This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSE.2021.3056329, IEEE Transactions on Network Science and Engineering

Subgraph Matching on Multiplex Networks

Jacob D. Moorman, Thomas K. Tu, Qinyi Chen, Xie He, and Andrea L. Bertozzi, *Member, IEEE*

Abstract—An active area of research in computational science is the design of algorithms for solving the subgraph matching problem to find copies of a given template graph in a larger world graph. Prior works have largely addressed single-channel networks using a variety of approaches. We present a suite of filtering methods for subgraph isomorphisms for multiplex networks (with different types of edges between nodes and more than one edge within each channel type). We aim to understand the entire solution space rather than focusing on finding one isomorphism. Results are shown on several classes of datasets: (a) Sudoku puzzles mapped to the subgraph isomorphism problem, (b) Erdős-Rényi multigraphs, (c) real-world datasets from Twitter and transportation networks, (d) synthetic data created for the DARPA MAA program.

Index Terms—subgraph isomorphism, graph isomorphism, graph matching, subgraph matching, multiplex network

1 Introduction

Multiplex networks (labeled directed multigraphs, Definition 1.1) [32] are increasingly useful data structures for representing entities and their interactions in disciplines such as bioinformatics [73], social networks [65], ecological networks [49], and neural networks [5]. Subgraph matching is the process of determining whether a given *template* network occurs as a subgraph of a *world* network, and if so, exactly where it occurs and how many times [13].

Subgraph matching is commonly used in bioinformatics [71], social network analysis [19], [68], and other applications [13]. It is also an important subroutine in frequent subgraph mining [28], [67] and graph database search [74]. Despite the abundance of multiplex network data in these applications, there are relatively few subgraph matching algorithms that expressly support multiplex networks [29], [43] compared to the number of algorithms that support single-channel networks [6], [9], [10], [14], [24], [57], [61]. In this paper, we introduce a new algorithm for subgraph matching on multiplex networks and discuss some simplifications of the subgraph matching problem.

We refer to any subgraph of the world that matches the template as a *signal*. For sufficiently simple templates, there are efficient algorithms for counting and listing out all corresponding signals [1], [53], [54]. However, in general there can be a nonsensically large number of signals, in which case listing or even counting them can be impossible. In such situations, it is appealing to have methods that can characterize the space of all signals in some way. For example, one might be interested in the set of world nodes which participate in at least one signal. Alternatively, one may seek the set of world nodes which correspond to a particular template node in at least one signal. We find that these problems can be feasible, even when it is impossible to list or count the signals.

Definition 1.1 (Multiplex Network)**.** *A multiplex network $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{L}, \mathcal{C})$ is a set of nodes (frequently called vertices), directed edges between the nodes, labels on the nodes, and channels on the edges. The number of nodes is denoted n. Each node $\mathbf{v} \in \mathcal{V}$ has a label $\mathcal{L}(\mathbf{v})$ belonging to some arbitrary set of labels. There can be any number of edges between each pair of nodes $(\mathbf{u}, \mathbf{v})$ in either direction. Each edge belongs to one of the channels $\mathcal{C}$. Edges between the same pair of nodes in the same channel with the same direction are indistinguishable. The function $\mathcal{E} : \mathcal{V} \times \mathcal{V} \to \mathbb{N}^{|\mathcal{C}|}$ describes the number of edges in each channel between each pair of nodes. In particular, $\mathcal{E}(\mathbf{u}, \mathbf{v})$ can be represented as a $|\mathcal{C}|$-dimensional vector the k^{th} element of which is the number of edges from node $\mathbf{u}$ to node $\mathbf{v}$ in the k^{th} channel. $|\mathcal{E}|_0$ denotes the number of distinguishable edges in $\mathcal{G}$.*

In the remainder of this section, we will define several problems related to subgraph matching and discuss existing approaches to solve these problems. In Section 1.3, we explain our contributions to solving these problems. We expand on the details of our approach in Sections 2, 3.1 and 3.2. In Section 4, we perform several experiments to show that the methods we discuss are successful in solving the problems of interest. We make some concluding remarks and suggest some future avenues for research in Sections 5 and 6 respectively.

1.1 Problem Statements

Given two multiplex networks, a template $\mathcal{G}_t = (\mathcal{V}_t, \mathcal{E}_t, \mathcal{L}_t, \mathcal{C})$ and a world $\mathcal{G}_w = (\mathcal{V}_w, \mathcal{E}_w, \mathcal{L}_w, \mathcal{C})$, we explore the space of all subgraphs of the world which *match* the template. There are several closely related problems that achieve this aim to different extents with different computational costs. Each of these problems relies on the same concept of *subgraph isomorphism (SI)* which characterizes what it means for a subgraph of the world to *match* the template.

- *Jacob D. Moorman, Thomas K. Tu, and Andrea L. Bertozzi are with the Department of Mathematics, University of California, Los Angeles, Los Angeles, CA, 90095. E-mail: jdmoorman@math.ucla.edu, thomastu@math.ucla.edu, bertozzi@math.ucla.edu*
- *Qinyi Chen is with the Operations Research Center, Massachusetts Institute of Technology, Cambridge, MA, 02139. E-mail: qinyic@mit.edu*
- *Xie He is with the Department of Mathematics, University of North Carolina, Chapel Hill, Chapel Hill, NC, 27514. E-mail: hexie@unc.edu*

Definition 1.2 (SI: Subgraph Isomorphism)**.** *An injective function $f : \mathcal{V}_t \to \mathcal{V}_w$ is called a* subgraph isomorphism (SI) *from $\mathcal{G}_t = (\mathcal{V}_t, \mathcal{E}_t, \mathcal{L}_t, \mathcal{C})$ to $\mathcal{G}_w = (\mathcal{V}_w, \mathcal{E}_w, \mathcal{L}_w, \mathcal{C})$ if*

$$\begin{aligned} \mathcal{L}_t(\mathbf{v}) &= \mathcal{L}_w(f(\mathbf{v})) && \forall \mathbf{v} \in \mathcal{V}_t \\ \mathcal{E}_t(\mathbf{u}, \mathbf{v}) &\leq \mathcal{E}_w\left(f(\mathbf{u}), f(\mathbf{v})\right) && \forall \mathbf{u}, \mathbf{v} \in \mathcal{V}_t \times \mathcal{V}_t. \end{aligned}$$

The set of all SIs from $\mathcal{G}_t$ to $\mathcal{G}_w$ is denoted $\mathcal{F}(\mathcal{G}_t, \mathcal{G}_w)$.

In short, an injective function f mapping template nodes to world nodes is a SI if each node $\mathbf{v}$ in the template has the same label as the corresponding node $f(\mathbf{v})$ in the world, and for each pair of nodes $(\mathbf{u}, \mathbf{v})$ in the template, the corresponding pair of nodes $(f(\mathbf{u}), f(\mathbf{v}))$ in the world have *at least as many* edges in each channel and direction as $\mathbf{u}$ and $\mathbf{v}$ have between them. This is sometimes referred to as a subgraph monomorphism in the literature. We use the term *induced SI* when there are instead exactly the same number of edges in each channel and direction. The problems introduced in the remainder of Section 1.1 are summarized in terms of SIs in Table 1.

Problem	Description
SIP	Check if there are *any* SIs.
SNSP	Find all the world nodes involved in SIs.
MCSP	Find all pairs $(\mathbf{u}, \mathbf{v})$ where $\mathbf{u} = f(\mathbf{v})$ for some SI f.
SICP	Count the number of SIs.
SMP	Find all the SIs.

TABLE 1: A summary of the various problems introduced in Section 1.1, in increasing order of computation cost.

Nodes in the image of a SI are called *signal nodes*, and the induced subgraph of the image of a SI is called a *signal*. For example, Figure 1 highlights several signals in an example problem. Note that there may be more edges between signal nodes than there are between the corresponding template nodes. As an example, in Figure 1 there are more edges between $\mathbf{5}$ and $\mathbf{7}$ than there are between $\mathbf{B}$ and $\mathbf{C}$.

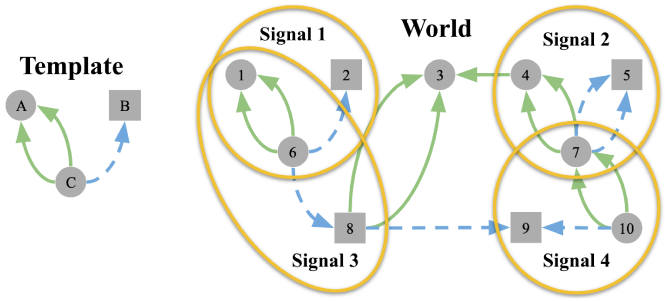


Fig. 1: In the above networks, the shapes of the nodes corresponds to their labels (circle or square) and the patterns of the edges correspond to their channel (solid green or dashed blue). Given the template and world networks above, there are four signals consisting of the subgraphs of the world induced by $\{\mathbf{1}, \mathbf{2}, \mathbf{6}\}$, $\{\mathbf{1}, \mathbf{6}, \mathbf{8}\}$, $\{\mathbf{4}, \mathbf{5}, \mathbf{7}\}$, and $\{\mathbf{7}, \mathbf{9}, \mathbf{10}\}$.

A typical definition of SI would include a map from template edges to world edges. However, we omit this consideration since we consider edges to be indistinguishable. If edges were considered to be distinct, there would be two SIs in Figure 1, both corresponding to signal 2, one for each way to choose the dashed blue edge between template nodes $\mathbf{B}$ and $\mathbf{C}$ from among those between world nodes $\mathbf{5}$ and $\mathbf{7}$.

Definition 1.3 (SIP: Subgraph Isomorphism Problem)**.** *Given a template $\mathcal{G}_t = (\mathcal{V}_t, \mathcal{E}_t, \mathcal{L}_t, \mathcal{C})$ and a world $\mathcal{G}_w = (\mathcal{V}_w, \mathcal{E}_w, \mathcal{L}_w, \mathcal{C})$, check if there is at least one SI from the template $\mathcal{G}_t$ to the world $\mathcal{G}_w$. That is,*

$$\textit{check if} \quad \mathcal{F}(\mathcal{G}_t, \mathcal{G}_w) = \emptyset. \tag{SIP}$$

Typically the SIP is solved by exhaustively searching for any SI $f \in \mathcal{F}(\mathcal{G}_t, \mathcal{G}_w)$. If none can be found, then $\mathcal{F}(\mathcal{G}_t, \mathcal{G}_w) = \emptyset$. Though the SIP is NP-complete [22], it can be solved in practice even for some very large networks [53], [59]. The challenge of finding all SIs, rather than simply checking whether there are any, is called the *subgraph matching problem (SMP)*.

Definition 1.4 (SMP: Subgraph Matching Problem)**.** *Given a template $\mathcal{G}_t = (\mathcal{V}_t, \mathcal{E}_t, \mathcal{L}_t, \mathcal{C})$ and a world $\mathcal{G}_w = (\mathcal{V}_w, \mathcal{E}_w, \mathcal{L}_w, \mathcal{C})$, find all SIs from the template $\mathcal{G}_t$ to the world $\mathcal{G}_w$. That is,*

$$\textit{find all} \quad f \in \mathcal{F}(\mathcal{G}_t, \mathcal{G}_w). \tag{SMP}$$

The solution to the SMP for Figure 1 is the set of SIs $\mathcal{F}(\mathcal{G}_t, \mathcal{G}_w) = \{f_1, f_2, f_3, f_4\}$, where the f_i are described in Table 2. Notice that SIs can differ by as little as one node, for example $f_1(\mathbf{v}) = f_3(\mathbf{v})$ for every template node $\mathbf{v}$ except when $\mathbf{v} = \mathbf{B}$. Additionally, some SIs are completely disjoint. For example, $f_2(\mathcal{V}_t) \bigcap f_3(\mathcal{V}_t) = \emptyset$.

Template node $\mathbf{v}$	$f_1(\mathbf{v})$	$f_2(\mathbf{v})$	$f_3(\mathbf{v})$	$f_4(\mathbf{v})$	$\bigcup_i \{f_i(\mathbf{v})\}$
A	**1**	**4**	**1**	**7**	$\{\mathbf{1}, \mathbf{4}, \mathbf{7}\}$
B	**2**	**5**	**8**	**9**	$\{\mathbf{2}, \mathbf{5}, \mathbf{8}, \mathbf{9}\}$
C	**6**	**7**	**6**	**10**	$\{\mathbf{6}, \mathbf{7}, \mathbf{10}\}$

TABLE 2: Solutions to SMP and MCSP corresponding to the template and world shown in Figure 1.

The time taken to solve the SMP scales with the number of SIs $|\mathcal{F}(\mathcal{G}_t, \mathcal{G}_w)|$, since any algorithm must list out all of the SIs. When there are too many SIs, it is impossible to solve the SMP. The problems described in the remainder of this section can be used to more cheaply explore the space of SIs.

In certain contexts, one may wish only to count the number of SIs from the template to the world. For example, when the count will be used as a summary statistic of the world network as in triangle counting [54] or motif counting [2]. This is called the *SI counting problem (SICP)*.

Definition 1.5 (SICP: Subgraph Isomorphism Counting Problem)**.** *Given a template $\mathcal{G}_t = (\mathcal{V}_t, \mathcal{E}_t, \mathcal{L}_t, \mathcal{C})$ and a world $\mathcal{G}_w = (\mathcal{V}_w, \mathcal{E}_w, \mathcal{L}_w, \mathcal{C})$, determine the number of SIs from the template $\mathcal{G}_t$ to the world $\mathcal{G}_w$. That is,*

$$\textit{find} \quad \left|\mathcal{F}(\mathcal{G}_t, \mathcal{G}_w)\right|. \tag{SICP}$$

Another concise summary of the set of SIs is the set of all signal nodes, i.e. those nodes that are in the image of at least one SI. Finding this set of nodes is the *signal node set problem (SNSP)*. Though the SNSP lacks the full descriptive power of the full set of SIs, it is much more compact and easier to calculate.

