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) are increasingly useful data structures for representing entities and their interactions in disciplines such as bioinformatics, ecological networks, and neural networks. 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. Subgraph matching commonly used in bioinformatics, social network analysis, and other applications. It is also an important subroutine in frequent subgraph mining and graph database search. Despite the abundance of multiplex network data in these applications, there are relatively few subgraph matching algorithms that expressly support multiplex networks compared to the number of algorithms that support single-channel networks. 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. However, in general there can be a vasty large number of signals, in which case listing or even counting them can be computationally intractable. In such situations, it is appealing to have methods that characterize the space of all signals in some way. For example, one might identify the world nodes that participate in at least one signal. Alternatively, one may seek the set of world nodes that correspond to a particular template node in at least one signal. We find that these problems can be feasible, even when it is prohibitive to list or count all of the signals.

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

In the remainder of this section, we 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 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 \( G_t = (V_t, E_t, L_t, C) \) and a world \( G_w = (V_w, E_w, L_w, C) \), we explore the space of all subgraphs of the world that match the template. There are several closely related problems with different computational costs. Each of these problems relies on the same concept of subgraph isomorphism (SI) as described below.

Definition 1.2 (SI: Subgraph Isomorphism). An injective function \( f : V_t \rightarrow V_w \) is called a subgraph isomorphism (SI)
from \( G_t = (V_t, \mathcal{E}_t, \mathcal{L}_t, C) \) to \( G_w = (V_w, \mathcal{E}_w, \mathcal{L}_w, C) \) if

\[
\mathcal{L}_t(v) = \mathcal{L}_w(f(v)) \quad \forall v \in V_t
\]

\[
\mathcal{E}_t(u, v) \leq \mathcal{E}_w(f(u), f(v)) \quad \forall u, v \in V_t \times V_t.
\]

The set of all SIs from \( G_t \) to \( G_w \) is denoted \( \mathcal{F}(G_t, G_w) \).

In short, an injective function \( f \) mapping template nodes to world nodes is a SI if each node \( v \) in the template has the same label as the corresponding node \( f(v) \) in the world, and for each pair of nodes \( (u, v) \) in the template, the corresponding pair of nodes \( (f(u), f(v)) \) in the world have at least as many edges in each channel and direction as \( u \) and \( v \) have between them. This is sometimes referred to as a subgraph monomorphism in the literature. We use the term induced SI when there are 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.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIP</td>
<td>Check if there are any SIs.</td>
</tr>
<tr>
<td>SNSP</td>
<td>Find all the world nodes involved in SIs.</td>
</tr>
<tr>
<td>MCSP</td>
<td>Find all pairs ((u, v)) where ( u = f(v) ) for some SI ( f ).</td>
</tr>
<tr>
<td>SICP</td>
<td>Count the number of SIs.</td>
</tr>
<tr>
<td>SMP</td>
<td>Find all SIs from the template ( G_t ) to the world ( G_w ).</td>
</tr>
</tbody>
</table>

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 5 and 7 than there are between B and C.

![Figure 1](image)

Fig. 1: In the above networks, the shapes of the nodes correspond 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 \{1, 2, 6\}, \{1, 6, 8\}, \{4, 5, 7\}, and \{7, 9, 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 B and C from among those between world nodes 5 and 7.

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

\[
\text{check if } \mathcal{F}(G_t, G_w) = \emptyset. \quad \text{(SIP)}
\]

Typically the SIP is solved by exhaustively searching for any SI \( f \in \mathcal{F}(G_t, G_w) \). If none can be found, then \( \mathcal{F}(G_t, G_w) = \emptyset \). 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 \( G_t = (V_t, \mathcal{E}_t, \mathcal{L}_t, C) \) and a world \( G_w = (V_w, \mathcal{E}_w, \mathcal{L}_w, C) \), find all SIs from the template \( G_t \) to the world \( G_w \). That is,

\[
\text{find all } f \in \mathcal{F}(G_t, G_w). \quad \text{(SMP)}
\]

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

<table>
<thead>
<tr>
<th>Template node ( v )</th>
<th>( f_1(v) )</th>
<th>( f_2(v) )</th>
<th>( f_3(v) )</th>
<th>( f_4(v) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>4</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>5</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>C</td>
<td>6</td>
<td>7</td>
<td>6</td>
<td>10</td>
</tr>
</tbody>
</table>

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

Since any algorithm for solving the SMP must list out all of the SIs, the computation time required, at minimum, must scale with the number of SIs \( |\mathcal{F}(G_t, G_w)| \). The problems described in the remainder of this section can be used to explore the space of SIs, especially when solving the SMP is computationally prohibitive.

