. We define missing edges in the following way:

Definition 2 (*Missing Edges*)**.** *Given graphs* T *and* W*, and a map* $F : V(T) \longrightarrow V(W)$*, for a pair of nodes* (u, v)*, we say that* $(u, v) \in E(T)$ *is a missing edge if* $(F(u), F(v)) \notin E(W)$*. In the case of multiplex graphs, we say that the number of missing edges in pair* (u, v) *is*

$$\mathrm{miss}(F, (u, v)) = \sum_c \max\left\{C_T^c(u, v) - C_W^c(F(u), F(v)), 0\right\}.$$

From now on, we only consider multiplex graphs. Since our goal is to find subgraph matches with fewest number of missing edges, we define the cost of a subgraph match as follows:

Definition 3 (*Cost of Subgraph Match*)**.** *Given graphs* T *and* W*, and a subgraph match* $F : V(T) \longrightarrow V(W)$*, the cost of subgraph match* F *is defined by*

$$g(T, W, F) = \sum_{(u,v) \in E(T)} \mathrm{miss}(F, (u, v)).$$

Note that to solve the noisy subgraph isomorphism problem, different costs have been proposed in the literature. For example, the paper [21] defines the cost by the Frobenius norm of the difference between graphs adjacency matrices. Our definition is suitable for the case where we allow more edges in the world graph than the template graph.

Recall that we want to find a subgraph match with the lowest *cost*, which we refer to as a *Noisy Subgraph Isomorphism*:

Definition 4 (*Noisy Subgraph Isomorphism*)**.** *Given graphs* T *and* W*, we say that* F *is a Noisy Subgraph Isomorphism (NSI) if* F *minimizes the cost* $g(T, W, F)$*.*

We give an example of a NSI in Figure 1. We use the A* search algorithm to find all minimizing NSIs and other subgraph matches within a certain noise threshold. The A* search algorithm is widely used to find the shortest path in a graph. To apply this algorithm in our problem, we need to construct a search tree, where the root node represents the empty match, inner nodes represent partial matches and leaf

nodes represent complete matches. We define partial matches formally as follows:

Definition 5 (*Partial Match*)**.** *Given graphs T and W, a partial match P is defined by $P = (M(P), F(P))$, where $M(P)$ is a subset of $V(T)$ and $F(P)$ is a map from $M(P)$ to $V(W)$. We also call $M(P)$ matched nodes of this partial match. An empty match is a partial match with $M(P) = \varnothing$ and a complete match is a partial match with $M(P) = V(T)$.*

During the search process, we need to go from a node to its successors in a search tree. It means that we need to extend our partial match. The way of extending is described in the following definition:

Definition 6 (*Extending a Partial Match*)**.** *Given graphs T and W, and a partial match $P = (M(P), F(P))$. For an unmatched node $u \in V(T)$ and an unmatched world node $v \in V(W)$, a partial match P extended by $u \to v$ is defined by*

$$P + \{u \to v\} = (M(P) \cup \{u\}, F')$$

where

$$F'(u_0) = \begin{cases} v, & u_0 = u \\ F(u_0), & u_0 \in M(P). \end{cases}$$

V. Algorithms

In the A* search algorithm, we need a heuristic function defined on each node in the search tree. The heuristic estimates the cost of the best path from the root node through the current node to a leaf node. In our problem setting, given a partial match, we need to estimate the cost of the best complete match extended from the current partial match. Here, estimating the cost is equivalent to estimating the number of missing edges. To do this, we borrow the ideas of the Statistics Filter and the Topology Filter from [16] and propose our two estimations, the *Statistics Estimation* and the *Topology Estimation*. Again, we use T to represent the template graph and W to represent the world graph.

A. Statistics Estimation

The Statistics Estimation uses in-degree and out-degree to estimate the number of missing edges. Given $u \in V(T)$, the in-degree of u is given by $d^c_{\text{in}}(u) = \sum_{(u',u)\in E(T)} C^c_T(u', u)$, and out-degree of u is given by $d^c_{\text{out}}(u) = \sum_{(u,u')\in E(T)} C^c_T(u, u')$. If we match $u \in V(T)$ to $v \in V(W)$, we will have at least

$$\begin{aligned}\text{StatEst}(u, v) = \sum_c &\max\left\{d^c_{\text{in}}(u) - d^c_{\text{in}}(v), 0\right\} \\ &+ \max\left\{d^c_{\text{out}}(u) - d^c_{\text{out}}(v), 0\right\}\end{aligned}$$

missing edges.