Definition 1.6 (SNSP: Signal Node Set Problem)**.** *Given a template* $\mathcal{G}_t = (\mathcal{V}_t, \mathcal{E}_t, \mathcal{L}_t, \mathcal{C})$ *and a world* $\mathcal{G}_w = (\mathcal{V}_w, \mathcal{E}_w, \mathcal{L}_w, \mathcal{C})$*, find all world nodes that belong to at least one signal. That is,*

$$\textit{find} \quad \bigcup_{f \in \mathcal{F}(\mathcal{G}_t, \mathcal{G}_w)} f(\mathcal{V}_\mathbf{t}). \qquad \text{(SNSP)}$$

A compromise between the compactness of the SNSP and the descriptive power of the SMP is to find for each template node its *minimal candidate set*, the smallest set containing all world nodes that correspond to that template node in any signal. This preserves the relation between template nodes and world nodes, but loses some information about compatibility between candidates for different template nodes.

Definition 1.7 (MCSP: Minimal Candidate Sets Problem)**.** *Given a template* $\mathcal{G}_t = (\mathcal{V}_t, \mathcal{E}_t, \mathcal{L}_t, \mathcal{C})$ *and a world* $\mathcal{G}_w = (\mathcal{V}_w, \mathcal{E}_w, \mathcal{L}_w, \mathcal{C})$*, for each template node* $\mathbf{v} \in \mathcal{V}_t$ *find all world nodes which correspond to* $\mathbf{v}$ *in at least one signal. That is,*

$$\textit{find} \quad \bigcup_{f \in \mathcal{F}(\mathcal{G}_t, \mathcal{G}_w)} \{f(\mathbf{v})\} \quad \textit{for each} \quad \mathbf{v} \in \mathcal{V}_t. \qquad \text{(MCSP)}$$

A naive algorithm for solving the MCSP is to solve the SIP with the added constraint $\mathbf{u} = f(\mathbf{v})$ for each $(\mathbf{u}, \mathbf{v}) \in \mathcal{V}_w \times \mathcal{V}_t$. Thus, solving the MCSP is at most $|\mathcal{V}_t|\,|\mathcal{V}_w|$ times as hard as the hardest among those SIPs.

1.2 Related Work

Most state of the art algorithms for subgraph matching follow one of three approaches [9], [10]: tree search, constraint propagation, and graph indexing. Existing algorithms are largely restricted to the single-channel case and thus are not applicable to multiplex networks. Also, existing algorithms focus mainly on addressing the SIP, SMP, and SICP, and do not address the MCSP or SNSP directly.

Tree search approaches keep track of a search state, and navigate the tree of possible search states, backtracking when they reach the end of a branch. Due to the enormity of this tree, to limit computational complexity as much as possible, these approaches refine the search space at each step of the search to avoid unnecessary branches. Examples of tree search approaches include Ullmann's algorithm [61], VF2 [14] and its variants (VF2Plus [11], VF3 [9], [10], VF2++ [30]), and for specific graphs, RI/RI-DS [7].

Constraint propagation approaches view subgraph isomorphism as a constraint satisfaction problem, where variables are assigned values while satisfying a given set of constraints. These approaches keep track of a compatibility matrix, which indicates which nodes in the world are possible matches for which template nodes. By repeatedly applying local constraints, this matrix is reduced until only a few possible candidate matchings remain. The matrix can then be explored to find all solutions. Examples of constraint propagation approaches include McGregor [42], nRF+ [36], ILF [69], LAD [57] (and its variants, IncompleteLAD and PathLAD [35]), McCreesh and Prosser (Glasgow) [40], and FocusSearch [62].

Graph indexing approaches seek to retrieve from a database of graphs all graphs that match a given subgraph query. To accelerate searching, they construct indexes for the database, so that future searches will be efficient. These indexes are often based on characteristic substructures of the template. A Cartesian product is then performed on the results of these indexed queries, identifying all possible matches. Next, there is a verification step to check which retrieved graphs fully match the pattern; this typically involves running another subgraph isomorphism algorithm, such as VF2. Examples of graph indexing approaches include GraphQL [25], SPath [72], GADDI [71], QuickSI [55], TurboISO [24], BoostISO [52], CFL-Match [6], and CNI-Match [45]. Though our problem does not involve graph databases and we thus do not construct indices, several of these approaches involve filtering techniques which can be used independently of the indices they construct [58].

1.2.1 Ullmann's Algorithm

Ullmann's algorithm [61] is a backtracking tree search with refinement. For every template node $\mathbf{v}$, they create a list of candidate nodes in the world which could correspond to $\mathbf{v}$. Initially, this is simply all nodes with degree greater than or equal to the degree of $\mathbf{v}$. At each step in the tree search, candidate node $\mathbf{u}$ is chosen as a match for template node $\mathbf{v}$. To reduce computation time, the remaining candidates are then refined as follows. For every pair of template nodes joined by an edge, their candidates should also be joined by an edge in the world. Any candidate for one of these connected nodes that does not have an edge to any candidate for the other connected node can be removed from consideration. Our topology filter, as detailed in Section 2.3, is a simple extension of this constraint to the multiplex network case.

1.2.2 VF2

VF2 [14] generalizes Ullmann's algorithm to directed graphs and extends refinement with additional semantic feasibility rules. They distinguish matched nodes (nodes with only one candidate) from other nodes, and ensure that candidates have more matched neighbors and unmatched neighbors than their corresponding template node; in the directed case, these neighbors must have edges with matching directions. They also enforce a matching order where neighbors of previously matched nodes are matched before other nodes.

1.2.3 Constraint Propagation Approaches

In [36], the authors classify existing constraints using two categorizations: binary vs. non-binary and forward checking (FC) vs. really forward look ahead (RF). Binary constraints map n variables to m values using an nxm matrix of binary-valued variables, while non-binary constraints use a vector of length n, whose entries are restricted to m values. The authors propose nRF+, a non-binary really forward lookahead algorithm, with specific look ahead for subgraph isomorphism. They enforce the constraint that if $\mathbf{v}_w$ is a candidate for template node $\mathbf{v}_t$, then the number of $\mathbf{v}_t$'s neighbors must be less than or equal to the number of candidates for those neighbors that are adjacent to $\mathbf{v}_w$.

In [57], many constraints previously used in different subgraph isomorphism algorithms are classified. The author proposes a new algorithm, LAD, using the constraint that for a match to exist between nodes m and n, there must exist a matching between their neighborhoods subject to

the alldifferent constraint introduced in [51], which can be identified using the Hopcroft-Karp algorithm [27].

1.2.4 Graph Indexing Approaches

In GraphQL [25], the authors describe a graph query language and provide an algorithm for resolving graph database queries. In particular, they iterate a constraint similar to that of [57], where there must exist a bipartite matching between the neighborhoods of a template node and its candidate. This approach is further expanded upon in SPath [72], where the k-distance neighborhood is also considered.

TurboISO proposes a method for handling permutable template nodes, as well as addressing the order of matching when identifying isomorphisms. They construct an NEC tree, whose vertices represent groups of permutable template nodes, and perform matching using this tree. At the end, they combine and permute the candidates for each of these template nodes. For the matching order, they divide the graph into candidate subregions, each of which contains a group of candidates that may take part in a signal. They then prioritize searching the smaller candidate subregions. We take a similar approach in Section 3.1, prioritizing nodes with the least number of candidates.

In CFL-Match [6], the template is first decomposed into three types of structures: Core, Forest, and Leaf. To do this, they first construct a spanning tree of the template. Then, they compute the minimal connected subgraph containing all nontree edges of the template. They then iteratively remove all nodes with degree 1, updating the degree counts after each removal. The remaining nodes form the core.

For each node in the core that is connected to a non-core node, a forest structure is created, consisting of that node and all non-core nodes it is connected to. Finally, the leaf structure consists of chains of removed non-core nodes.

After decomposing the template in this way, each structure is queried independently. In order to postpone the Cartesian product of the results as much as possible, the core is queried first, and the results are then used to restrict queries for each forest structure. Similarly, the leaf queries are restricted by the results of the forest queries.

1.2.5 Multiplex Approaches

Though most subgraph isomorphism algorithms in the literature focus on the single-channel networks, there are two algorithms that explicitly solve the multiplex SMP. SuMGrA [29] is a graph indexing and backtracking tree search approach and RI [7] can be extended to the multiplex case (MultiRI) [43]. These existing multiplex approaches are designed to solve the SMP. However, the SMP is infeasible when there are too many SIs as in many of the problem instances in Section 4. Additionally, the networks considered by SuMGrA and MultiRI differ slightly from Definition 1.1. Though they allow multiple edges between nodes, they do not allow multiple edges between a pair of nodes in the same channel. SuMGra also does not allow directed edges.

To extend any existing single-channel algorithms to multiplex networks, one possibility is to use a single-channel approach (e.g., VF2, LAD) to solve the SMP in each channel and take an intersection across all channels. However, this approach is infeasible when there are too many SIs in any channel, even if there are relatively few SIs overall. We will show in Section 4 that in some cases the number of SIs is too large to solve the multiplex SMP, let alone the single-channel SMP.

1.3 Contributions

We extend existing constraint satisfaction approaches to operate on multiplex networks. We demonstrate experimentally that these approaches allow us to list out or count all SIs for world graphs with thousands to hundreds of thousands of nodes and templates with tens to hundreds of nodes.

Due to the underconstrained nature of the templates in some datasets, there are often too many SIs to reasonably list, or sometimes even count. In such situations, we propose that a natural problem to solve in place of the SMP or SICP is the MCSP. On datasets where we can solve the MCSP, we observe that the output of our filters can be a good approximation to the solution of the MCSP.

We have published our code as an open-source Python package available on GitHub (https://github.com/jdmoorman/uclasm/tree/master) [44].

2 Filtering

Our algorithms for solving the problems discussed in Section 1.1 revolve around the use of *filters* to cheaply approximate the solution of the MCSP (Definition 1.7). For each template node $\mathbf{v}_t \in \mathcal{V}_t$, we keep track of its *candidates* $D(\mathbf{v}_t) \subseteq \mathcal{V}_w$. Initially, we treat every world node as a candidate for every template node. Each filter enforces a different set of constraints to eliminate candidates which cannot be part of any signal. The filters are applied repeatedly until no further candidates can be eliminated (Algorithm 1), applying the cheaper filters before the more expensive ones.

Algorithm 1 Filtering

1: **Input** template $\mathcal{G}_t$, world $\mathcal{G}_w$, candidate set $D(\mathbf{v}_t)$ for each $\mathbf{v}_t \in \mathcal{V}_t$, list of filters `filters`
2: `converged` $\leftarrow$ False
3: **while** `converged` *is* False
4: `converged` $\leftarrow$ True ▷ Stop unless progress is made
5: **for** `filter` $\in$ `filters`
6: $D \leftarrow$ `filter`$(\mathcal{G}_t, \mathcal{G}_w, D)$ ▷ Apply the filter
7: **if** $|D(\mathbf{v}_t)|$ decreased for some $\mathbf{v}_t \in \mathcal{V}_t$
8: `converged` $\leftarrow$ False ▷ Progress was made
9: Break ▷ Restart from the first filter
10: **Output** updated candidate set $D(\mathbf{v}_t)$ for each $\mathbf{v}_t \in \mathcal{V}_t$

In this paper, in general, we are not searching for induced subgraphs: we do not require equal number of edges to exist between nodes in the template and nodes in the world graph; instead, we only require the edges between template nodes to be less than or equal to the number of edges between their corresponding candidates in the world. However, modifying our filters to find induced subgraphs is simple: in Algorithm 3, change $\geq$ to $==$ on Lines 6 and 7, and in the Algorithm 5, change $\geq$ to $==$ each time it appears on Line 8.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSE.2021.3056329, IEEE Transactions on Network Science and Engineering

2.1 Node Label Filter

In order for a world node $\mathbf{v}_w$ to be a candidate for a template node $\mathbf{v}_t$, the label $\mathcal{L}_w(\mathbf{v}_w)$ must match the label $\mathcal{L}_t(\mathbf{v}_t)$. For example, consider the template and world shown in Figure 1 in which the node labels correspond to their shapes. By the label filter, the square world nodes **2**, **5**, **8**, and **9** can be eliminated as candidates for the circular template nodes **A** and **C**, while the circular world nodes **1**, **3**, **4**, **6**, **7**, and **10** can be eliminated as candidates for the square template node **B**. The label filter is run only once, immediately after receiving the template and world.

In some applications, template node labels cannot be specified exactly. For example, in geospatial applications, template node labels may represent broad regions, whereas world node labels may represent exact coordinates. In such applications, it does not make sense to require equality between template node labels and world node labels. Rather, one should require the world node labels to be somehow "compatible" with the template node labels. The notion of compatibility will depend on the application. In Section 4.3.1 we discuss an example application where a world node is compatible with a template node if the coordinates of the world node lie within the region of the template node.

2.2 Node-level Statistics Filter

The idea behind the node-level statistics filter (Algorithm 2) is intuitive: for a world node $\mathbf{v}_w$ to be a candidate for a template node $\mathbf{v}_t$, certain statistical properties of $\mathbf{v}_w$ should not be less than those of $\mathbf{v}_t$. The idea of the node-level statistics filter has been applied to simpler settings in the related literature [43], [57]. Any statistic that is non-decreasing as nodes and/or edges are added to a graph can be used as part of the filter. The statistics that have been applied in our filter include in/out-degree, number of in/out-neighbors, number of reciprocated edges, and number of self-edges. Each of these statistics can be used in each channel in the networks.

Algorithm 2 Node-level Statistics Filter

Input template $\mathcal{G}_t$, world $\mathcal{G}_w$, candidate set $D(\mathbf{v}_t)$ for each $\mathbf{v}_t \in \mathcal{V}_t$
for `statistic` $\in$ [in-degree, out-degree, ...]
  for template or world node $\mathbf{v} \in \mathcal{V}_t \cup \mathcal{V}_w$
    Compute `statistic`$(\mathbf{v})$
  for template node $\mathbf{v}_t \in \mathcal{V}_t$
    for candidate $\mathbf{v}_w \in D(\mathbf{v}_t)$
      if `statistic`$(\mathbf{v}_w) <$ `statistic`$(\mathbf{v}_t)$
        Remove $\mathbf{v}_w$ from $D(\mathbf{v}_t)$
Output updated candidate set $D(\mathbf{v}_t)$ for each $\mathbf{v}_t \in \mathcal{V}_t$

Other more complex statistics can be used in each channel, such as number of triangles, number of nodes within k steps, and number of paths of length ℓ. Statistics combining information from multiple channels can also be used, such as the number of in-neighbors in channel a which are also out-neighbors in channel b. We use only the simplest statistics to ensure that the cost of computing the statistics will scale only linearly in the number of distinguishable edges (edges whose source, destination, direction, and channel are distinct).