In certain contexts, one may wish to only count the number of SIs from the template to the world. Examples of such contexts include summary statistics 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 \( G_t = (V_t, \mathcal{E}_t, \mathcal{L}_t, C) \) and a world \( G_w = (V_w, \mathcal{E}_w, \mathcal{L}_w, C) \), determine the number of SIs from the template \( G_t \) to the world \( G_w \). That is,

\[
\text{find } |\mathcal{F}(G_t, G_w)|. \quad \text{(SICP)}
\]

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 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 \( G_t = (V_t, \mathcal{E}_t, \mathcal{L}_t, C) \) and a world \( G_w = \)
Given a template \( G \) the template. A Cartesian product is then performed on query. To accelerate searching, they construct indexes for database of graphs all graphs that match a given subgraph

\[ \text{find } \bigcup_{f \in F(G, w)} f(V_t). \]  

(SNSP)

A compromise between the compactness of the SNSP and the descriptive power of the SMP is to find the minimal candidate sets. For each template node, the minimal candidate set is 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 \( G_t = (V_t, E_t, L_t, C) \) and a world \( G_w = (V_w, E_w, L_w, C) \), for each template node \( v \in V_t \) find all world nodes which correspond to \( v \) in at least one signal. That is,

\[ \text{find } \bigcup_{f \in F(G_t, G_w)} \{ f(v) \} \text{ for each } v \in V_t. \]  

(MCSP)

A naive algorithm for solving the MCSP is to solve the SIP with the added constraint \( u = f(v) \) for each \( (u, v) \in V_w \times V_t \). Thus, solving the MCSP is at most \(|V_t||V_w|\) 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: 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 the world nodes that are possible matches for each template node. By repeatedly applying local constraints, this matrix is reduced until only a few possible matches remain. The matrix can then be explored to find all solutions. Examples of constraint propagation approaches include McGregor [42], nRF+ [36], ILF [69], LAD [37] (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], QuickSL [53], TurboISO [24], BoostISO [52], CFL-Match [6], and CNI-Match [43]. Our problem does not involve graph databases and we thus do not construct indices. However, several graph indexing 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 each template node \( v \), it creates a list of candidate nodes in the world that could correspond to \( v \). Initially, this is simply all nodes with degree greater than or equal to the degree of \( v \). At each step in the tree search, candidate node \( u \) is chosen as a match for template node \( v \). 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. It distinguishes matched nodes (nodes with only one candidate) from other nodes, and ensures 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. It also enforces 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 look-ahead algorithm, with specific look ahead for subgraph isomorphism. They enforce the constraint that if \( v_w \) is a candidate for template node \( v_t \), then the number of \( v_t \)'s neighbors must be less than or equal to the number of candidates for those neighbors that are adjacent to \( v_w \).

The work [37] classifies many constraints previously used in different subgraph isomorphism algorithms. 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 all-different 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. It constructs an NEC tree, whose vertices represent groups of permutable template nodes, and perform matching using this tree. At the end, it combines and permutes the candidates for each of these template nodes. For the matching order, it divides the graph into candidate subregions, each of which contains a group of candidates that may take part in a signal. It then prioritizes 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 [3], the template is first decomposed into three types of structures: Core, Forest, and Leaf. To do this, it first constructs a spanning tree of the template. Then, it computes the minimal connected subgraph containing all non-tree edges of the template. It then iteratively removes 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) [45]. 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 few SIs overall. In Section 4, we show examples where 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 \( v_i \in \mathcal{V}_t \), we keep track of its candidates \( D(v_i) \subseteq \mathcal{V}_w \). Initially, we treat every world node as a candidate for every template node. Each filter enforces different sets of constraints to eliminate candidates that 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 \( G_t \), world \( G_w \), candidate set \( D(v_i) \) for each \( v_i \in \mathcal{V}_t \), list of filters filters
2: converged ← False
3: while converged is False
4: converged ← True >> Stop unless progress is made
5: for filter ∈ filters
6: \( D \leftarrow \text{filter}(G_t, G_w, D) \) >> Apply the filter
7: if \( |D(v_i)| \) decreased for some \( v_i \in \mathcal{V}_t \)
8: converged ← False >> Progress was made
9: Break >> Restart from the first filter
10: Output updated candidate set \( D(v_i) \) for each \( v_i \in \mathcal{V}_t \)

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.