B. Topology Estimation

Topology estimation estimates the number of missing edges for each pair of nodes in the template graph T. During the estimation, we use an auxiliary variable $\text{NodeMiss}(u)$ to record the number of missing edges already counted around node u.

First, we set $\text{NodeMiss}(u) = 0$ for every node u of the graph T initially. Given a partial match $P = (M, F)$, we iterate over all edges $(u, u') \in E(T)$. For each edge, we have the following three cases:

1) If both u and v are matched, the number of missing edges for this pair is

$$\begin{aligned}&\text{PairMiss}(u, u') \\ =&\text{miss}(F, (u, u')) + \text{miss}(F, (u', u)).\end{aligned}$$

2) If one of u and u' is matched, without loss of generality, we assume $u \in M(P), u' \notin M(P)$. Since u' is unmatched, we need to try all unmatched world nodes v' and then take the minimum over all unmatched world nodes to get the minimum number of missing edges. If we match u' to $v' \in V(W)$, then from statistics estimation, there will be $\text{StatEst}(u', v')$ missing edges around node u'. However, node u' may connect to other matched node, so some missing edges around node u' are already counted when we iterate over previous pairs. Recall that $\text{NodeMiss}(u')$ represents the number of missing edges already counted around node u'. So there are $\text{StatEst}(u', v') - \text{NodeMiss}(u')$ more missing edges. Also we can estimate the number of missing edges on the pair (u, u') as in case 1. Then

$$\begin{aligned}&\text{PairMiss}(u, u') = \\ &\min_{v'\in V(W)} \{\max\{\text{PairMiss}_0(u, u', v'), \\ &\qquad\qquad \text{StatEst}(u', v') - \text{NodeMiss}(u')\}\},\end{aligned}$$

where

$$\begin{aligned}&\text{PairMiss}_0(u, u', v') \\ =&\max\{C^c_T(u, u') - C^c_W(F(u), v'), 0\} \\ +&\max\{C^c_T(u', u) - C^c_W(v', F(u)), 0\}.\end{aligned}$$

After we get $\text{PairMiss}(u, u')$, we need to update $\text{NodeMiss}(u')$ by

$$\text{NodeMiss}(u') \leftarrow \text{NodeMiss}(u') + \text{PairMiss}(u, u').$$

3) If both u and v are unmatched, we simply set

$$\text{PairMiss}(u, u') = 0.$$

Now we can estimate the number of missing edges for a partial match P in the following way:

$$\text{TopoEst}(P) = \sum_{(u,u')\in E(T)} \text{PairMiss}(u, u').$$

So $\text{TopoEst}(P)$ can be used as the heuristic or f value in the A* search algorithm.

After we have the Statistics Estimation and the Topology Estimation, we use the A* search algorithm to find noisy subgraph isomorphisms. A partial match P is regarded as a state in the A* search algorithm. The f value of the state is evaluated by $\text{TopoEst}(P)$. The detailed procedure is described in Algorithm 1. We start with the empty match

$(\varnothing, \varnothing)$ where no nodes are matched. In each iteration of the while loop, we pick out a partial match P_0 with the lowest f value from openList. If all template nodes are matched in P_0, we find one subgraph match with low cost. Otherwise we pick an unmatched node which minimizes the increment of $\mathrm{TopoEst}(P)$ and try to match it to any one of the unmatched world nodes. We generate lots of new partial matches in this procedure. Then we calculate the f values of these partial matches and add them to openList.

One problem here is that $\mathrm{TopoEst}(P)$ may overestimate the cost. It means that for a subgraph match F extended from partial match P, $\mathrm{TopoEst}(P)$ may be larger than the cost of F. So the heuristic function is not consistent and we can not guarantee that the first subgraph match we find is the optimal one. In experiments below, in which we generate noisy data from existing datasets that contain exact matches, we verify that our algorithm always finds the noisy subgraph isomorphism, suggesting that $\mathrm{TopoEst}(P)$ does not overestimate the cost too much.

