UNIVERSITY OF CALIFORNIA

Los Angeles

Subgraph Matching on Attributed Multiplex Networks with Applications to Knowledge Graphs

A dissertation submitted in partial satisfaction
of the requirements for the degree
Doctor of Philosophy in Mathematics

by

Thomas K. Tu

2021

ABSTRACT OF THE DISSERTATION

Subgraph Matching on Attributed Multiplex Networks
with Applications to Knowledge Graphs

by

Thomas K. Tu
Doctor of Philosophy in Mathematics
University of California, Los Angeles, 2021
Professor Andrea L. Bertozzi, Chair

Abstract: For applications involving graph-structured data, a natural problem is to identify subgraphs that match a given template. In this thesis, we present several approaches for solving this *subgraph matching problem*, including approaches for both exact and inexact subgraph matching. Additionally, we define several problems related to subgraph matching in order to obtain a better understanding of the entire solution space for the subgraph matching problem. We analyze a variety of synthetic and real-world multiplex datasets, including Sudoku, transportation networks, Twitter, the DBPedia knowledge graph, and several datasets from the DARPA-MAA program for modeling adversarial activity. We then show how our exact subgraph matching approach can be extended to handle errors in network alignment, the process of merging multiple networks together to form a single multiplex network. Finally, we address the closely related problem of matching variable length path-based graph templates, and apply our method to identify potential activation pathways within a biological knowledge graph extracted from scientific publications on COVID-19.

The dissertation of Thomas K. Tu is approved.

University of California, Los Angeles

2021

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

First, I would like to thank my advisor, Andrea Bertozzi, for her invaluable mentorship and guidance throughout my Ph.D. career. I also want to thank Jacob Moorman, Qinyi Chen, Xie He, Dominic Yang, Yurun Ge, and the rest of my collaborators on the UCLA DARPA-MAA team for their dedication, hard work, and companionship over the past three and a half years.

Additionally, I thank the students from the UCLA 2020 CAM knowledge graphs REU for providing me with their annotated bibliography on knowledge graphs, and Jamie Atlas, Omri Azencot, Zachary Boyd, Jeremy Budd, Yonatan Dukler, Matthew Jacobs, Blaine Keetch, Hao Li, Kevin Miller, Mason A. Porter, Bao Wang, Yotam Yaniv, and Baichuan Yuan for many helpful discussions.

I thank the teams at Pacific Northwest National Laboratories, Ivysys Technologies, and the Graphing Observables from Realistic Distributions In Activity Networks (GORDIAN) team for providing us with datasets through the DARPA-MAA program. I would also like to thank my colleagues at HRL Laboratories, Systems and Technology Research, the University of Maryland DARPA-MAA team, and all of our other fellow performers for their work on the DARPA-MAA project, and the spirit of collaboration they have helped to create and foster.

Chapter 2 is adapted from [MTC21] and I acknowledge my coauthors Jacob Moorman, Qinyi Chen, Xie He, and Andrea Bertozzi. Jacob, Qinyi, and I developed the main subgraph matching algorithm and code. Qinyi and Xie conducted the experiments on real-world datasets, and Qinyi, Xie, Jacob, and I analyzed the DARPA dataset experiments. I was responsible for the Sudoku and Erdős–Rényi experiments. Andrea led the project and supervised our work.

Chapter 3 is adapted from [TMY20] and I acknowledge my coauthors Jacob Moorman, Dominic Yang, Qinyi Chen, and Andrea Bertozzi. Jacob, Qinyi, and I developed the inexact subgraph matching algorithm and code. I was in charge of the AIDA V2.1.2

experiments and provided the resulting solution space to Dominic, who performed the analysis in Subsection 3.3.1.1. Andrea led the project and supervised our work. Subsection 3.2.3 did not appear in [TMY20] and was written by Jacob. It was published as a part of Jacob's thesis dissertation [Moo21]. Subsection 3.3.2 also did not appear in [TMY20], and was done by me. I also thank the PNNL team for providing a definition and Python implementation of the graph edit distance metric, without which much of the work would not have been possible.

Chapter 4 is adapted from [TY20] and I acknowledge my coauthor Dominic Yang, who contributed the visualization and analysis of the solution space found in Section 4.4. I was responsible for the rest of the work, including the code and algorithms as well as the experiments on the AIDA V2.0.2 dataset.