2.1 Node Label Filter

In order for a world node \( v_w \) to be a candidate for a template node \( v_t \), the label \( L_w(v_w) \) must match the label \( L_t(v_t) \). 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 depends 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 \( v_w \) to be a candidate for a template node \( v_t \), certain statistical properties of \( v_w \) should not be less than those of \( v_t \). 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 is used in each channel in the networks.

Algorithm 2: Node-level Statistics Filter

1. **Input** template \( G_t \), world \( G_w \), candidate set \( D(v_i) \) for each \( v_i \in V_t \)
2. **for** statistic \( \in \) [in-degree, out-degree, ... ]
3. **for** template or world node \( v \in V_t \cup V_w \)
4. **Compute** statistic \( (v) \)
5. **for** template node \( v_t \in V_t \)
6. **for** candidate \( v_w \in D(v_t) \)
7. **if** statistic \( (v_w) < \) statistic \( (v_t) \)
8. **Remove** \( v_w \) from \( D(v_t) \)
9. **Output** updated candidate set \( D(v_t) \) for each \( v_t \in V_t \)

Other more complex statistics can be used channel-wise, such as number of triangles, number of nodes within \( k \) steps, and number of paths of length \( \ell \). Statistics combining information from multiple channels can also be used, such as the number of in-neighbors in channel \( a \) that are also out-neighbors in channel \( b \). We use only simple statistics so that the cost of computing them scales only linearly in the number of distinguishable edges (edges whose source, destination, direction, and channel are distinct).

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 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 Table 4.

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

<table>
<thead>
<tr>
<th>Template</th>
<th>World</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>2</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>4</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Candidates for</th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>A candidates</td>
<td>1, 3, 4</td>
<td>1, 2, 3, 4</td>
<td>4, 4</td>
</tr>
<tr>
<td>B candidates</td>
<td>1, 2, 3, 4</td>
<td>1, 2</td>
<td>2</td>
</tr>
<tr>
<td>C candidates</td>
<td>4, 4</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

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 help 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 \( v_w \) is a candidate for template node \( v_t \), then for every template node \( u_t \) neighboring \( v_t \), there must exist a candidate \( u_w \) for \( u_t \) that neighbors \( v_w \). The natural extension to multiplex networks is that if \( v_w \) is a candidate for template node \( v_t \), then for every template node \( u_t \) neighboring \( v_t \), there must exist a candidate \( u_w \) for \( u_t \) that has as many edges in each channel and direction between \( v_w \) and \( u_w \) as there are between \( v_t \) and \( u_t \).
Algorithm 3 Topology Filter

1. **Input** template $G_t$, world $G_w$, candidate set $D(v_t)$ for each $v_t \in V_t$
2. **for** neighboring template nodes $v_t$ and $u_t$
3. **for** candidate $v_w \in D(v_t)$
4. $nbr\_cand\_found \leftarrow$ False
5. **for** candidate $u_w \in D(u_t)$
6. $\text{enough\_out} \leftarrow E_w(v_w, u_w) \geq E_t(v_t, u_t)$
7. $\text{enough\_in} \leftarrow E_w(u_w, v_w) \geq E_t(u_t, v_t)$
8. **if** $\text{enough\_out}$ and $\text{enough\_in}$
9. $nbr\_cand\_found \leftarrow$ True
10. **break**
11. **if** $nbr\_cand\_found$ is not True
12. Remove $v_w$ from $D(v_t)$
13. **Output** updated candidate set $D(v_t)$ for each $v_t \in V_t$

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 $O(|E_t|^2 |E_w|^2)$.

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 $v_w$ to be a candidate for template node $v_t$, there must exist some injective mapping from the template nodes to their candidates under which $v_t$ is mapped to $v_w$. To enforce GAC(AllDiff), we identify sets of template nodes $T \subseteq V_t$ where the union of their candidates $\bigcup_{v_t \in T} D(v_t)$ has the same cardinality as $T$. 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 $O\left(\sqrt{|V_t| + |V_w| \sum_{v_t \in V_t} |D(v_t)|}\right)$ 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 that 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 $O(|V_w|^{3/2})$, whereas Algorithm 4 scales linearly with $|V_w|$.