Algorithm 1 A* Search Algorithm

```
Put the empty match on the openList
while openList is not empty do
    currentState = state with the lowest f value in openList
    Remove currentState from openList
    if all template nodes in currentState are matched then
        add currentState to solutionList
    Pick an unmatched template nodes u in currentState
    for each candidate v of u do
        newState.partialMatch = currentState.partialMatch
        + {u → v}
        newState.f = TopoEst(newState.partialMatch)
        add newState to openList
return solutionList
```

One drawback of the A* search algorithm is that it takes too much memory when the search space is too large. To solve this problem, we use the iterative deepening A* search algorithm [12], which takes much less memory and guarantees an optimal solution if the heuristic function is consistent.

To further speed up our code, we borrow the idea of *Structural Equivalence* from [18]. Structural Equivalence refers to the ability to exchange two vertices in a graph without changing the structure of the graph, which occurs when two vertices have the same exact connections to the same set of neighbors. By integrating Structural Equivalence into the A* search algorithm, we reduce the redundant computations for highly symmetric templates. Instead of considering all permutations of equivalent template nodes, we only consider one of them.

Each subgraph isomorphism found using equivalence class can be expanded, by permutation, without changing the cost. In other words, we can compress the template graph and thus reduce the search space, thus accelerating the solution search in highly symmetric graphs. Consequently, when we perform the A* search algorithm, we keep only the representative of an equivalence class instead of all equivalent partial matches.

TABLE I: Overview of PNNL V6 B0

Dataset	Instance	Template		World	
		Nodes	Edges	Nodes	Edges
PNNL Version 6	B0-S0	74	1620	22996	12381816

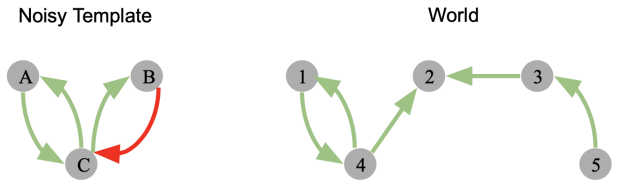

Fig. 2: Template noise toy model where the added edge is edge (B, C), shown in red.

VI. Noisy Models and Experiments

We have applied our algorithm to the dataset developed by Pacific Northwest National Laboratory (PNNL) [5] as part of the DARPA-MAA program [19]. This dataset consists of a 22K-node world graph with one 74-node template, which simulates a large number of transactions that take place between different agents. Table I summarizes the size of instance B0-S0 on which we perform our experiments. In this instance, we are provided with one world graph with an embedded signal that is isomorphic to the template. Previously, we sought to locate the hidden signal within the world graph. We now modify the given graphs with different noise models such that we are no longer guaranteed to find an exact match in the world graph. Now we seek for the NSIs. We define two noise models in the following subsections.

A. Template Noise Model

Definition 7 (*Template Noise Model*)**.** *Given a template graph T and a world graph W, where there exists exact match(es) between T and W, we say T_n is a template noise model of T if there exists no exact match between T_n and W, and for each pair of node (u, v), $C^c_T(u, v) \leq C^c_{T_n}(u, v)$ for each channel c. Here n is the number of missing edges between T_n and T, that is $n = \sum_c \sum_{(u,v)\in E(T)} C^c_{T_n}(u, v) - C^c_T(u, v)$.*

As an example, consider applying a template noise model on the problem in Figure 2 where we add an edge from node B to node C. In this case, there exists no exact match between the template noise model and the world graph any more. And the number of missing edges n is 1.

In our experiment, we create a template noise model T_n by adding edges to the existing template T. In this case, n is the number of edge(s) being added to T. Note that the edge(s) added ensure(s) that there are no exact matches between the template noise model T_n and the world graph W. We then perform our algorithm on T_n and W to optimize the Noisy Subgraph Isomorphism (NSI), which is the subgraph match with the lowest cost.