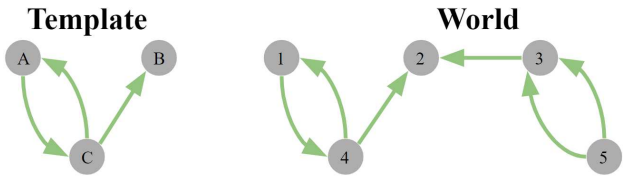


Fig. 2: In the network shown, there is only one valid signal for the template: **1** for **A**, **2** for **B**, and **4** for **C**.

	Template			World				
	A	**B**	**C**	**1**	**2**	**3**	**4**	**5**
in-degree	1	1	1	1	2	2	1	0
out-degree	1	0	2	1	0	1	2	2

TABLE 3: In/out-degree for nodes in the template and world shown in Figure 2

As an example, consider applying an in/out-degree filter to the problem in Figure 2. The in-degree and out-degree of each template and world node are listed in Table 3. Node **A** has one out-going edge and one in-coming edge, thus nodes **2** and **5** can be ruled out as candidates since node **2** has no out-going edges and node **5** has no in-coming edges. Similarly, node **B** has one in-coming edge, so node **5** can be ruled out as a candidate since it does not have any in-coming edges. Finally, node **C** has two out-going edges and one in-coming edge, so all of the world nodes except **4** are eliminated as candidates since **4** is the only world node with at least two out-going edges *and* at least one in-coming edge. The candidates for each template node after applying the node-level statistics filter are summarized in the "Statistics" column of Table 4.

	Filters run		
	statistics	statistics, topology	statistics, topology, repeated-sets
Candidates for **A**	**1**, **3**, **4**	**1**	**1**
Candidates for **B**	**1**, **2**, **3**, **4**	**1**, **2**	**2**
Candidates for **C**	**4**	**4**	**4**

TABLE 4: Candidates per template node for the problem shown in Figure 2 after various filters have been applied.

The node-level statistics filter is the second cheapest filter to apply, after the node label filter, and is most effective when some template nodes have local structure that is uncommon in the world. Narrowing down the candidates for even one template node can enhance the ability of the other filters to refine the candidates for the remaining template nodes.

2.3 Topology Filter

In the topology filter, we enforce the constraint proposed by [61], extended to multiplex networks. The original constraint, denoted as *AC(Edges)* [57], is that if $\mathbf{v}_w$ is a candidate for template node $\mathbf{v}_t$, then for every template node $\mathbf{u}_t$ neighboring $\mathbf{v}_t$, there must exist a candidate $\mathbf{u}_w$ for $\mathbf{u}_t$ that neighbors $\mathbf{v}_w$. The natural extension to multiplex networks

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSE.2021.3056329, IEEE Transactions on Network Science and Engineering

is that if $\mathbf{v}_w$ is a candidate for template node $\mathbf{v}_t$, then for every template node $\mathbf{u}_t$ neighboring $\mathbf{v}_t$, there must exist a candidate $\mathbf{u}_w$ for $\mathbf{u}_t$ which has as many edges in each channel and direction between $\mathbf{v}_w$ and $\mathbf{u}_w$ as there are between $\mathbf{v}_t$ and $\mathbf{u}_t$.

Algorithm 3 Topology Filter

Input template $\mathcal{G}_t$, world $\mathcal{G}_w$, candidate set $D(\mathbf{v}_t)$ for each $\mathbf{v}_t \in \mathcal{V}_t$
for neighboring template nodes $\mathbf{v}_t$ and $\mathbf{u}_t$
  for candidate $\mathbf{v}_w \in D(\mathbf{v}_t)$
    `nbr_cand_found` $\leftarrow$ False
    for candidate $\mathbf{u}_w \in D(\mathbf{u}_t)$
      `enough_out` $\leftarrow \mathcal{E}_w(\mathbf{v}_w, \mathbf{u}_w) \geq \mathcal{E}_t(\mathbf{v}_t, \mathbf{u}_t)$
      `enough_in` $\leftarrow \mathcal{E}_w(\mathbf{u}_w, \mathbf{v}_w) \geq \mathcal{E}_t(\mathbf{u}_t, \mathbf{v}_t)$
      if `enough_out` *and* `enough_in`
        `nbr_cand_found` $\leftarrow$ True
        break
    if `nbr_cand_found` *is not* True
      Remove $\mathbf{v}_w$ from $D(\mathbf{v}_t)$
Output updated candidate set $D(\mathbf{v}_t)$ for each $\mathbf{v}_t \in \mathcal{V}_t$

To clarify the concept, consider applying the topology filter to the problem in Figure 2 after already applying the node-level statistics filter. Since node **4** is the only candidate for node **C**, nodes **3** and **4** are eliminated from the candidate lists for both nodes **A** and **B** because they do not have neighbors that are candidates for node **C**. This results are shown in Table 4.

In terms of computational complexity, the topology filter scales linearly with the number of distinguishable template edges, and with the number of world nodes squared. In practice, by using sparse matrices, the second factor can be reduced to the number of distinguishable world edges. Thus, the computational complexity is $O(|\mathcal{E}_t|_0 \, |\mathcal{E}_w|_0)$.

The topology filter is particularly useful when one or more template nodes have few candidates remaining after applying other filters. In such situations, it can significantly reduce the candidates for neighboring template nodes.

2.4 Repeated-Sets Filter

Here, we apply another constraint, *GAC(AllDiff)* [57], [64]: generalized arc consistency (also known as hyper-arc consistency) for the alldifferent constraint. *GAC(AllDiff)* requires that for a world node $\mathbf{v}_w$ to be a candidate for template node $\mathbf{v}_t$, there must exist some injective mapping from the template nodes to their candidates under which $\mathbf{v}_t$ is mapped to $\mathbf{v}_w$. To enforce *GAC(AllDiff)*, we identify sets of template nodes $\mathcal{T} \subseteq \mathcal{V}_t$ where the union of their candidates $\bigcup_{\mathbf{v}_t \in \mathcal{T}} D(\mathbf{v}_t)$ has the same cardinality as $\mathcal{T}$. These are known as *tight sets* [64]. Candidates for template nodes in a tight set cannot be candidates for any nodes outside of the tight set, since this would violate *GAC(AllDiff)*.

Standard algorithms for enforcing *GAC(AllDiff)* run in $O\Big(\sqrt{|\mathcal{V}_t| + |\mathcal{V}_w|} \sum_{\mathbf{v}_t \in \mathcal{V}_t} |D(\mathbf{v}_t)|\Big)$ time complexity [23], [51]. Some improvements and modifications to the standard algorithms have been explored to mixed benefit [23]. In the repeated-sets filter (Algorithm 4), we enforce *GAC(AllDiff)* by directly considering unions of candidate sets to find tight sets. Though the number of candidate set unions grows exponentially with the number of template nodes [64], we restrict this growth by not considering unions with more nodes than there are template nodes. We also keep track of template nodes which belong to tight sets and do not use them in unions. In practice, template sizes are often small, and we don't observe the worst case exponential scaling. In terms of the world graph, naive maximum cardinality matching-based algorithms scale as $O(|\mathcal{V}_w|^{3/2})$, whereas Algorithm 4 scales linearly with $|\mathcal{V}_w|$.

Algorithm 4 Repeated-Sets Filter

Input template $\mathcal{G}_t$, world $\mathcal{G}_w$, candidate set $D(\mathbf{v}_t)$ for each $\mathbf{v}_t \in \mathcal{V}_t$
`unions` $\leftarrow$ `EmptyMap()`
▷ Map sets of world nodes $\mathcal{U}_w$ to sets of template nodes $\mathcal{U}_t$ for which $\mathcal{U}_w = \bigcup_{\mathbf{v}_t \in \mathcal{U}_t} D(\mathbf{v}_t)$. Any $\mathcal{U}_t$ with $|$`unions`$[\mathcal{U}_t]| == |\mathcal{U}_t|$ is a tight set.
$\mathcal{T} \leftarrow \emptyset$ ▷ Nodes known to belong to tight sets.
`todo` $\leftarrow \{\mathbf{v}_t \in \mathcal{V}_t : |D(\mathbf{v}_t)| < |\mathcal{V}_t|\}$
▷ Template nodes with too many candidates cannot belong to a tight set.
while $|$`todo`$| > 0$
  $\mathbf{v}_t \leftarrow$ element of `todo` with fewest candidates $D(\mathbf{v}_t)$
  `todo` $\leftarrow$ `todo` $\setminus \{\mathbf{v}_t\}$
  for $(\mathcal{U}_w, \mathcal{U}_t) \in$ `unions`
    $\mathcal{U}_w \leftarrow \mathcal{U}_w \cup D(\mathbf{v}_t)$
    if $|\mathcal{U}_w| < |\mathcal{V}_t|$
      $\mathcal{U}_t \leftarrow \mathcal{U}_t \cup \{\mathbf{v}_t\}$
      if $|\mathcal{U}_t| == |\mathcal{U}_w|$ ▷ $\mathcal{U}_t$ is a tight set.
        $\mathcal{T} \leftarrow \mathcal{T} \cup \mathcal{U}_t$
        $D(\mathbf{u}_t) \leftarrow D(\mathbf{u}_t) \setminus \mathcal{U}_w$ for $\mathbf{u}_t \in \mathcal{V}_t \setminus \mathcal{U}_t$
      else
        `unions`$[\mathcal{U}_w] \leftarrow \mathcal{U}_t$
  if $D(\mathbf{v}_t) \notin$ `unions`
    if $|D(\mathbf{v}_t)| == 1$ ▷ $\{\mathbf{v}_t\}$ is a tight set.
      $\mathcal{T} \leftarrow \mathcal{T} \cup \{\mathbf{v}_t\}$
      $D(\mathbf{u}_t) \leftarrow D(\mathbf{u}_t) \setminus D(\mathbf{v}_t)$ for $\mathbf{u}_t \in \mathcal{V}_t \setminus \{\mathbf{v}_t\}$
    else
      `unions`$[D(\mathbf{v}_t)] \leftarrow \{\mathbf{v}_t\}$
  `todo` $\leftarrow \{\mathbf{v}_t \in$ `todo` $: |D(\mathbf{v}_t)| < |\mathcal{V}_t \setminus \mathcal{T}|\}$
Output updated candidate set $D(\mathbf{v}_t)$ for $\mathbf{v}_t \in \mathcal{V}_t$

To illustrate the application of the repeated-sets filter, consider the example in Figure 2. Using the node-level statistics and topology filters, the candidates for the template nodes were narrowed down to those in Table 4. Since **A** has only one candidate, $\{\mathbf{A}\}$ is a tight set and **1** can be removed as a candidate for the remaining template nodes. This leaves only one candidate for each template node, exactly corresponding to the only signal in Figure 2.

The repeated-sets filter is most important when some template nodes have only one candidate, since any template node with only one candidate forms a tight set. It is also useful for templates such as those discussed in Sections 4.4.1 and 4.4.5 which contain some nodes that are structurally interchangeable.

2.5 Neighborhood Filter

In this filter, we extend the local alldifferent (LAD) constraint introduced in [57] to the multiplex network case. In the undirected single-layer graph context, the LAD constraint ensures that for a world node $\mathbf{v}_w$ to be a candidate for a template node $\mathbf{v}_t$, there must be some injective mapping f_ℓ from the neighbors of $\mathbf{v}_t$ to their candidates under which $f_\ell(\mathbf{u}_t)$ is a neighbor of $\mathbf{v}_w$ for each $\mathbf{u}_t$. We extend this to the multiplex network context by requiring not just that $f_\ell(\mathbf{u}_t)$ neighbors of $\mathbf{v}_w$, but also that there are enough edges in each channel and direction

$$\begin{aligned} \mathcal{E}_w(\mathbf{v}_w, \mathbf{u}_w) &\geq \mathcal{E}_t(\mathbf{v}_t, \mathbf{u}_t) \\ \text{and} \quad \mathcal{E}_w(\mathbf{u}_w, \mathbf{v}_w) &\geq \mathcal{E}_t(\mathbf{u}_t, \mathbf{v}_t) \end{aligned} \tag{1}$$

where $\mathbf{u}_w = f_\ell(\mathbf{u}_t)$. In the neighborhood filter (Algorithm 5), we enforce the multiplex LAD constraint by searching for such a mapping f_ℓ. If none exists, we eliminate $\mathbf{v}_w$ as a candidate for $\mathbf{v}_t$.

We now transform the search for a mapping f_ℓ into a matching problem on a bipartite graph $\mathcal{B}$. Let $\mathcal{N}_{\mathbf{v}_t}$ and $\mathcal{N}_{\mathbf{v}_w}$ denote the neighborhoods of $\mathbf{v}_t$ and $\mathbf{v}_w$ respectively. Define the undirected bipartite graph $\mathcal{B}$ with parts $\mathcal{N}_{\mathbf{v}_t}$ and $\mathcal{N}_{\mathbf{v}_w}$ to have an edge between nodes $\mathbf{u}_t \in \mathcal{N}_{\mathbf{v}_t}$ and $\mathbf{u}_w \in \mathcal{N}_{\mathbf{v}_w}$ whenever $\mathbf{u}_w$ is a candidate for $\mathbf{u}_t$ and Equation (1) holds. A matching on $\mathcal{B}$ is a subset of its edges where no two edges share a node. A mapping f_ℓ is equivalent to a matching on $\mathcal{B}$ of size $|\mathcal{N}_{\mathbf{v}_t}|$ where each edge $(\mathbf{u}_t, \mathbf{u}_w)$ in the matching corresponds to $f_\ell(\mathbf{u}_t) = \mathbf{u}_w$. The Hopcroft-Karp algorithm [27] can be used to find the maximum cardinality matching on $\mathcal{B}$ in $O(\sqrt{|\mathcal{N}_{\mathbf{v}_t}| + |\mathcal{N}_{\mathbf{v}_w}|}\,|\mathcal{E}_\mathcal{B}|)$ time, where $\mathcal{E}_\mathcal{B}$ denotes the set of edges of $\mathcal{B}$.