Chapter 5 has not yet appeared in a publication and is my own work. I thank Jeremy Zucker and Jeremy Teuton from PNNL for providing me (via email) a technical description of the Indra-Blender COVID-19 knowledge graph dataset, which was used in the writing of Section 5.1.

Finally, I wish to thank the Defense Advanced Research Projects Agency for providing me with financial support during my Ph.D (agreement number FA8750-18-2-0066).

The material in this thesis is based on research sponsored by the Air Force Research Laboratory and DARPA under agreement number FA8750-18-2-0066. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Air Force Research Laboratory and DARPA or the U.S. Government.

VITA

2015	Software Development Intern, Trillium Labs LLC.
2016	B.S. (Applied Mathematics and Computer Science), NJIT.
2017	Data Science Consultant, NovaSignal (formerly Neural Analytics).
2016–2018	Teaching Assistant, Mathematics Department, UCLA.
2018	M.A. (Mathematics), UCLA.
2019	Research Intern, HRL Laboratories LLC.
2018–Present	Graduate Student Researcher, Mathematics Department, UCLA.

PUBLICATIONS

J. D. Moorman, Q. Chen, T. K. Tu, X. He, A. L. Bertozzi, "Subgraph Matching on Multiplex Networks." *IEEE Transactions on Network Science and Engineering*, vol. 8, no. 2, pp. 1367–1384. IEEE, 2021.

J. D. Moorman, Q. Chen, T. K. Tu, Z. M. Boyd, A. L. Bertozzi, "Filtering Methods for Subgraph Matching on Multiplex Networks." In *2018 IEEE International Conference on Big Data*, pp. 3980–3985. IEEE, 2018.

T. K. Tu, J. D. Moorman, D. Yang, Q. Chen, A. L. Bertozzi, "Inexact Attributed Subgraph

Matching." In *2020 IEEE International Conference on Big Data*, pp. 2575–2582. IEEE, 2020.

T. K. Tu, D. Yang, "Fault-tolerant Subgraph Matching on Aligned Networks." In *2020 IEEE International Conference on Big Data*, pp. 2569–2574. IEEE, 2020.

CHAPTER 1

Introduction

1.1 Applications and Problem Definitions

Multiplex networks [KAB14] are increasingly useful data structures for representing entities and their interactions in disciplines such as bioinformatics [ZL17], social networks [Ver79], ecological networks [PPP17], and neural networks [BBB16]. Within these, there often arises the need to identify patterns in the data matching a user-defined graph query. This task can be considered as *subgraph matching*, the process of determining whether a given pattern called a *template* network occurs as a subgraph of a larger *world* network, and if so, exactly where it occurs and how many times [CFS03].

Subgraph matching is commonly used in many applications involving graph-structured data [CFS03]. For example, in bioinformatics [ZLY09], subgraph matching is used to identify biological patterns in protein-protein interaction networks. In social network analysis [Fan12, YG13], subgraph matching is used to identify social communities or groups. Subgraph matching is also an important subroutine in frequent subgraph mining [YH02, HWP03] and graph database search [ZMC11]. Despite the abundance of multiplex network data in these applications, there are relatively few subgraph matching algorithms that expressly support multiplex networks [IIP16, MBF20] compared to the number of algorithms that support non-multiplex networks [Ull76, CFS04, CFS17, CFS18, Sol10, HLL13, BCL16].

1.2 Definitions

We begin by defining networks, subgraphs, and subgraph isomorphism. Within this thesis, we will occasionally also refer to networks as *graphs.*

Definition 1.2.1 (Network). *A network $\mathcal{G} = (\mathcal{V}, \mathcal{E}, Src, Dst)$ is a set of nodes (also known as vertices) and edges. Each edge has a source node and a destination node. The function $Src : \mathcal{E} \rightarrow \mathcal{V}$ maps each edge to its source, and the function $Dst : \mathcal{E} \rightarrow \mathcal{V}$ maps each edge to its destination. The number of nodes is denoted n. The set of edges is disjoint from the set of nodes.*

Note that we do not define edges to be a subset of the Cartesian product of the node set with itself, as this would not allow for repeated elements. In general, we allow that there may be more than one edge between the same source and destination (i.e. the network may be a multigraph). This necessitates the use of the Src and Dst functions, as edges are not uniquely defined by their source and destination.