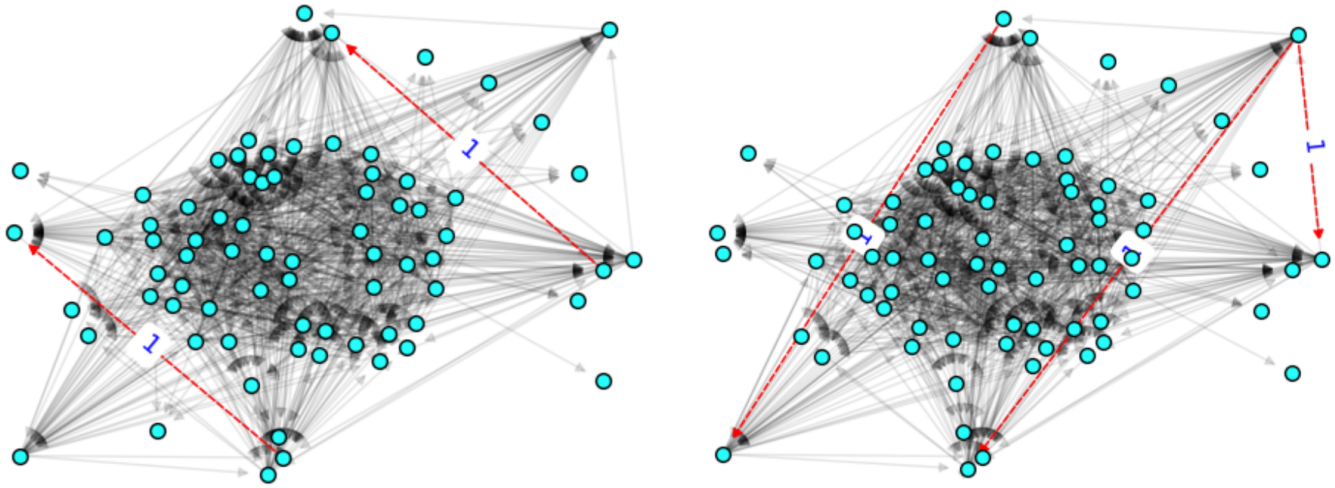

Fig. 3: Two NSIs we find on PNNL V6 B0 with the template noise model. Left panel: We add 2 edges to the original template graph and present a NSI with two missing edges. Right panel: We add 3 edges to the original template graph and present a NSI with three missing edges. The missing edges are marked as red dashed lines with the numbers of missing edges.

TABLE II: Time of finding a NSI on PNNL V6 B0 dataset with the template noise model

n	Cost of NSI	Run time (s)
1	1	528
2	2	756
3	3	1434
4	4	3083
5	5	2158
6	6	9717
7	7	7367
8	8	11986
9	9	15561

Table II summarizes the information of the NSI returned by our algorithm in the cases where we add different numbers of edges n to the template T. We show the cost of the NSI and the time we take to find this NSI in the second and third column, respectively. We also show the result of adding two and three edges to template graph in Figure 3. In those figures we use 1 to represent a missing edge.

Next we try to find more subgraph matches with higher cost. We let our program run for up to one hour trying to find all subgraph matches within the threshold. Note that our algorithm uses Structural Equivalence of the template graph for efficiency, so our program counts the number of subgraph match equivalence classes. Each subgraph match equivalence class can be extended to a set of equivalent subgraph matches. Table III summarizes the number of subgraph matches and subgraph match equivalence classes we found. Although we find many subgraph matches with some specific costs, there exist overlaps between the subgraph matches being recovered.

B. World Graph Noise Model

Definition 8 (*World Graph Noise Model*)**.** *Given a template graph T and a world graph W, where there exists exact match(es) between T and W, we say W_n is a world graph noise model of W if for each pair of node (u, v), $C^c_{W_n}(u, v) \leq C^c_W(u, v)$ for each channel c. Here n is the number of missing edges between W and W_n, that is $n = \sum_c \sum_{(u,v)\in E(T)} C^c_W(u, v) - C^c_{W_n}(u, v)$. Further, we denote p to be the noise percentage of the world graph noise model where $p\% = n/|E(W)| \times 100\%$, here $|E(W)|$ is total number of edges in world graph W.*