Algorithm 5 Neighborhood Filter

Input template $\mathcal{G}_t$, world $\mathcal{G}_w$, candidate set $D(\mathbf{v}_t)$ for each $\mathbf{v}_t \in \mathcal{V}_t$
for template node $\mathbf{v}_t \in \mathcal{V}_t$
  for candidate $\mathbf{v}_w \in D(\mathbf{v}_t)$
    $\mathcal{N}_{\mathbf{v}_t} \leftarrow$ `Neighborhood`$(\mathbf{v}_t)$
    $\mathcal{N}_{\mathbf{v}_w} \leftarrow$ `Neighborhood`$(\mathbf{v}_w)$
    $\mathcal{B} \leftarrow$ `EmptyBipartiteGraph`$(\mathcal{N}_{\mathbf{v}_t}, \mathcal{N}_{\mathbf{v}_w})$
    for $(\mathbf{u}_t, \mathbf{u}_w) \in \mathcal{N}_{\mathbf{v}_t} \times \mathcal{N}_{\mathbf{v}_w}$
      if $\begin{cases} \mathbf{u}_w \in D(\mathbf{u}_t) \\ \textit{and } \mathcal{E}_w(\mathbf{v}_w, \mathbf{u}_w) \geq \mathcal{E}_t(\mathbf{v}_t, \mathbf{u}_t) \\ \textit{and } \mathcal{E}_w(\mathbf{u}_w, \mathbf{v}_w) \geq \mathcal{E}_t(\mathbf{u}_t, \mathbf{v}_t) \end{cases}$
        Add an edge between $\mathbf{u}_t$ and $\mathbf{u}_w$ in $\mathcal{B}$
    `max_match` = `MaxCardinalityMatching`$(\mathcal{B})$
    if $|$`max_match`$| < |\mathcal{N}_{\mathbf{v}_t}|$
      Remove $\mathbf{v}_w$ from $D(\mathbf{v}_t)$
Output updated candidate set $D(\mathbf{v}_t)$ for each $\mathbf{v}_t \in \mathcal{V}_t$

In the example from earlier, in Figure 2, the node-level statistics, topology, and repeated-sets filters were sufficient to narrow down the candidates until they solve the MCSP (Definition 1.7). In Figure 3, we give an example where those filters are not sufficient. The resulting candidates before and after neighborhood filter are given in Table 5. Before the neighborhood filter is applied, node **4** remains a candidate for node **C** because node **3** is a candidate for its neighbors **A** and **B**. The biadjacency matrix of $\mathcal{B}$ corresponding to $\mathbf{v}_t = \mathbf{C}$ and $\mathbf{v}_w = \mathbf{4}$ is shown in Table 6. Since the maximum cardinality matching on $\mathcal{B}$ has only two elements, the neighborhood filter is able to eliminate node **4** as a candidate for node **C**.

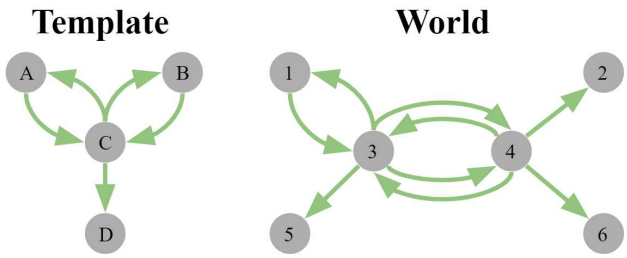


Fig. 3: An example template and world for which the neighborhood filter plays a role in eliminating candidates.

	Filters run	
	statistics, topology, repeated-sets	statistics, topology, repeated-sets, neighborhood
Candidates for **A**	**1**, **3**, **4**	**1**, **4**
Candidates for **B**	**1**, **3**, **4**	**1**, **4**
Candidates for **C**	**3**, **4**	**3**
Candidates for **D**	**2**, **5**, **6**	**5**

TABLE 5: Candidates per template node for the problem shown in Figure 3 after various filters have been applied.

$\mathcal{N}_\mathbf{C}$ \ $\mathcal{N}_\mathbf{4}$	**2**	**3**	**6**
A	0	1	0
B	0	1	0
D	1	0	1

TABLE 6: Biadjacency matrix of $\mathcal{B}$ used in the matching problem between the neighborhoods of template node **C** and world node **4**.

2.6 Elimination Filter

The elimination filter (Algorithm 6) attempts to eliminate candidates by identifying any contradictions that would result from them being assigned. For each template node-candidate pair ($\mathbf{v}_t$,$\mathbf{v}_w$), we do a one step lookahead. We assign $\mathbf{v}_w$ to $\mathbf{v}_t$ and iterate over all other filters until convergence. If this results in one or more template nodes having no candidates, then $\mathbf{v}_t$ cannot be mapped to $\mathbf{v}_w$ and we eliminate $\mathbf{v}_w$ as a candidate for $\mathbf{v}_t$.

As the elimination filter is a very expensive operation, scaling with the number of remaining candidates, we restrict its use until all other filters have converged and no further candidates can be removed by other means. In certain contexts, the elimination filter can be impractical to use. However, in others, it can greatly reduce the number of candidates. A simple example where elimination filter proves useful can be seen in Figure 4, where we have three cycle graphs consisting of 3, 4 and 5 nodes respectively. We also assume that each edge is bidirectional and there is only one channel.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSE.2021.3056329, IEEE Transactions on Network Science and Engineering

Algorithm 6 Elimination Filter

Input template $\mathcal{G}_t$, world $\mathcal{G}_w$, candidate set $D(\mathbf{v}_t)$ for each $\mathbf{v}_t \in \mathcal{V}_t$
$S \leftarrow \texttt{copy}(D)$ ▷ Save the candidate sets
for template node $\mathbf{v}_t \in \mathcal{V}_t$
  for candidate $\mathbf{v}_w \in D(\mathbf{v}_t)$
    $D(\mathbf{v}_t) \leftarrow \{\mathbf{v}_w\}$ ▷ Assign $\mathbf{v}_w$ to $\mathbf{v}_t$
    Iterate all filters to convergence
    if $D(\mathbf{u}_t)$ is empty for any $\mathbf{u}_t \in \mathcal{V}_t$
      Remove $\mathbf{v}_w$ from $D(\mathbf{v}_t)$ in S
    Reset candidate sets to S
Output updated candidate set $D(\mathbf{v}_t)$ for each $\mathbf{v}_t \in \mathcal{V}_t$

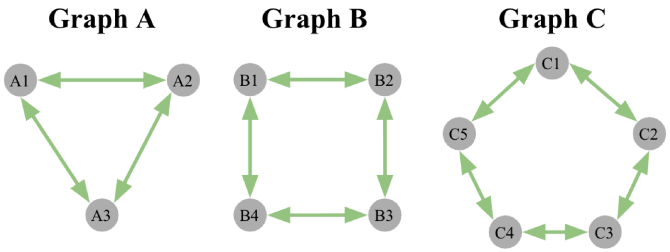


Fig. 4: Subgraph matching problems consisting of Graph A, B and C.

Suppose we use Graph A as the template graph and Graph B as the world graph. After applying the node-level statistics, topology, and neighborhood filters, every node in Graph B will remain a candidate of every template node in Graph A. By including elimination filter, we will be able to correctly tell that there is no valid signal for this problem. A more difficult problem arises when we use Graph B as the template graph and Graph C as the world graph. In this case, the node-level statistics, topology, and neighborhood filters will no longer get rid of any invalid candidates. Only after additionally applying the elimination filter can we conclude that there is no valid signal existing in Graph C.

In practice, the elimination filter that iterates over node-level statistics, topology, and repeated-sets filters is often sufficient, since iterating over neighborhood filters is expensive. However, in situations like the second example posed above, we might need to apply all of the existing filters to determine the solution. In particular, the elimination filter is useful in many synthetic and real-world datasets, especially in some of the examples discussed in Section 4.4.

3 Solving the Problems

We present the details of how to solve the SICP and MCSP using the filters from Section 2 as a subroutine. The SIP and SMP can be solved with similar methods to SICP, and the SNSP can be solved with a similar method to MCSP.

3.1 Isomorphism Counting

After applying the filters described in Section 2, some template nodes may have exactly one candidate. Template nodes that still have multiple candidates we refer to as *unspecified* nodes. When an edge exists between two unspecified nodes, we have to enforce that a corresponding edge exists between the two candidates we choose for them. As this makes counting isomorphisms computationally complex, we start by finding a set of unspecified nodes which, if specified, would cause the remaining unspecified nodes to have no edges between them. This set is called a *node cover*, and the smallest such set is called the *minimal node cover*.

For example, in Figure 2, if all three template nodes have multiple candidates, the minimal node cover would be $\{\mathbf{C}\}$. Since the minimal node cover is expensive to compute in general, we settle for a small node cover [4].

Next, we iterate through all possible choices of candidates for nodes in the node cover. For each choice, we reapply the topology and repeated-sets filters so we can be sure that any remaining candidates belong to signals. Since the remaining unspecified nodes have no edges between them, it is much simpler to count the ways to choose their candidates. The only constraint is that the same candidate cannot be chosen for more than one node, which is simply the alldifferent constraint satisfaction problem [51].

Algorithm 7 Isomorphism Counting

function IsoCount
  Input template $\mathcal{G}_t$, world $\mathcal{G}_w$, candidate set $D(\mathbf{v}_t)$ for each $\mathbf{v}_t \in \mathcal{V}_t$, unspecified node cover $\mathcal{NC}$
  if $\mathcal{NC} = \emptyset$
    $\texttt{N}_{\texttt{isos}} \leftarrow \texttt{CountAllDiff}(D)$
  else
    $\texttt{N}_{\texttt{isos}} \leftarrow 0$
    $\mathbf{v}_t \leftarrow \mathcal{NC}.\texttt{pop}()$
    $S \leftarrow \texttt{copy}(D)$ ▷ Save the candidate sets
    for world node $\mathbf{v}_w \in D(\mathbf{v}_t)$
      $D(\mathbf{v}_t) \leftarrow \{\mathbf{v}_w\}$ ▷ Assign $\mathbf{v}_w$ to $\mathbf{v}_t$
      Apply filters
      $\texttt{N}_{\texttt{isos}} \leftarrow \texttt{N}_{\texttt{isos}} + \texttt{IsoCount}(\mathcal{G}_t, \mathcal{G}_w, D, \mathcal{NC})$
      Reset candidate sets to S
  Output number of isomorphisms $\texttt{N}_{\texttt{isos}}$

Simple modifications can be made in order to solve the SIP and SMP. For the SIP, the algorithm exits and returns true once a solution is found, i.e. once `CountAllDiff` returns a nonzero result. For the SMP, replace `CountAllDiff` with an algorithm that returns a list of all solutions to the alldifferent problem, and instead of adding the counts together, combine the two lists of solutions.

3.2 Validation

A simpler problem than listing or counting all isomorphisms (SMP or SICP) is the minimal candidate sets problem (MCSP) of Definition 1.7. Here, we refer to the procedure for solving the MCSP as *validation*, in which we we seek to identify all template node-candidate pairs that participate in at least one isomorphism. An algorithm for validation is as follows (Algorithm 8). First, pick an unvalidated pair of a template node and its candidate. Next, make assignments until an isomorphism is found, running filters after each step and backtracking as necessary. If all possible assignments fail, then the pair is invalid; remove the pair from the set of possible candidates. Otherwise, validate every assignment made as part of the isomorphism, including the initial one; these pairs have been validated as taking

part in at least one isomorphism. Repeat this process until all pairs have been validated or removed from candidates. This ensures that every candidate-template node pairing remaining after validation is valid i.e. participates in at least one signal. This is because in order to pass validation, it must participate in at least one isomorphism that was found, and in order to fail, there must not exist an isomorphism containing it.

An optimization for Algorithm 8 when using the node cover approach in Section 3.1 is that if the node cover has been assigned, the topology and repeated-sets filters have been run to convergence, and the problem has thus been reduced to an alldifferent problem. Then, instead of attempting to find a single solution to the alldifferent problem, all possible candidate pairs corresponding to the alldifferent problem can be simultaneously validated. This is because all remaining candidate pairs passed the repeated-sets filter, which enforces *GAC(AllDiff)*: thus, there must exist a solution for each pair, and thus an isomorphism since edges were previously enforced by the topology filter.

Algorithm 8 Validation

Input template $\mathcal{G}_t$, world $\mathcal{G}_w$, candidate set $D(\mathbf{v}_t)$ for each $\mathbf{v}_t \in \mathcal{V}_t$
$D'(\mathbf{v}_t) \leftarrow \emptyset$ for each $\mathbf{v}_t \in \mathcal{V}_t$ ▷ Valid candidates
for template node $\mathbf{v}_t \in \mathcal{V}_t$
  for candidate $\mathbf{v}_w \in D(\mathbf{v}_t) \setminus D'(\mathbf{v}_t)$
    $S \leftarrow \texttt{copy}(D)$
    $S(\mathbf{v}_t) \leftarrow \{\mathbf{v}_w\}$ ▷ Assign $\mathbf{v}_w$ to $\mathbf{v}_t$
    Attempt to find an SI f using S ▷ Solve the SIP
    if isomorphism found
      $D'(\mathbf{u}_t) \leftarrow D'(\mathbf{u}_t) \cup \{f(\mathbf{u}_t)\}$ for each $\mathbf{u}_t \in \mathcal{V}_t$
    else
      Remove $\mathbf{v}_w$ from $D(\mathbf{v}_t)$
Output minimal candidate set $D'(\mathbf{v}_t)$ for each $\mathbf{v}_t \in \mathcal{V}_t$

This algorithm can also be modified to solve the SNSP. If, instead of adding the tuple $(\mathbf{u}_t, f(\mathbf{u}_t))$ to `validated`, only the element $f(\mathbf{u}_t)$ is added, then the algorithm will validate only which world nodes participate, without respect to which template nodes they are candidates for.

4 Experiments

All experiments were run on an HP Z8 G4 Workstation with two 12-core 6136 3.0 2666MHz CPUs, using version 0.2.0 of the python code available on GitHub (https://github.com/jdmoorman/uclasm/tree/v0.2.0) [44]. The node-level statistics, topology, repeated-sets filters run in under a minute for all datasets and under a second for most. The elimination filter takes longer, between minutes and hours, depending on the dataset. Validation, if used, can also take hours or longer. The combinatorial complexity of the solution space is largely responsible for the long time needed for the elimination, counting and validation.