Algorithm 4 Repeated-Sets Filter

1. **Input** template $G_t$, world $G_w$, candidate set $D(v_t)$ for each $v_t \in V_t$
2. $\text{unions} \leftarrow \text{EmptyMap}()$
3. **for** $(u_t, w_t) \in \text{unions}$
4. $D(u_t) \leftarrow D(u_t) \cup D(v_t)$
5. **if** $|D(u_t)| < |V_t|$
6. $u_t \leftarrow u_t \cup \{v_t\}$
7. **else**
8. $T \leftarrow T \cup u_t$
9. $D(u_t) \leftarrow D(u_t) \setminus u_t$ for $u_t \in V_t \setminus T$
10. $\text{unions}[u_t] \leftarrow \text{unions}[u_t]$
11. **if** $|D(u_t)| = 1$
12. $\{v_t\}$ is a tight set.
13. $T \leftarrow T \cup \{v_t\}$
14. $D(u_t) \leftarrow D(u_t) \setminus D(v_t)$ for $u_t \in V_t \setminus \{v_t\}$
15. **else**
16. $\text{unions}[u_t] \leftarrow \text{unions}[u_t]$
17. **if** $|D(v_t)| < |V_t|$ or $|D(u_t)| < |V_t|$
18. $\{v_t\}$ is a tight set.
19. $T \leftarrow T \cup \{v_t\}$
20. $D(u_t) \leftarrow D(u_t) \setminus D(v_t)$ for $u_t \in V_t \setminus \{v_t\}$
21. **else**
22. $\text{unions}[D(v_t)] \leftarrow \{v_t\}$
23. $\text{todo} \leftarrow \{v_t \in \text{todo} : |D(v_t)| < |V_t| \setminus T\}$
24. **Output** updated candidate set $D(v_t)$ for $v_t \in V_t$

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, $\{A\}$ is a tight set and $T$ 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 in Sections 4.4.3 and 4.4.5 that contain 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 $v_w$ to be a candidate for a template node $v_t$, there must be some injective mapping $f_v$ from the neighbors of $v_t$ to their candidates under which
the neighborhood filter is able to eliminate node \(v_w\) as a candidate for node \(v_t\). We extend this to the multiplex network context by requiring not just that \(f_t(u_t)\) neighbors of \(v_w\) but also that there are enough edges in each channel and direction

\[
\mathcal{E}_w(v_w, u_w) \geq f_t(v_t, u_t) \\
\text{and } \mathcal{E}_w(u_w, v_w) \geq f_t(u_t, v_t)
\]

where \(u_w = f_v(u_t)\). In the neighborhood filter Algorithm 5, we enforce the multiplex LAD constraint by searching for such a mapping \(f_t\). If none exists, we eliminate \(v_w\) as a candidate for \(v_t\).

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

**Algorithm 5 Neighborhood Filter**

1: Input template \(G_t\), world \(G_w\), candidate set \(D(v_t)\) for each \(v_t \in V_t\)
2: for template node \(v_t \in V_t\) do
3: for candidate \(v_w \in D(v_t)\) do
4: \(\mathcal{N}_{v_t} \leftarrow \text{Neighborhood}(v_t)\)
5: \(\mathcal{N}_{v_w} \leftarrow \text{Neighborhood}(v_w)\)
6: \(B \leftarrow \text{EmptyBipartiteGraph}(\mathcal{N}_{v_t}, \mathcal{N}_{v_w})\)
7: for \((u_t, u_w) \in \mathcal{N}_{v_t} \times \mathcal{N}_{v_w}\) do
8: \(\begin{cases} 
\{u_w \in D(u_t) \\
\text{and } \mathcal{E}_w(v_w, u_w) \geq f_t(v_t, u_t) \\
\text{and } \mathcal{E}_w(u_w, v_w) \geq f_t(u_t, v_t) \}
\end{cases}\)
9: Add an edge between \(u_t\) and \(u_w\) in \(B\)
10: \(\text{max_match} \leftarrow \text{MaxCardinalityMatching}(B)\)
11: if \(\text{max_match} < |\mathcal{N}_{v_t}|\) then
12: Remove \(v_w\) from \(D(v_t)\)
13: Output updated candidate set \(D(v_t)\) for each \(v_t \in V_t\)

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 B corresponding to \(v_t = C\) and \(v_w = 4\) is shown in Figure 3. Since the maximum cardinality matching on \(B\) has only two edges, the neighborhood filter is able to eliminate node 4 as a candidate for node C.