TABLE III: Number of SMs (Subgraph Matched) and SM equivalence classed on PNNL V6 B0 dataset with the template noise model

n	Cost	Run Time (s)	SM equivalence class count	SM Count
1	1	528	1	1152
	2	1529	22922	26406144
	5	3600	68582	79006464
2	2	756	1	1152
	3	2061	22922	26406144
	5	3600	22926	26410752
3	3	1478	4	1152
	4	3600	91688	26406144
	6	3600	22926	6602688
4	4	3600	1	288
	7	3600	2	576
	8	3600	38519	11093472

As an example, consider applying a world graph noise model on the problem in Figure 5 where we remove an edge from the world graph from node 4 to node 1. In this case, there exists no exact match between the template and the world graph noise model any more. And the edge removed is edge $(4, 1)$, thus $n = 1$. Notice that the noise percentage of this world graph noise model is $1/5 = 20\%$ because we remove one edge from the world graph which originally had five edges.

In our experiments, we create a world graph noise model

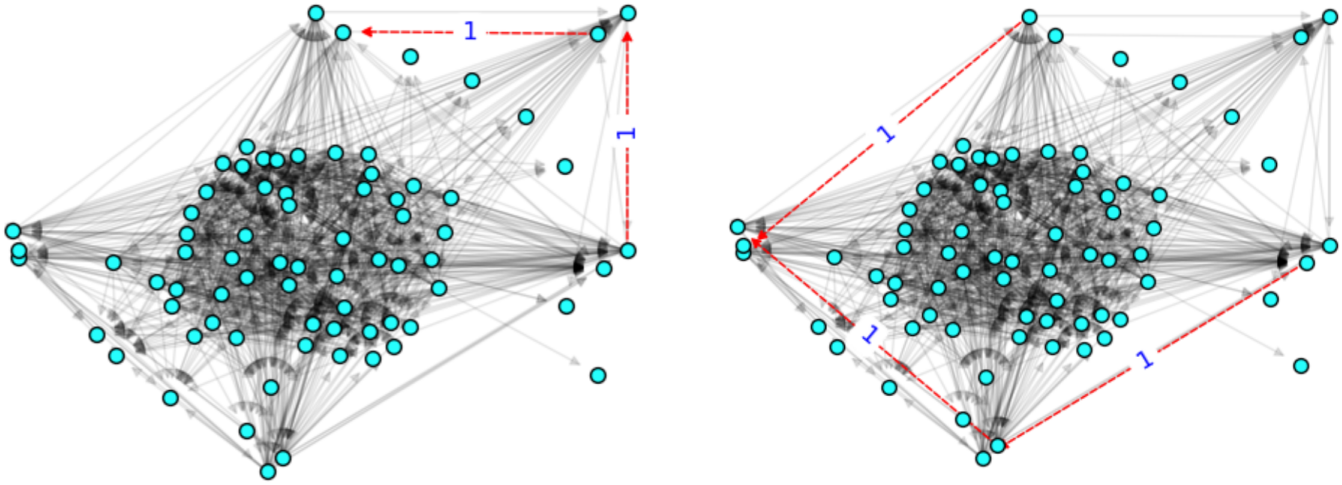

Fig. 4: Two NSIs we find on PNNL V6 B0 with the world graph noise model. Left panel: We remove 0.1% edges from the original world graph and present a NSI with two missing edges. Right panel: We remove 0.2% edges from the original world graph and present a NSI with three missing edges. The missing edges are marked as red dashed lines with the numbers of missing edges.

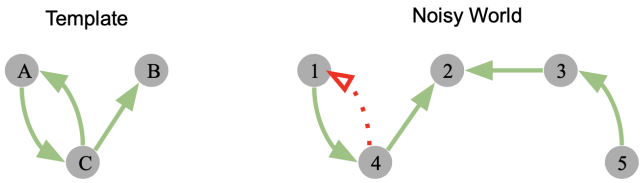

Fig. 5: World Graph noise toy model with 20% noise; the removed edge is (4,1)