4.1 Sudoku

The puzzle game of Sudoku involves a 9x9 grid that is partitioned into nine 3x3 *blocks*. Each *cell* of the grid must be filled with a digit 1-9 so that each row, column, and block contains each digit 1-9 exactly once. A Sudoku puzzle begins with some of the cells pre-filled with digits so that there is exactly one correct solution (Figure 5).

	1			8			9	
6		4	9		2	5		3
	5	6	2		8	9	3	
2								1
	3	8	1		7	6	2	
3		1	8		6	2		5
	9			7			8	

Fig. 5: Sudoku grid with initial clues.

One algorithm for solving Sudoku is Donald Knuth's dancing links algorithm [33], which models Sudoku as an exact cover problem. This is a backtracking tree search with refinement on a sparse boolean matrix, using doubly linked lists for efficiency. Sudoku can also be modeled as a constraint satisfaction problem [56], see e.g. Peter Norvig's constraint propagation algorithm [47]. Here, the two constraints propagated were that cells in the same row must contain different digits and that every cell must have a digit. After these constraints have been iterated, the algorithm uses a depth first backtracking search to find the solution.

The constraint satisfaction problem to solve Sudoku has much in common with subgraph isormorphism algorithms; here we show that Sudoku can be written in multiple ways as a subgraph isomorphism problem. We chose this example to test our algorithms primarily because constraint propagation is considered state of the art for Sudoku. Here, our goal is to validate the reasonable use of our code on this problem, rather than trying to beat the best Sudoku solvers.

4.1.1 9x9 Representation

Sudoku is equivalent to a single-channel subgraph isomorphism problem using the following correspondence: consider the cells as nodes of the template, two cells in the template are linked if they are in the same row, column, or 3x3 block. We consider the world as a set of 81 nodes, each corresponding to a digit. Each digit is locked to a particular 3x3 block, leading to nine sets of nine digits each, having values 1-9. And here, two nodes are linked if they do *not* have the same value. This graph has the same number of nodes, but more edges than the template, so this is a subgraph isomorphism problem. To represent known digits (to start the puzzle), we can restrict the initial candidates. This can be done simply: if a cell is filled in with a value, we identify which 3x3 block the cell is in, find the digit in the world that corresponds to that block and has the matching value, and say that the only candidate for the cell is that digit. The Sudoku puzzle is thus a semi-supervised version of the subgraph graph isomorphism problem and is designed to have a unique solution, unlike the unsupervised examples in our other experiments.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSE.2021.3056329, IEEE Transactions on Network Science and Engineering

4.1.2 9x9x3 Representation

One can also assign different edge types (i.e. row, column or block correspondences) to different channels, along with another channel that identifies which cells are the same. The template has three nodes for each cell, a row-node, a column-node, and a block-node. All three are linked by edges in a special "same-cell" channel. Otherwise, the three channels are completely separate: row-nodes are only linked to other row-nodes in the same row, column-nodes to column-nodes, etc. The world is then three channels of nine digits each, so there are once again the same number of world nodes as template nodes. In each channel, sets of nine digits are identified as being in the same row/column/block, and in these channels, there are the same number of edges as there in the template. However, for the "same-cell" channel, we add an edge between any two nodes that could represent the same cell. Since each 3x3 block only intersects three rows and three columns, these edges are restricted. Similar to before, we add an additional initial restriction that a row-node in the template can only have the nine row-digits in the world corresponding to its row as candidates, and the same for columns and blocks.

4.1.3 Results

To test our filters, we use the sets of Sudoku puzzles obtained from Peter Norvig's GitHub [48], including those from Project Euler problem 96 [50]. Though our code is not specialized for the task, and is thus slower, it solves all Sudoku examples in the dataset. Overall, it appears that 9x9 works best on the easier puzzles, but 9x9x3 is faster on harder puzzles. On the hardest puzzles, 9x9 is faster on average; this is primarily due to outliers for which the 9x9x3 takes much longer.

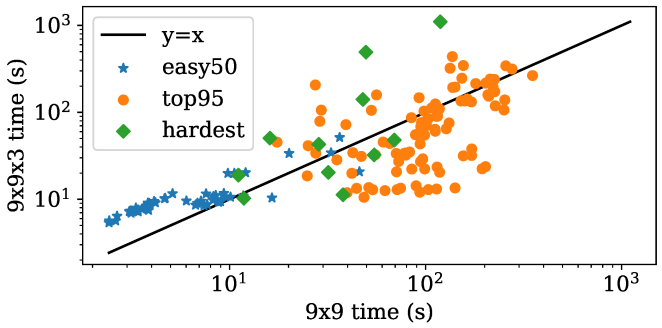


Fig. 6: Solving Sudoku puzzles as special case of a multiplex subgraph isomorphism problem using the 9x9 and 9x9x3 representations. Scatter plot of solution times (seconds) on the Sudoku puzzles from Peter Norvig's GitHub [48], including those from Project Euler problem 96 [50]. A black line along $y = x$ aids in comparison. Average solution time for 9x9 is 8.33 seconds on the 50 easy puzzles, 116.7 seconds on the top 95 puzzles, and 43.4 seconds on the hardest puzzles. Average solution time for 9x9x3 is 11.24 seconds on the 50 easy puzzles, 95.4 seconds on the top 95 puzzles, and 179.4 seconds on the hardest puzzles.

4.2 Multiplex Erdős-Rényi

To test the scalability of our algorithm, we perform experiments on randomly generated multiplex Erdős-Rényi networks, where each edge in each channel has the same probability p of occurring. For the templates, $p_t = 0.5$. For the worlds, the probability

$$p_w = 1 - \frac{1 - \sqrt[|\mathcal{C}|]{1 - p_t + p_t^2}}{p_t}$$

is scaled up as the number of channels increases to maintain a fixed probability of length $|\mathcal{C}|$ edge vectors matching

$$\mathbb{P}(\mathcal{E}_t(\mathbf{u}_t, \mathbf{v}_t) \leq \mathcal{E}_w(\mathbf{u}_w, \mathbf{v}_w)) = 0.75$$

for all template nodes $\mathbf{u}_t$, $\mathbf{v}_t$ and world nodes $\mathbf{u}_w$, $\mathbf{v}_w$. Templates are generated with 10 nodes, while worlds are generated with sizes ranging from 10 nodes to 300 nodes. We cap the algorithm runtime at 10000 seconds per instance.

To measure the difficulty of each instance, we count the number of search iterations (recursive calls to `IsoCount`) taken by Algorithm 7 to solve the SIP and SICP. For the SIP, the counting is terminated after a single isomorphism is found, or once the entire search space has been checked and no isomorphisms are found. We observe similar difficulty scaling for one, two, and three channel instances (Figure 7).

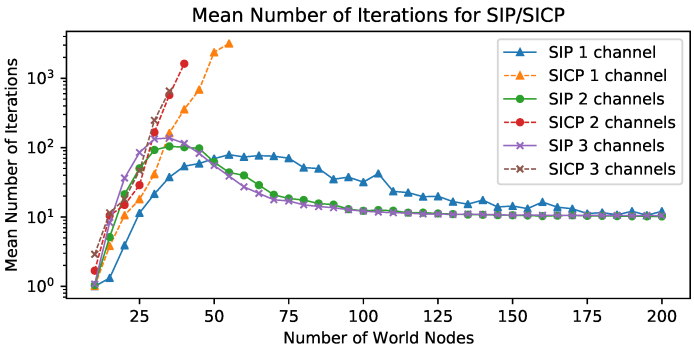


Fig. 7: Mean number of search iterations (recursive calls to `IsoCount`) taken by Algorithm 7 to solve the SIP and SICP with one, two, and three channels as a function of world size. Results are averaged over 500 trials.

For the SIP, there is an initial sharp rise in search iterations with a peak around 100 iterations, followed by a decline back to a constant level of 10 iterations. This pattern is partially due to the SIP being easier when SIs are either extremely likely or extremely unlikely [41]. When SIs are unlikely, as when the world is small, filtering often rules out all SIs before reaching the bottom of the search tree. When SIs are common, as when the world is large, every branch of the search tree is likely to have one, so the number of search iterations converges to the number of template nodes, 10.

For the SICP, the average number of search iterations scales monotonically with the number of world nodes. This is to be expected, as the expected number of SIs to be counted also scales monotonically with the number of world nodes, and the effort taken to count the SIs is closely related to the number of SIs.

4.3 Real-World Examples

We apply the algorithms to three real-world examples. From each dataset, we extract a small subgraph as our template and try to locate its subgraph matchings in the world graph. Table 7 summarizes the sizes and filtering results.

Dataset	Template		World		Channels	Number of isomorphisms	Filters	Problems solved
	Nodes	Edges	Nodes	Edges				
Britain Transportation	53	56	262,377	475,502	5	N/A	S, T, R, E	SIP
Britain Transportation (3km)	53	56	262,377	475,502	5	3.76×10^7	L, S, T, R, E	SIP, SNSP, MCSP, SICP
Higgs Twitter	115	2,668	456,626	5,367,315	4	1.03×10^{14}	S, T, R	SIP, SNSP, MCSP, SICP
Commercial Airlines	37	210	450	7,177	37	3.65×10^9	S, T, R	SIP, SNSP, MCSP, SICP

TABLE 7: Sizes and filtering results on real-world examples in Sections 4.3.1, 4.3.2 and 4.3.3. The last column records the types of problems stated in Section 1.1 that we are able to solve. The names of the filters have been abbreviated: L = Node Label; S = Node-level Statistics; T = Topology; R = Repeated-Sets; N = Neighborhood; E = Elimination.

4.3.1 Great Britain Transportation

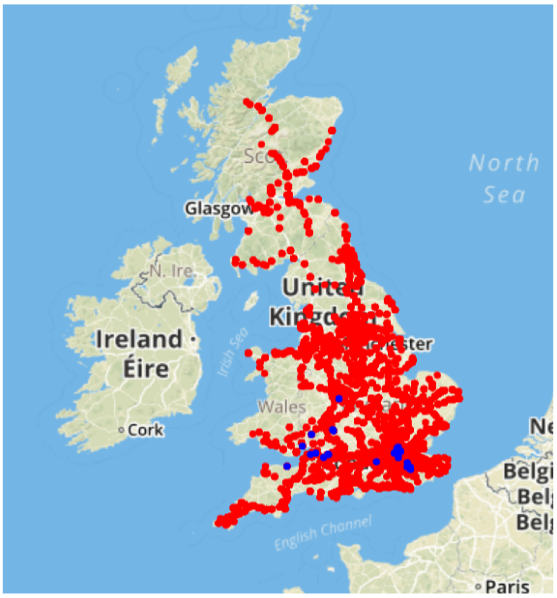


Fig. 8: All candidates for the Great Britain Transportation dataset: blue nodes represent template nodes and red nodes represent candidates for the SIP. An interactive map of the template can be found at [26].

The Great Britain Transportation Network [20] combines the public transportation dataset available through the United Kingdom open-data program [63] with timetables of domestic flights in the UK to obtain a multiplex time-dependent network that reflects the transportation network in the UK. There are six channels representing different transportation methods, including air, ferry, railway, metro, coach, and bus. This dataset has 262,377 nodes and 475,502 edges. The original dataset can be found at [21]. We also created an online interactive map [26] for users to visualize the template.

To test our algorithm, we first create a template. We identify a small set of locations that interact with each other through all channels (excluding airlines, since this channel is very sparse). If a location involves all five non-air channels in the network, we assume that it is important. There are only three nodes that interact in these 5 channels, and we randomly chose one of them as our template center, which is Blackfriars Station in London. Starting from this node, we use a random walk to create a template, which has 53 nodes and 56 edges. The results are shown in Figure 9.

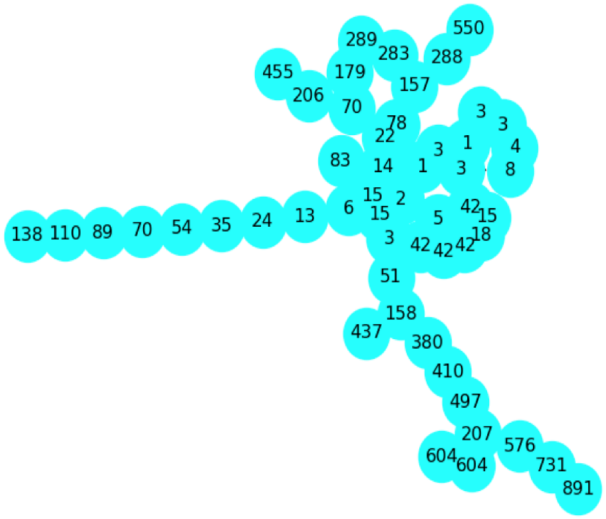


Fig. 9: Candidate count for each node in the Great Britain Transportation template after applying the node-level statistics, topology, repeated-sets and elimination filters; without node label filtering.

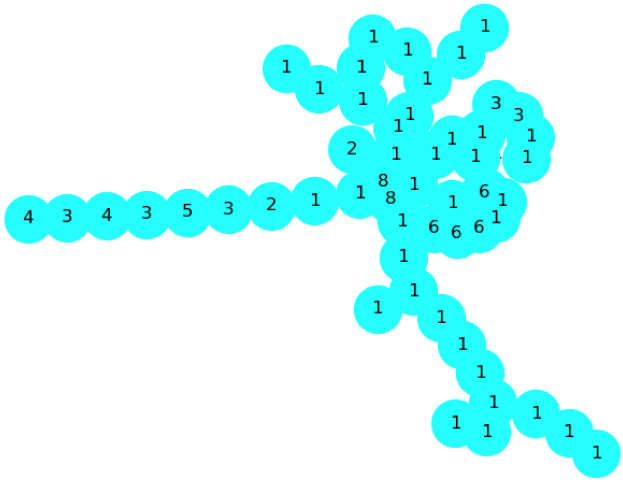


Fig. 10: Candidate count for each node in the Great Britain Transportation template after applying the node label filter with a 3km radius in addition to the node-level statistics, topology, repeated-sets, and elimination filters.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSE.2021.3056329, IEEE Transactions on Network Science and Engineering

We can improve this result using the node label filter; the result is shown in Figure 10. In Figure 8, we see that the remaining candidates after filtering are geographically distributed across the UK. As previously mentioned in Section 2.1, these nodes are actually transportation stops with given latitude and longitudes. If we know *a priori* that a certain node should be confined in a region, say a radius of 3 km, then we can apply the node label filter. Node labels help reduce the size of the world before we apply filtering. Without the node label filter, even after the elimination filter, there are still too many candidates to count. However, if we apply geographical information before we run other filtering algorithms and restrict the candidates to be within 3 km from the coordinates of the template node, then we reduce the size of remaining candidates to a tractable level.