### 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 \((v_t, v_w)\), we do a one step lookahead. We assign \(v_w\) to \(v_t\) and iterate over all other filters until convergence. If this results in one or more template nodes having no candidates, then \(v_t\) cannot be mapped to \(v_w\) and we eliminate \(v_w\) as a candidate for \(v_t\).

**Algorithm 6 Elimination Filter**

1: Input template \(G_t\), world \(G_w\), candidate set \(D(v_t)\) for each \(v_t \in V_t\)
2: \(S \leftarrow \text{copy}(D)\) \(\triangleright\) Save the candidate sets
3: for template node \(v_t \in V_t\) do
4: for candidate \(v_w \in D(v_t)\) do
5: \(D(v_t) \leftarrow \{v_w\}\) \(\triangleright\) Assign \(v_w\) to \(v_t\)
6: Iterate all filters to convergence
7: if \(D(u_t)\) is empty for any \(u_t \in V_t\) then
8: Remove \(v_w\) from \(D(v_t)\) in \(S\)
9: Reset candidate sets to \(S\)
10: Output updated candidate set \(D(v_t)\) for each \(v_t \in V_t\)

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.

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 remains a candidate of every template node in Graph A. By including elimination filter, we are able to


### Algorithm 7 Isomorphism Counting

1. function ISO_COUNT
2. Input template $G_t$, world $G_w$, candidate set $D(v_t)$ for each $v_t \in V_t$, unspecified node cover $N_C$
3. if $N_C = \emptyset$
4. $N_{isos} \leftarrow$ CountAllDiff($D$)
5. else
6. $N_{isos} \leftarrow 0$
7. $v_t \leftarrow N_C.pop()$
8. $S \leftarrow$ copy($D$) $\triangleright$ Save the candidate sets
9. for world node $v_w \in D(v_t)$
10. $D(v_t) \leftarrow \{v_w\} \triangleright$ Assign $v_w$ to $v_t$
11. Apply filters
12. $N_{isos} \leftarrow N_{isos} +$ IsoCount($G_t, G_w, D, N_C$)
13. Reset candidate sets to $S$
14. Output number of isomorphisms $N_{isos}$

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 turns 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 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.

### Table 6: Biadjacency matrix of $B$ used in the matching problem between the neighborhoods of template node $C$ and world node 4.

<table>
<thead>
<tr>
<th>$N_C$</th>
<th>2</th>
<th>3</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

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 $[51]$.
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

```
1: Input template \( G_t \), world \( G_w \), candidate set \( D(v_t) \) for each \( v_t \in V_t \)
2: \( D'(v_t) \leftarrow \emptyset \) for each \( v_t \in V_t \) \hfill \triangleright \text{Valid candidates}
3: for template node \( v_t \in V_t \)
4: for candidate \( v_w \in D(v_t) \setminus D'(v_t) \)
5: \( S \leftarrow \text{copy}(D) \)
6: \( S(v_t) \leftarrow \{v_w\} \) \hfill \triangleright \text{Assign} \( v_w \) \text{ to} \( v_t \)
7: Attempt to find an SI \( f \) using \( S \) \hfill \triangleright \text{Solve the SIP}
8: if isomorphism found
9: \( D'(u_t) \leftarrow D'(u_t) \cup \{f(u_t)\} \) for each \( u_t \in V_t \)
10: else
11: Remove \( v_w \) from \( D(v_t) \)
12: Output minimal candidate set \( D'(v_t) \) for each \( v_t \in V_t \)
```

This algorithm can also be modified to solve the SNSP. If, instead of adding the tuple \((u_t, f(u_t))\) to \textit{validated}, only the element \( f(u_t) \) 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, and 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).

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 are 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 isomorphism 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.

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.

```
<table>
<thead>
<tr>
<th></th>
<th>9x9 time (s)</th>
<th>9x9x3 time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>easy50</td>
<td>8.33</td>
<td>11.24</td>
</tr>
<tr>
<td>top95</td>
<td>116.7</td>
<td>179.4</td>
</tr>
<tr>
<td>hardest</td>
<td>43.4</td>
<td>116.7</td>
</tr>
</tbody>
</table>
```

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.