W_n by removing edges from the existing world graph W. In this case, $n = p\% \times |E(W)|$ is the number of edges being removed from W. To get W_n, we randomly remove $p\%$ of existing edges within each channel c of the world graph W. Then n^c is the number of edges to be removed for the corresponding channel c. We then perform the algorithm on T and W_n to find the NSI. We run our experiments for $p = 0.1, 0.2$, and 0.4. The number of removed edges can be calculated with the formula in Definition 8, where $n = 12,319$ for $p = 0.1$ and $n = 49,275$ for $p = 0.4$. Table IV summarizes the information of the NSI when we remove $p\%$ of edges from the world graph W. We also show the result of removing 0.1% and 0.2% edges from world graph in Figure 4. We use 1 to represent a missing edge. The red dashed line represents a missing edge, and we find one signal with two missing edges for the 0.1% world graph noise model and one signal with three missing edges for the 0.2% world graph noise model. Again, we let our program run for up to one hour, trying to find all subgraph matches with a certain cost. Table V summarizes the number of subgraph matches and subgraph match equivalence classes we find. Here we only change the world graph, so the Structural Equivalence of the template

TABLE IV: Time of finding a NSI on PNNL V6 B0 dataset with the world graph noise model

p	Cost of NSI	Run-time
0.1	2	275
0.2	3	2199
0.4	5	12241

TABLE V: Number of SMs (Subgraph Matches) and SM equivalence classes on PNNL V6 B0 dataset with the world graph noise model

p	Cost	Run Time (s)	SM equivalence class count	SM Count
0.1	2	847	1	1152
	3	1296	22922	26406144
	5	3600	22922	26406144
0.2	3	1059	1	1152
	4	2957	22922	26406144
	6	3600	22923	26407296

graph does not change and each equivalence class can be extended to 1152 subgraph matches. When $p = 0.1$, we find 22922 subgraph matches equivalence classes with cost of 3 and 22922 subgraph match equivalence classes with cost of 5. If we let the code run for more than one hour, we could find more subgraph match equivalence classes with cost of 5.

VII. Conclusion

In this paper, we find noisy subgraph matches with missing edges on large multiplex networks. We propose an algorithm that integrates the statistics filter, topology filter from [16], and the A* search algorithm [9] to find noisy subgraph matches with minimum costs. We further integrate Structural Equivalence [18] to speed up the search by compressing the highly

symmetry graphs before performing noisy subgraph matching so that each NSI can be further expanded by permutation.

We build two noisy models — a template noise model and a world graph noise model. We apply our method on PNNL V6 B0. With the template noise model, we find subgraph matches for $n \leq 9$, and the first NSI (with a minimum cost) can be found within an hour for $n \leq 5$. With the world graph noise model, we find subgraph matches if we randomly remove fewer than 0.4% of the world edges; the first NSI can be found in 0.6 hours for $p\% \leq 0.2\%$.

In the future, we could take other factors into consideration when we design the cost of subgraph matches. For example, additional edges and missing edges can both be considered as cost. We could modify our algorithm and try to find the match with the smallest number of edge differences.

Further, we could use parallel computing to further speed up our code. We have already used Structural Equivalence, but it still takes over an hour to find a NSI when we add 0.4% noise to the world graph or add six edges to the template graph. We can parallelize the Statistics Estimation and the Topology Estimation on each node and edge. Since we identify that these computations as bottlenecks, we expect significant speedup after parallelization.

ACKNOWLEDGMENT

This material is based on research sponsored by the Air Force Research Laboratory and DARPA under agreement number FA8750-18-2-0066. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon.

The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Air Force Research Laboratory and DARPA or the U.S. Government.Data provided by the MAA groups at Pacific Northwest National Laboratory.

We thank Thien Nguyen, Dominic Yang, Yurun Ge, Jacob Moorman, Thomas Tu, Qinyi Chen, Mason Porter for useful conversations.

REFERENCES

[1] Tero Aittokallio and Benno Schwikowski. Graph-based methods for analysing networks in cell biology. *Briefings in Bioinformatics*, 7(3):243–255, 09 2006.

[2] Vincenzo Bonnici and Rosalba Giugno. On the variable ordering in subgraph isomorphism algorithms. *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)*, 14(1):193–203, 2017.

[3] Vincenzo Carletti, Pasquale Foggia, Alessia Saggese, and Mario Vento. Introducing VF3: A new algorithm for subgraph isomorphism. In *International Workshop on Graph-Based Representations in Pattern Recognition*, pages 128–139. Springer, 2017.