4.3.2 Higgs Twitter

The Higgs Twitter dataset [16] records Twitter activities from July 1-7, 2012, during and after the discovery of the Higgs boson particle. It can be formulated into a directed, multi-edge network with four channels, representing retweets, replies, mentions and friend/follower relationships. The world graph contains 456,626 nodes (Twitter users) and 15,367,315 edges (interactions), and the user identities have been aligned across all channels. To demonstrate how our filtering algorithm performs on detecting a small, relatively dense template, we select a group of 115 Twitter users frequently involved in retweets or replies during the week, and whose induced subgraph is connected, has multi-edges and contains edges in all four channels. The template has a total of 2,668 edges.

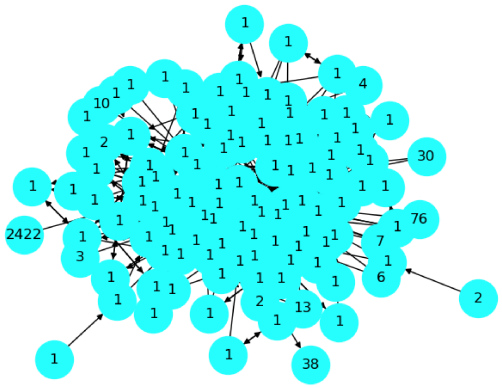


Fig. 11: Candidate count for each node in the Higgs Twitter template after applying the node-level statistics, topology, and repeated-sets filters.

We apply our filtering methods to the world graph and the template. Our methods narrow down the candidates significantly and find exact matches for 102 out of the 115 template nodes (Figure 11). For the template nodes that have multiple remaining candidates, many of these candidates do belong to a valid subgraph isomorphism. We proceed to compute the number of subgraph isomorphisms, totalling 1.03×10^{14} valid isomorphisms. Our method works particularly well in detecting signals that exist across different channels and have multi-edges, as these graph properties provide us with enough information to remove invalid candidates.

4.3.3 Commercial Airlines

We also test our filtering methods on a commercial airlines dataset [8]. This dataset contains a multiplex airline network in Europe which consists of 37 channels, each representing a different airline. Compared with the other real world examples above, its world graph has a smaller scale, only containing 450 nodes (airports) and 7177 directed, unweighted edges (flights).

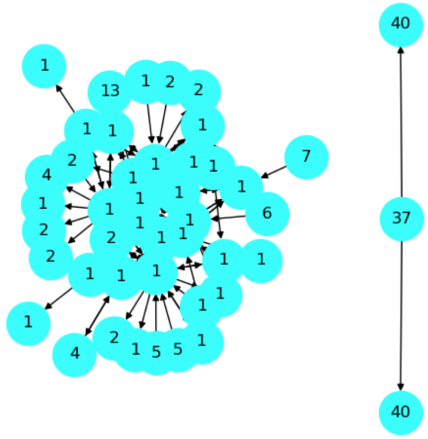


Fig. 12: Candidate count for each node in the commercial airline template after applying the node-level statistics, topology and repeated-sets filters.

We construct a template that has 37 nodes and 210 edges by taking the induced subgraph of 20 core nodes and introducing some additional periphery nodes and edges. When creating the template, we also ensure that it has nodes and edges across all channels to fully test the capability of our algorithm. Due to the small size of the dataset, our filtering algorithm is able to return the final candidate lists of all template nodes within a second, the result of which can be seen in Figure 12. The node-level statistics filter alone manages to find the exact match candidate of 10 templates, while the topology filter increases the number of exact matches to 28 nodes. Our counting algorithm further concludes that there exist around 3.6 million subgraph isomorphisms in the world graph. By using just the topological properties of the multiplex airline network, our algorithm is able to effectively identify the exact locations of most of the target airports as well as the potential locations of the remaining ones.

4.4 Adversarial Activity

We apply our filtering methods to a series of datasets developed for the DARPA MAA program [17], [66]. This program uses networks to represent adversarial activities (e.g., human trafficking, financial transactions, email communication, phone calls, distribution of narcotics). Because these activities can be covert, they may not be detectable through a single activity type. Multiplex networks provide a mathematical structure for identifying related network

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSE.2021.3056329, IEEE Transactions on Network Science and Engineering

modalities for such complex adversarial actions. Thus, an important area of research is to match like patterns from an activity template to part of a larger dataset of multiplex actions. Here, we present some results on datasets created by three different teams: (1) Pacific Northwest National Laboratory (PNNL), (2) the Graphing Observables from Realistic Distributions In Activity Networks (GORDIAN) team, and (3) IvySys Technologies. These datasets consist of a large-scale world graph with one or more roughly 100-node templates, that simulate a scenario of activities by a specific group of agents.

Table 8 summarizes the sizes and filtering results of each dataset. In each instance, there is one world graph with an embedded signal that is isomorphic to the template. We identify the signal with our filtering methods, and when there are multiple matching signals, we perform counting or validation to understand the solution space. For all of the datasets, node-level statistics and topology filters have been applied to eliminate the number of candidates. When the candidate counts remain high, the neighborhood and elimination filters are applied to further narrow down the solution.

4.4.1 PNNL Version 6

PNNL Version 6 [15] has 12 instances, each consisting of one 20k-node world graph and two 100-node templates. They have on the order of 22-23K nodes and over 12 million edges. The templates have 74 to 81 nodes and 1200-1650 edges.

By filtering based on node-level statistics, topology and repeated-sets, we are able to identify the exact signals of most of our given templates. However, there remain templates (e.g., instances B1-S1, B7-S1 according to Table 8) that we cannot fully solve by applying the previous filters. Under such circumstances, we additionally apply the elimination filter, which narrows down the candidate counts significantly.

Figure 13 gives an overview of how different methods of filtering gradually reduce the number of candidates for each template node in instance B1-S1. In the bottom histogram, we can observe that after node-level statistics and topology filters, around 5000 world nodes still remain in the candidate list of some template node. However, the subsequent application of elimination filter reduces the number of candidate world nodes to roughly a hundred. Figure 14 shows the comparison of filtering results on instance B1-S1 before and after adding in the elimination filter. The sharp contrast in their respective candidate counts shows that the elimination filter, in some cases, reduces the size of the candidate domain significantly. After applying all levels of filtering, we observe in Figure 14 that some sets of the template nodes are permutable and share the same candidates, which lead to 1,152 signal isomorphisms in this graph.

The experiments on different instances of the PNNL Version 6 dataset demonstrate the efficacy of node-level statistics and topology filters for solving subgraph matching problems. The filtering results on the more difficult instances also display the potential of the elimination filter, especially in tackling problems that initially appear resistant to the node-level statistics and topology filters.

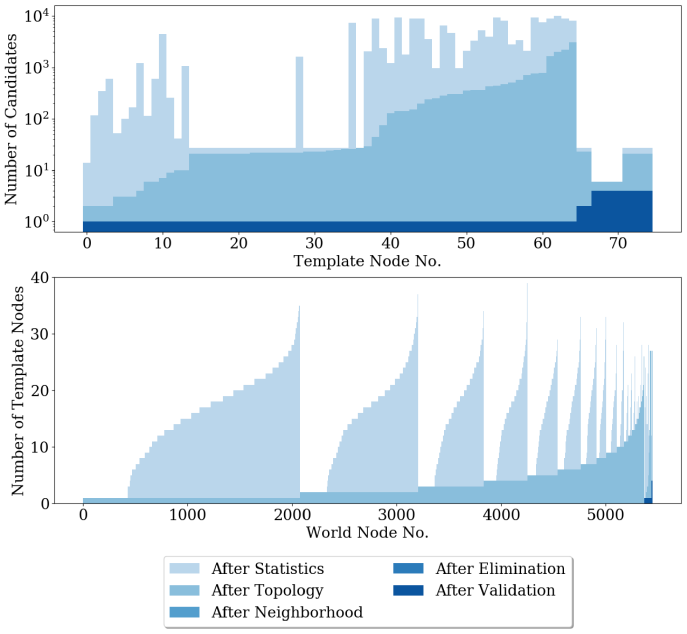


Fig. 13: (Top): The number of candidates for each template node after different levels of filtering are applied to PNNL Version 6 B1-S1. (Bottom): The number of template nodes for which each world node is a candidate.. Note that the Validation histogram perfectly overlaps the Elimination histogram and the Neighborhood histogram perfectly overlaps the Topology histogram.

4.4.2 PNNL Real World

This dataset was created by PNNL from a social media dataset collected by Matteo Magnani and Luca Rossi [12], [39]. It involves friend/follower relationships on three social media platforms, each of which corresponds to a channel.

After applying all of the filters, we identify unique candidates for 20 template nodes. Since many of the remaining template nodes have hundreds of candidates, we proceed to check the validity of these candidates. By additionally applying validation to solve the MCSP, we verify that all of the remaining candidates participate in at least one signal. This result suggests that the filters have the potential to reduce the candidates all the way to the solution of the MCSP.

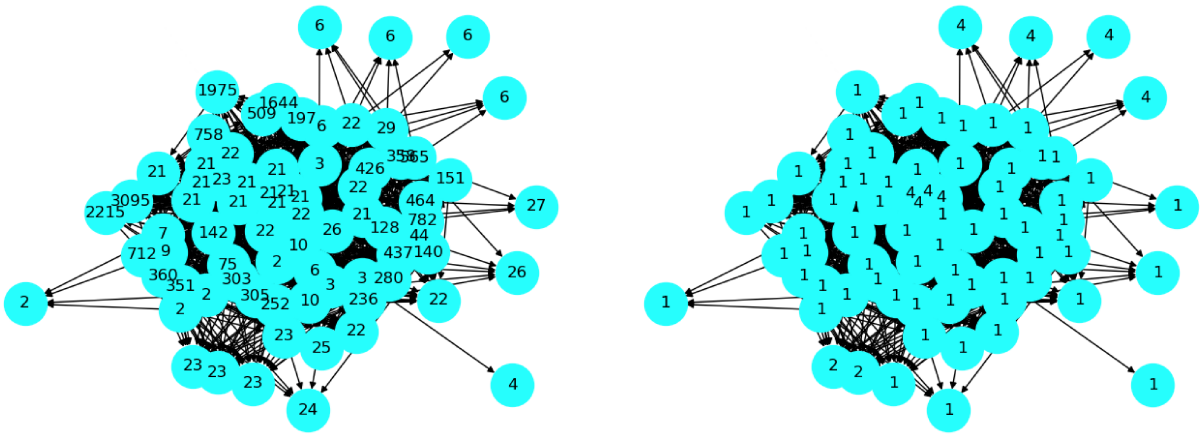
Both histograms in Figure 16 demonstrate how candidates get eliminated as different filters are applied. All filters are able to reduce the number of candidates. In the top histogram, the number of remaining candidates after the elimination filter entirely overlaps with that after validation, confirming that the result after the elimination filter is essentially the solution to the MCSP, identifying all world nodes that correspond to each template node in at least one signal. In the bottom histogram, we can clearly observe how each filter eliminates the candidacy of world nodes in stages.

4.4.3 GORDIAN Version 7

GORDIAN Version 7 [31] has two instances: Batch-1 and Batch-2. Despite the distinction in sizes and structures of their respective templates, both instances can be solved with node-level statistics, topology and repeated-sets filters.

Dataset	Instance	Template		World		Channels	Number of Isomorphisms	Filters	Problems Solved
		Nodes	Edges	Nodes	Edges				
PNNL Version 6	B0-S0	74	1,620	22,996	12,318,861	7	1,152	S, T, R	SIP, SNSP, MCSP, SICP, SMP
	B5-S0	64	1,201	22,994	12,324,975	7	1,152	S, T, R	SIP, SNSP, MCSP, SICP, SMP
	B1-S1	75	1,335	22,982	12,324,340	7	1,152	S, T, R, E	SIP, SNSP, MCSP, SICP, SMP
	B7-S1	81	1,373	23,011	12,327,168	7	3.13×10^8	S, T, R, E	SIP, SNSP, MCSP, SICP
PNNL Real World		35	158	6,407	74,862	3	2.12×10^{12}	S, T, R, N, E	SIP, SNSP, MCSP, SICP
GORDIAN Version 7	Batch-1	156	3,045	190,869	123,267,100	10	4.27×10^{15}	S, T, R, E	SIP, SNSP, MCSP, SICP
	Batch-2	44	715	190,869	123,264,754	10	1.35×10^{16}	S, T, R, E	SIP, SNSP, MCSP, SICP
IvySys Version 7		92	195	2,488	5,470,970	3	N/A	S, T, R, N, E	SIP
IvySys Version 11		103	387	1,404	5,719,030	5	N/A	S, T, R, E	SIP

TABLE 8: Overview of the sizes and filtering results of different DARPA datasets. For each instance of the datasets, the table records its basic statistics, which filters have been applied, and the number of isomorphisms. The last column concludes the types of problems stated in Section 1.1 that we can solve for each instance. The names of the filters have been abbreviated: L = Node Label; S = Node-level Statistics; T = Topology; R = Repeated-Sets; N = Neighborhood; E = Elimination.



(a) Without elimination filter.

(b) With elimination filter.

Fig. 14: Candidate count for each node in the PNNL Version 6 B1-S1 template after applying the node-level statistics, topology, and repeated-sets filters, with and without additionally applying the elimination filter.

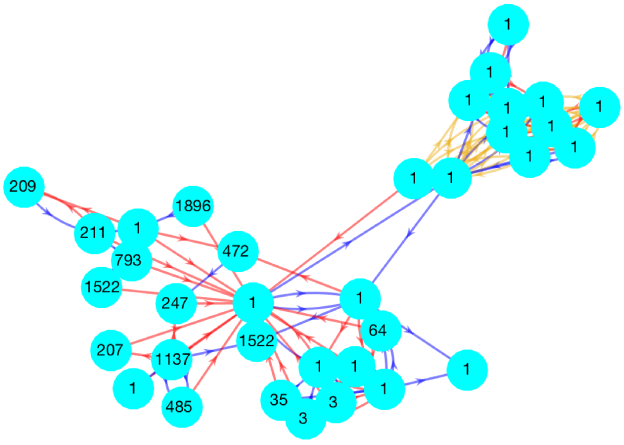


Fig. 15: Candidate count for each node in the PNNL Real World template after solving the MCSP using validation. Edge colors correspond to their channels.