### 4.2 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

\[
p_w = 1 - \frac{1 - \sqrt{1 - p_t + p_t^2}}{p_t}
\]

is scaled up as the number of channels increases to maintain a fixed probability of length \(|C|\) edge vectors matching

\[
\mathbb{P}(E_t(u_t, v_t) \leq E_w(u_w, v_w)) = 0.75
\]

for all template nodes \( u_t, v_t \) and world nodes \( u_w, v_w \). 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 \texttt{IsoCount}) taken by \texttt{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).

```
Mean Number of Iterations for SIP/SICP

<table>
<thead>
<tr>
<th></th>
<th>SIP 1 channel</th>
<th>SIP 2 channels</th>
<th>SIP 3 channels</th>
</tr>
</thead>
<tbody>
<tr>
<td>SICP 1 channel</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SICP 2 channels</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SICP 3 channels</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

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 matching subgraphs in the world graph. Table 7 summarizes the sizes and filtering results.

#### 4.3.1 Great Britain Transportation

The Great Britain Transportation Network [20] combines the public transportation dataset available through the United
<table>
<thead>
<tr>
<th>Dataset</th>
<th>Template Nodes</th>
<th>Edges</th>
<th>World Nodes</th>
<th>Edges</th>
<th>Channels</th>
<th>Number of isomorphisms</th>
<th>Filters</th>
<th>Problems solved</th>
</tr>
</thead>
<tbody>
<tr>
<td>Britain Transportation</td>
<td>53</td>
<td>56</td>
<td>262,377</td>
<td>475,502</td>
<td>5</td>
<td>N/A</td>
<td>S, T, R, E</td>
<td>SIP</td>
</tr>
<tr>
<td>Britain Transportation (3km)</td>
<td>53</td>
<td>56</td>
<td>262,377</td>
<td>475,502</td>
<td>5</td>
<td>$3.76 \times 10^7$</td>
<td>L, S, T, R, E</td>
<td>SIP, SNSP, MCSP, SICP</td>
</tr>
<tr>
<td>Higgs Twitter</td>
<td>115</td>
<td>2,668</td>
<td>456,626</td>
<td>5,367,315</td>
<td>4</td>
<td>$1.03 \times 10^{14}$</td>
<td>S, T, R</td>
<td>SIP, SNSP, MCSP, SICP</td>
</tr>
<tr>
<td>Commercial Airlines</td>
<td>37</td>
<td>210</td>
<td>450</td>
<td>7,177</td>
<td>37</td>
<td>$3.65 \times 10^9$</td>
<td>S, T, R</td>
<td>SIP, SNSP, MCSP, SICP</td>
</tr>
</tbody>
</table>

**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.

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].

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.

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.

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.

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 (Fig. 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 $1.03 \times 10^{14}$ 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 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 world and two templates. The worlds have around 23k nodes and over 12 million edges each. The templates have 74 to 81 nodes and 1200-1650 edges.

By filtering based on node-level statistics, topology and repeated-sets, we identify the signals for most of the 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 filters gradually reduce the number of candidates for each template node in instance B1-S1. In the bottom histogram, we observe that after node-level statistics and topology filters, around 5000 world nodes still remain candidates of at least one template node. However, the subsequent application of elimination filter reduces the number of candidate world nodes to roughly 100. 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.

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]. 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. Many of the remaining template nodes have hundreds of candidates. By additionally applying validation to solve the MCSP, we verify that all of the remaining candidates participate in at least one signal. This 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. 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
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.

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.

the number of isomorphisms to be $1.51 \times 10^{12}$, 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 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.
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 all different constraints. When applied iteratively until convergence, our filtering methods can significantly narrow down the search space. We also implement a more computationally 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 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
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 with respect to individual nodes/edges. Though our code does not currently take advantage of this, we hope to implement more efficient parallel versions in the future.

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 transactions, timestamps can also serve as additional attributes. This type of information is considered in [38].

**REFERENCES**


[77] Thomas K. Tu is currently working towards a Ph.D. in 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, stochastic optimization algorithms, machine learning, and image analysis.

[78] Qiinyi 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.

[79] 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.

[80] Andrea L. Bertozzi, Member, IEEE completed all her degrees in mathematics at Princeton. She is an Applied Mathematician with expertise in non-linear 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 SIAM Outstanding Paper 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.