[4] Luigi P Cordella, Pasquale Foggia, Carlo Sansone, and Mario Vento. A (sub) graph isomorphism algorithm for matching large graphs. *IEEE transactions on pattern analysis and machine intelligence*, 26(10):1367–1372, 2004.

[5] J. A. Cottam, S. Purohit, P. Mackey, and G. Chin. Multi-channel large network simulation including adversarial activity. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 3947–3950, Dec 2018.

[6] Nicholas Dahm, Horst Bunke, Terry Caelli, and Yongsheng Gao. Efficient subgraph matching using topological node feature constraints. *Pattern Recognition*, 48(2):317–330, 2015.

[7] Fred DePiero and David Krout. An algorithm using length-r paths to approximate subgraph isomorphism. *Pattern recognition letters*, 24(1-3):33–46, 2003.

[8] Andreas Fischer, Ching Y Suen, Volkmar Frinken, Kaspar Riesen, and Horst Bunke. Approximation of graph edit distance based on Hausdorff matching. *Pattern Recognition*, 48(2):331–343, 2015.

[9] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[10] Huahai He and Ambuj K Singh. Graphs-at-a-time: query language and access methods for graph databases. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 405–418. ACM, 2008.

[11] Yi Jia, Jintao Zhang, and Jun Huan. An efficient graph-mining method for complicated and noisy data with real-world applications. *Knowledge and Information Systems*, 28(2):423–447, 2011.

[12] Richard E Korf. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial intelligence*, 27(1):97–109, 1985.

[13] Alper Küçükural, Andras Szilagyi, O Ugur Sezerman, and Yang Zhang. Protein homology analysis for function prediction with parallel subgraph isomorphism. In *Bioinformatics: concepts, methodologies, tools, and applications*, pages 386–399. IGI Global, 2013.

[14] Vincent Lacroix, Cristina G Fernandes, and Marie-France Sagot. Motif search in graphs: application to metabolic networks. *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)*, 3(4):360–368, 2006.

[15] Lorenzo Livi and Antonello Rizzi. The graph matching problem. *Pattern Analysis and Applications*, 16(3):253–283, 2013.

[16] Jacob D Moorman, Qinyi Chen, Thomas K Tu, Zachary M Boyd, and Andrea L Bertozzi. Filtering methods for subgraph matching on multiplex networks. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 3980–3985. IEEE, 2018.

[17] Michel Neuhaus, Kaspar Riesen, and Horst Bunke. Fast suboptimal algorithms for the computation of graph edit distance. In *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR)*, pages 163–172. Springer, 2006.

[18] Thien Nguyen, Dominic Yang, Yurun Ge, Hao Li, and Andrea L. Bertozzi. Applications of structural equivalence to subgraphisomorphism on multichannel multigraphs. 2019.

[19] Boyan Onyshkevych. Modeling adversarial activity (maa).

[20] Christine Solnon. Alldifferent-based filtering for subgraph isomorphism. *Artificial Intelligence*, 174(12-13):850–864, 2010.

[21] Daniel Sussman, Youngser Park, Carey E Priebe, and Vince Lyzinski. Matched filters for noisy induced subgraph detection. *IEEE transactions on pattern analysis and machine intelligence*, 2019.

[22] Julian R Ullmann. An algorithm for subgraph isomorphism. *Journal of the ACM (JACM)*, 23(1):31–42, 1976.

[23] Xiaoli Wang, Xiaofeng Ding, Anthony KH Tung, Shanshan Ying, and Hai Jin. An efficient graph indexing method. In *2012 IEEE 28th International Conference on Data Engineering*, pages 210–221. IEEE, 2012.

[24] David W Williams, Jun Huan, and Wei Wang. Graph database indexing using structured graph decomposition. In *2007 IEEE 23rd International Conference on Data Engineering*, pages 976–985. IEEE, 2007.

[25] Xifeng Yan, Philip S Yu, and Jiawei Han. Substructure similarity search in graph databases. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 766–777. ACM, 2005.

[26] Zhiping Zeng, Anthony KH Tung, Jianyong Wang, Jianhua Feng, and Lizhu Zhou. Comparing stars: On approximating graph edit distance. *Proceedings of the VLDB Endowment*, 2(1):25–36, 2009.