For example, in Batch-1, even without applying the more advanced elimination and neighborhood filters, we manage to find the exact match for 129 template nodes out of 156. In Figure 17, we plot the largest connected component of instance Batch-1, in which we can clearly see that the majority of the template nodes are matched with their single candidate. These filtering results again demonstrate the potential of our basic filters (node-level statistics, topology, repeated-sets) in narrowing down the solution to subgraph matching problems. The small number of remaining candidates also enables us to apply validation and verify that all remaining candidates participate in some signal. We further compute the number of isomorphisms to be 1.51×10^{12}, and the enormous number here is a direct result of the permutability of some template nodes.

4.4.4 *IvySys Version 7*

IvySys Version 7 [3] has three channels corresponding to financial, communication and logistics transactions. Due to the sparse structure of the template, we are unable to identify any template nodes with unique candidates after applying different levels of filtering methods, as seen in

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSE.2021.3056329, IEEE Transactions on Network Science and Engineering

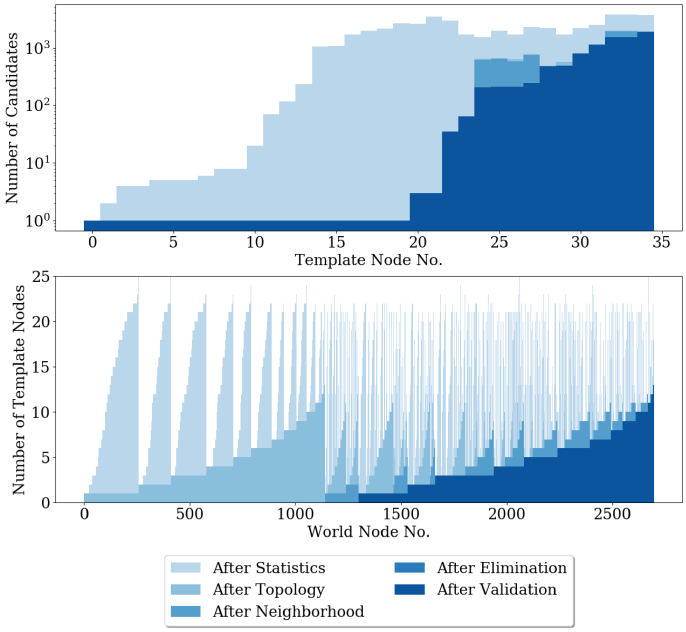


Fig. 16: (Top): The number of candidates for each template node after different levels of filtering are applied to PNNL Real World. (Bottom): The number of template nodes for which each world node is a candidate. Note that the Validation histogram almost perfectly overlaps the Elimination histogram.

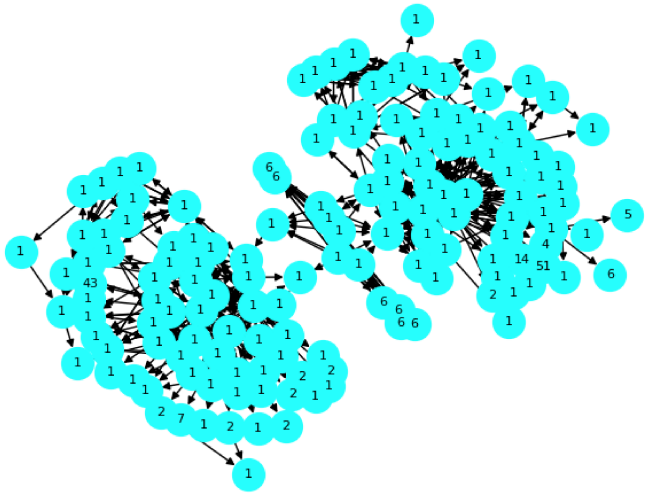


Fig. 17: Candidate count for each node in the GORDIAN Version 7 Batch-1 template after applying the node-level statistics, topology, repeated-sets, and elimination filters.

Figure 18. However, after our filtering methods reduce the size of the search space, we find that there are in fact many signals in the world graph, some of which are almost completely disjoint from each other. It is their existence that leads to the abundance of candidates for each template node. Our approach allows us to easily solve the subgraph isomorphism problem (SIP) for IvySys Version 7, which is finding just one valid signal. However, the enormity of the solution space makes it unreasonable to proceed to solve the rest of the variants, like SICP, SNSP and MCSP.

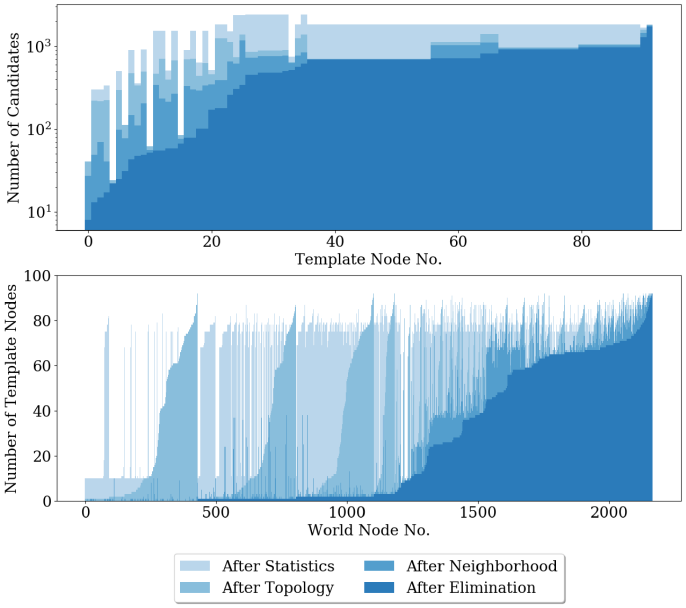


Fig. 18: (Top): The number of candidates for each template node after different levels of filtering are applied to IvySys Version 7. (Bottom): The number of template nodes for which each world node is a candidate.

4.4.5 *IvySys Version 11*

IvySys Version 11 contains one instance with five channels. It is somewhat similar to that of IvySys Version 7, as illustrated in Figure 19. Even after the application of all of our existing filters, we can only identify exact matches for 13 out of the 103 template nodes. The majority of the template nodes still have a considerable number of candidates. A closer examination of Figure 19 shows the existence of permutable sets of template nodes. This suggests the possibility that we might have already reduced the graph to the point where almost all of the remaining candidates serve as part of a valid signal.

5 Conclusion

We have introduced a series of effective filtering methods that can be used to solve subgraph isomorphism problems within a large-scale multiplex network. The filtering approaches reduce the search space using node labels/attributes, node-level statistics, topology, and alldifferent constraints. When applied iteratively until convergence, our filtering methods can significantly narrow down the search space. We also implement a more computationally

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSE.2021.3056329, IEEE Transactions on Network Science and Engineering

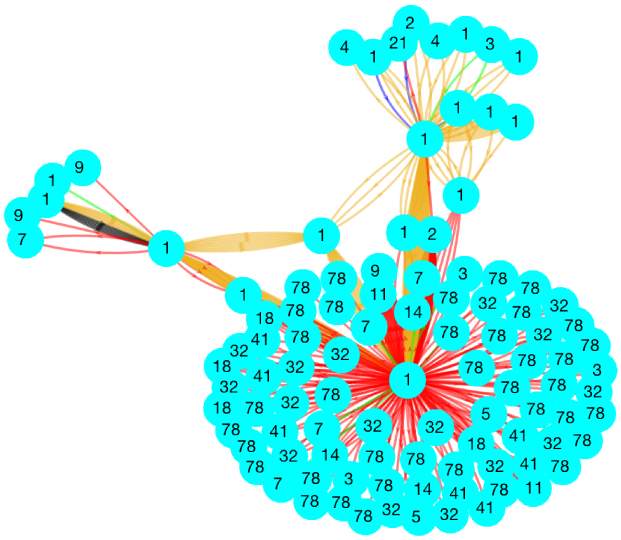


Fig. 19: Candidate count for each node in the IvySys Version 11 template after applying the node-level statistics, topology, repeated-sets, and elimination filters. The colors of the edges correspond to their different channels.

expensive isomorphism counting approach, which can be applied after filtering to solve the Subgraph Isomorphism Counting Problem (Definition 1.5). We can also proceed with a validation step that checks whether each world node participates in at least one subgraph isomorphism, hence solving the Signal Node Set Problem (Definition 1.6).

We have applied our methods to multiple types of networks: Sudoku puzzles, three real-world datasets, and the synthetic datasets created by PNNL, GORDIAN and IvySys as part of the DARPA MAA program. In all of these experiments, our methods have greatly narrowed down the candidates of the template nodes and provided us with an understanding of the size of the solution space. When applied to the Higgs Twitter network, the commercial airline network, PNNL Version 6, PNNL Real World and GORDIAN Version 7, our methods are capable of solving most, if not all, of the proposed subgraph isomorphism problems. However, even in more difficult settings such as the Great Britain Transportation and IvySys Versions 7 and 11, our methods reduce the search space to a much smaller scale. Our results show that further narrowing down the candidate lists of template nodes is often hindered by an abundance of subgraph isomorphisms existing in the world graphs. Under such circumstances, involving more node attributes such as geospatial coordinates or timestamps as part of the filtering pipeline can assist in making further progress.

6 Future Work

It would also be interesting to explore when each filter is sufficient to solve a problem, and on what types of graphs are various filters required. Different types of filters will perform differently depending on the template and world graphs, and characterizing these graphs would be useful for determining whether a filter should be run or omitted for time. In a similar vein, exploring generally which types of template and world graphs are easier or harder to solve would also be useful.

Templates with a large amount of symmetry, as in IvySys Version 11 (Section 4.4.5), can lead to a huge number of subgraph isomorphisms (SIs) that differ very little. However, these SIs can often be summarized using the concept of structural equivalence to group similar SIs together [46]. In the future, we aim to incorporate the concept of equivalence into our algorithm to improve performance, as well as provide additional insight into the structure of the solution space.

Another performance improvement would be to incorporate parallelization into our code. Counting can be easily parallelized with respect to different choices of candidates for the same node, and many of the filters can be parallelized as well with respect to individual nodes/edges. Though our code does not currently take advantage of this, we hope in the future to implement more efficient parallel versions of our code.

Another possible direction is the noisy or inexact case of the subgraph matching problem. From the application standpoint, one could consider datasets with missing edges or nodes from the template or world graphs, while the goal is to find the closest possible matches to the template. There are various possibilities for the noisy case—missing edges, missing nodes, extra edges, extra nodes. Approaches that deal with the noisy case frequently relax the original discrete matching problem to a continuous problem, allowing for the use of methods based on gradient descent or convex optimization [18], [34], [37], [60], [70].

Additionally, the use of more edge attributes will also help narrow down the solution of the subgraph matching problem. We have considered approximated location information in the Great Britain Transportation. In the case of social media, email traffic, or financial transations, timestamps can also serve as additional attributes. This type of information has been considered in [38].

Acknowledgments

This work was supported by DARPA award FA8750-18-2-0066. Moorman was supported by NSF grant DGE-1829071. We thank Jamie Atlas, Omri Azencot, Zachary Boyd, Jeremy Budd, Yoni Dukler, Matthew Jacobs, Blaine Keetch, Hao Li, Kevin Miller, Mason A. Porter, Bao Wang, Yotam Yaniv, and Baichuan Yuan for helpful discussions. We thank the anonymous reviewers for their valuable suggestions.

References

[1] N. K. Ahmed, J. Neville, R. A. Rossi, and N. Duffield. Efficient graphlet counting for large networks. In *2015 IEEE International Conference on Data Mining*, pages 1–10, Nov 2015.

[2] N. Alon, P. Dao, I. Hajirasouliha, F. Hormozdiari, and S. C. Sahinalp. Biomolecular network motif counting and discovery by color coding. *Bioinformatics*, 24(13):i241–i249, 07 2008.

[3] K. O. Babalola, O. B. Jennings, E. Urdiales, and J. A. DeBardelaben. Statistical methods for generating synthetic email data sets. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 3986–3990, Dec 2018.

[4] R. Bar-Yehuda and S. Even. A local-ratio theorem for approximating the weighted vertex cover problem. In *Analysis and Des. of Algorithms for Combinatorial Problems*, pages 27–45. Elsevier, 1985.

[5] B. Bentley, R. Branicky, C. L Barnes, Y. Chew, E. Yemini, E. T Bullmore, P. E Vértes, and W. R Schafer. The multilayer connectome of caenorhabditis elegans. *PLoS Computational Biology*, 12(12):e1005283, 2016.

[6] F. Bi, L. Chang, X. Lin, L. Qin, and W. Zhang. Efficient subgraph matching by postponing cartesian products. In *Proceedings of the 2016 International Conference on Management of Data*, pages 1199–1214. ACM, 2016.

[7] V. Bonnici, R. Giugno, A. Pulvirenti, D. Shasha, and A. Ferro. A subgraph isomorphism algorithm and its application to biochemical data. *BMC bioinformatics*, 14(7):S13, 2013.

[8] A. Cardillo, J. Gómez-Gardeñes, M. Zanin, M. Romance, D. Papo, F. del Pozo, and S. Boccaletti. Emergence of network features from multiplexity. *Scientific Reports*, 3:1344, 2013.

[9] V. Carletti, P. Foggia, A. Saggese, and M. Vento. Introducing VF3: A new algorithm for subgraph isomorphism. *Graph-Based Representations in Pattern Recognition*, pages 128–139, 2017.

[10] V. Carletti, P. Foggia, A. Saggese, and M. Vento. Challenging the time complexity of exact subgraph isomorphism for huge and dense graphs with vf3. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(4):804–818, April 2018.

[11] V. Carletti, P. Foggia, and M. Vento. Vf2 plus: An improved version of vf2 for biological graphs. In *International Workshop on Graph-Based Representations in Pattern Recognition*, pages 168–177. Springer, 2015.

[12] F. Celli, F. M. L. Di Lascio, M. Magnani, B. Pacelli, and L. Rossi. Social Network Data and Practices: the case of Friendfeed. In *International Conference on Social Computing, Behavioral Modeling and Prediction*, Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2010.

[13] D. Conte, P. Foggia, C. Sansone, and M. Vento. Graph matching applications in pattern recognition and image processing. In *Proceedings 2003 International Conference on Image Processing (Cat. No.03CH37429)*, volume 2, pages II–21, Sep. 2003.

[14] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento. A (sub)graph isomorphism algorithm for matching large graphs. *IEEE Trans. on Pattern Analysis and Machine Intell.*, 26(10):1367–1372, Oct 2004.

[15] 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.

[16] M. De Domenico, A. Lima, P. Mougel, and M. Musolesi. The anatomy of a scientific rumor. *Scientific Reports*, 3:2980, 2013.

[17] J. Douglas, B. Zimmerman, J. Xu A. Kopylov, D. Sussman, and V. Lyzinski. Metrics for evaluating network alignment. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, WSDM '18, New York, NY, USA, 2018. ACM.

[18] A. Egozi, Y. Keller, and H. Guterman. A probabilistic approach to spectral graph matching. *IEEE Trans. Pattern Anal. Mach. Intell.*, 35(1):1827, January 2013.

[19] Wenfei Fan. Graph pattern matching revised for social network analysis. In *Proceedings of the 15th International Conference on Database Theory*, ICDT '12, page 821, New York, NY, USA, 2012. Association for Computing Machinery.

[20] R. Gallotti and M. Barthelemy. The multilayer temporal network of public transport in Great Britain. *Scientific data*, 2:140056, 2015.

[21] R. Gallotti and M. Barthelemy. The multilayer temporal network of public transport in Great Britain. https://datadryad.org/resource/doi:10.5061/dryad.pc8m3, 2015.

[22] M. R. Garey and D. S. Johnson. *Computers and intractability*, volume 174. freeman San Francisco, 1979.

[23] I. P. Gent, I. Miguel, and P. Nightingale. Generalised arc consistency for the AllDifferent constraint: An empirical survey. *Artificial Intelligence*, 172(18):1973–2000, December 2008.

[24] W. Han, J. Lee, and J. Lee. Turbo iso: towards ultrafast and robust subgraph isomorphism search in large graph databases. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, pages 337–348. ACM, 2013.

[25] H. He and A. 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.

[26] X. He. Interactive template map. https://hexie1995.github.io/transportation-on-the-map/global, 2019.

[27] J. E. Hopcroft and R. M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM J. Comput.*, 2:225–231, 1973.

[28] J. Huan, W. Wang, and J. Prins. Efficient mining of frequent subgraphs in the presence of isomorphism. In *Third IEEE International Conference on Data Mining*, pages 549–552. IEEE, 2003.

[29] V. Ingalalli, D. Ienco, and P. Poncelet. Sumgra: Querying multigraphs via efficient indexing. In *International Conference on Database and Expert Systems Applications*, pages 387–401. Springer, 2016.

[30] A. Jüttner and P. Madarasi. Vf2++an improved subgraph isomorphism algorithm. *Discrete Applied Mathematics*, 242:69–81, 2018.

[31] K. Karra, S. Swarup, and J. Graham. An empirical assessment of the complexity and realism of synthetic social contact networks*. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 3959–3967, Dec 2018.

[32] M. Kivelä, A. Arenas, M. Barthelemy, J. P Gleeson, Y. Moreno, and M. A Porter. Multilayer networks. *J. of Complex Networks*, 2(3):203–271, 2014.

[33] D. E. Knuth. Dancing links. *Millennial Perspectives in Computer Science*, pages 187–214, 2000.

[34] A. Kopylov and J. Xu. Filtering strategies for inexact subgraph matching on noisy multiplex networks. In *2019 IEEE International Conference on Big Data (Big Data)*, pages 4906–4912, 2019.

[35] L. Kotthoff, C. McCreesh, and C. Solnon. Portfolios of subgraph isomorphism algorithms. In *International Conference on Learning and Intelligent Optimization*, pages 107–122. Springer, 2016.

[36] J. Larrosa and G. Valiente. Constraint satisfaction algorithms for graph pattern matching. *Mathematical Structures in Computer Science*, 12(4):403422, 2002.

[37] V. Lyzinski, D. E. Fishkind, M. Fiori, J. T. Vogelstein, C. E. Priebe, and G. Sapiro. Graph matching: Relax at your own risk. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(1):60–73, Jan 2016.

[38] P. Mackey, K. Porterfield, E. Fitzhenry, S. Choudhury, and G. Chin. A chronological edge-driven approach to temporal subgraph isomorphism. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 3972–3979, Dec 2018.

[39] M. Magnani and L. Rossi. The ML-Model for Multi-layer Social Networks. In *ASONAM*, pages 5–12. IEEE Computer Society, 2011.

[40] C. McCreesh and P. Prosser. A parallel, backjumping subgraph isomorphism algorithm using supplemental graphs. In Gilles Pesant, editor, *Int. Conf. on Principles and Practice of Constraint Programming*, pages 295–312. Springer Int. Publishing, 2015.

[41] C. McCreesh, P. Prosser, C. Solnon, and J. Trimble. When subgraph isomorphism is really hard, and why this matters for graph databases. *Journal of Artificial Intelligence Research*, 61:723–759, 2018.

[42] J. J. McGregor. Relational consistency algorithms and their application in finding subgraph and graph isomorphisms. *Information Sciences*, 19(3):229–250, 1979.

[43] Giovanni Micale, Vincenzo Bonnici, Alfredo Ferro, Dennis Shasha, Rosalba Giugno, and Alfredo Pulvirenti. Multiri: Fast subgraph matching in labeled multigraphs. *arXiv preprint arXiv:2003.11546*, 2020.

[44] Jacob D. Moorman, Thomas K. Tu, Qinyi Chen, Dominic Yang, and Xie He. jdmoorman/uclasm: v0.2.0. Zenodo. https://doi.org/10.5281/zenodo.4052353, September 2020.

[45] C. Nabti and H. Seba. Compact neighborhood index for subgraph queries in massive graphs. *arXiv preprint arXiv:1703.05547*, 2017.

[46] Thien Nguyen, Dominic Yang, Yurun Ge, Hao Li, and Andrea L Bertozzi. Applications of structural equivalence to subgraph isomorphism on multichannel multigraphs. In *2019 IEEE International Conference on Big Data (Big Data)*, pages 4913–4920. IEEE, 2019.

[47] P. Norvig. http://www.norvig.com/sudoku.html, 2010.

[48] P. Norvig. https://github.com/norvig/pytudes, 2017.

[49] S. Pilosof, M. A Porter, M. Pascual, and S. Kéfi. The multilayer nature of ecological networks. *Nature Ecology & Evolution*, 1(4):0101, 2017.

[50] Project Euler problem 96: Su Doku. https://projecteuler.net/problem=96, 2005.

[51] J.-C. Régin. A filtering algorithm for constraints of difference in CSPs. In *Proc. of the 12th Nat. Conf. on Artificial Intell, Seattle, WA, USA, July 31 - August 4, 1994, Volume 1.*, pages 362–367, 1994.

[52] X. Ren and J. Wang. Exploiting vertex relationships in speeding up subgraph isomorphism over large graphs. *Proceedings of the VLDB Endowment*, 8(5):617–628, 2015.

[53] T. Reza, C. Klymko, M. Ripeanu, G. Sanders, and R. Pearce. Towards practical and robust labeled pattern matching in trillion-edge graphs. In *2017 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 1–12, Sep. 2017.

[54] T. Schank and D. Wagner. Finding, counting and listing all triangles in large graphs, an experimental study. In S. E. Nikoletseas, editor, *Experimental and Efficient Algorithms*, pages 606–609, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[55] H. Shang, Y. Zhang, X. Lin, and J. X. Yu. Taming verification hardness: an efficient algorithm for testing subgraph isomorphism. *Proceedings of the VLDB Endowment*, 1(1):364–375, 2008.

[56] H. Simonis. Sudoku as a constraint problem. In *Proceedings of the 4th International Workshop on Modelling and Reformulating Constraint Satisfaction Problems.*, pages 13–27. Citeseer, 2005.

[57] C. Solnon. AllDifferent-based Filtering for Subgraph Isomorphism. *Artificial Intell.*, 174:850–864, August 2010.

[58] S. Sun and X. Luo. Scaling up subgraph query processing with efficient subgraph matching. *Proceedings of the 2019 IEEE 35th International Conference on Data Engineering*, 2019.

[59] Z. Sun, H. Wang, H. Wang, B. Shao, and J. Li. Efficient subgraph matching on billion node graphs. *Proc. VLDB Endow.*, 5(9):788799, May 2012.

[60] D. Sussman, Y. Park, C. E. Priebe, and V. Lyzinski. Matched filters for noisy induced subgraph detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1–1, 2019.

[61] J. R. Ullmann. An algorithm for subgraph isomorphism. *J. ACM*, 23(1):31–42, January 1976.

[62] J. R Ullmann. Bit-vector algorithms for binary constraint satisfaction and subgraph isomorphism. *Journal of Experimental Algorithmics (JEA)*, 15:1–6, 2010.

[63] United Kingdom's national public transport data repository. https://data.gov.uk/dataset/d1f9e79f-d9db-44d0-b7b1-41c216fe5df6/national-public-transport-data-repository-nptdr, 2015.

[64] W.-J. van Hoeve. The alldifferent constraint: A survey. *CoRR*, cs.PL/0105015, 05 2001.

[65] L. M. Verbrugge. Multiplexity in adult friendships. *Social Forces*, 57:1286–1309, 1979.

[66] J. Xu, H. Tong, T. Lu, J. He, and N. Bliss. GTA3 2018: Workshop on graph techniques for adversarial activity analytics. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, WSDM '18, pages 803–803, New York, NY, USA, 2018. ACM.

[67] X. Yan and J. Han. gspan: Graph-based substructure pattern mining. In *2002 IEEE International Conference on Data Mining, 2002. Proceedings.*, pages 721–724. IEEE, 2002.

[68] Lyudmila Yartseva and Matthias Grossglauser. On the performance of percolation graph matching. COSN '13, page 119130, New York, NY, USA, 2013. Association for Computing Machinery.

[69] S. Zampelli, Y. Deville, and C. Solnon. Solving subgraph isomorphism problems with constraint programming. *Constraints*, 15:327–353, 07 2010.

[70] M. Zaslavskiy, F. Bach, and J. Vert. A path following algorithm for the graph matching problem. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(12):2227–2242, Dec 2009.

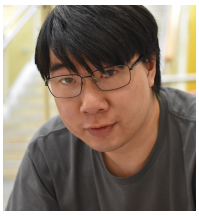
[71] S. Zhang, S. Li, and J. Yang. Gaddi: distance index based subgraph matching in biological networks. In *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology*, pages 192–203. ACM, 2009.

[72] P. Zhao and J. Han. On graph query optimization in large networks. *Proceedings of the VLDB Endowment*, 3(1-2):340–351, 2010.

[73] M. Zitnik and J. Leskovec. Predicting multicellular function through multi-layer tissue networks. *Bioinformatics*, 33(14):i190–i198, 2017.

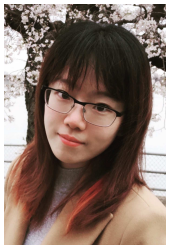
[74] L. Zou, J. Mo, L. Chen, M. T. Özsu, and D. Zhao. gstore: answering sparql queries via subgraph matching. *Proceedings of the VLDB Endowment*, 4(8):482–493, 2011.

Jacob D. Moorman is a Ph.D. candidate in Mathematics in the Department of Mathematics at the University of California, Los Angeles. He received B.S. degrees in Mathematics and Computer Science from the New Jersey Institute of Technology in 2016. His research interests include network analysis, linear algebra, randomized optimization algorithms, pattern recognition, and high-dimensional data analysis.



Thomas K. Tu is currently working towards a Ph.D. in Mathematics in the Department of Mathematics at the University of California, Los Angeles. He received a B.S. degree in Mathematics and Computer Science from the New Jersey Institute of Technology in 2016. His research interests include knowledge graphs, network science, randomized optimization algorithms, machine learning, and image analysis.

Qinyi Chen is a first-year Ph.D. student at the Operations Research Center at MIT. She received a B.S. degree in Applied Mathematics and a specialization in computing from the University of California, Los Angeles in 2020. Her research interests include network science, as well as online learning and optimization.



Xie He is a Ph.D. student at the University of North Carolina, Chapel-Hill. She received her bachelor's degree in Applied Mathematics from the University of California, Los Angeles in 2019. Her research interest include network analysis, community detection and scientific computing.

Andrea L. Bertozzi, *Member, IEEE* completed all her degrees in mathematics at Princeton. She is an Applied Mathematician with expertise in nonlinear partial differential equations and graphical models for machine learning. She was an L. E. Dickson Instructor and an NSF Postdoctoral Fellow at the University of Chicago from 1991 to 1995. She was the Maria Geoppert-Mayer Distinguished Scholar with the Argonne National Laboratory from 1995 to 1996. She was on the faculty at Duke University from 1995 to 2004, first as an Associate Professor of mathematics and then as a Professor of mathematics and physics. She has been on the faculty at UCLA since 2003 where she now holds the position of Distinguished Professor of Mathematics and Mechanical and Aerospace Engineering, along with the Betsy Wood Knapp Chair for Innovation and Creativity. She was elected to the American Academy of Arts and Sciences in 2010 and to the Fellows of the Society of Industrial and Applied Mathematics (SIAM) in 2010. She became a Fellow of the American Mathematical Society in 2013 and the American Physical Society in 2016. In 2018, she was elected to the U.S. National Academy of Sciences. Her honors include the Sloan Research Fellowship in 1995, the Presidential Early Career Award for Scientists and Engineers in 1996, and the SIAM Kovalevsky Prize in 2009, and the SIAM Kleinman Prize in 2019. She received the SIAM Outstanding Paper Prize in 2014 with A. Flenner, for her work on geometric graphbased algorithms for machine learning. She received the Simons Math + X Investigator Award in 2017. She is a Thomson-Reuters/Clarivate Analytics Highly Cited Researcher in mathematics in 2015 and 2016.