Definition 1.2.2 (Subgraph). *Given two networks $\mathcal{G}_1 = (\mathcal{V}_1, \mathcal{E}_1, Src_1, Dst_1)$ and $\mathcal{G}_2 = (\mathcal{V}_2, \mathcal{E}_2, Src_2, Dst_2)$, we say $\mathcal{G}_1$ is a* subgraph *of $\mathcal{G}_2$ if:*

1. $\mathcal{V}_1 \subseteq \mathcal{V}_2$
2. $\mathcal{E}_1 \subseteq \mathcal{E}_2$
3. $Src_1(E) = Src_2(E) \forall E \in \mathcal{E}_1$
4. $Dst_1(E) = Dst_2(E) \forall E \in \mathcal{E}_1$

Definition 1.2.3 (Subgraph Isomorphism). *Given two networks $\mathcal{G}_t$ and $\mathcal{G}_w$ (referred to as* template *and* world *respectively), we define a* subgraph isomorphism *as a set of two injective functions $f_{nodes} : \mathcal{V}_t \rightarrow \mathcal{V}_w$ and $f_{edges} : \mathcal{E}_t \rightarrow \mathcal{E}_w$ if:*

- $f_{nodes}(Src_t(E)) = Src_w(f_{edges}(E)) \forall E \in \mathcal{E}_t$

- $f_{nodes}(Dst_t(E)) = Dst_w(f_{edges}(E)) \forall E \in \mathcal{E}_t$

We call the subgraph of $\mathcal{G}_w$ that is mapped to by the subgraph isomorphism a *signal*, $\mathcal{G}_{\mathrm{s}}$. Though this injective mapping is not technically an isomorphism, it becomes one when you restrict the range to the signal nodes and edges, and it is in this sense that we refer to it as an isomorphism.

For sufficiently simple templates, there are efficient algorithms for counting and listing all corresponding signals [SW05, RKR17, ANR15]. However, in general, there can be a combinatorially large number of signals, particularly when the template has a large automorphism group. In such cases, listing or even counting the signals 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.

Within the context of Chapter 2, we consider *multiplex networks*, which differ from the networks defined in Definition 1.2.1 in that each edge belongs to a categorical *channel*. We also include the possibility of *labels* on nodes. However, in this multiplex context, we consider edges within the same channel and between the same two nodes to be indistinguishable. Thus, we omit the functions Src and Dst from Definition 1.2.1, and instead represent edges as a vector of counts in each channel.

Definition 1.2.4 (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. The number of distinguishable edges in $\mathcal{G}$ is denoted $|\mathcal{E}|_0$.*

For multiplex networks, we can store the edges in a set of *adjacency matrices*, one for each channel. The entry in the i^{th} row and j^{th} column of each adjacency matrix contains the number of edges from node i to node j in the corresponding channel.

1.2.1 Problem Statements

In this section, we will define five types of problems that will be explored in Chapter 2. We will define these within the multiplex network context using the definitions and notation from Definition 1.2.4.

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 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).*

Definition 1.2.5 (SI: Subgraph Isomorphism (Multiplex))**.** *Given two multiplex networks, a template $\mathcal{G}_t = (\mathcal{V}_t, \mathcal{E}_t, \mathcal{L}_t, \mathcal{C})$ and a world $\mathcal{G}_w = (\mathcal{V}_w, \mathcal{E}_w, \mathcal{L}_w, \mathcal{C})$, 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 an 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 [BGP13]. 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 Subsection 1.2.1 are summarized in terms of SIs in Table 1.1.

Problem	Description
SIP	Check if there are any SIs.
SNSP	Find all of 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 of the SIs.

Table 1.1: A summary of the various problems defined in Subsection 1.2.1, in increasing order of computation cost.

Nodes in the image of an SI are called *signal nodes*, and the induced subgraph of the image of an SI is called a *signal*. Figure 1.1 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 1.1 there are more edges between **5** and **7** than there are between **B** and **C**.

A typical definition of SI includes a map from template edges to world edges. However, we omit this consideration since for multiplex networks, we consider edges to be indistinguishable. If edges were considered to be distinct, there would be four SIs in Figure 1.1, all corresponding to signal 2. There are two dashed blue edges between world nodes **5** and **7** that could match the dashed blue edge between template nodes **B** and **C**. Also, there are two ways to map the two solid green edges between template nodes **A** and **C** to the two solid green edges between world nodes **4** and **7**, resulting in four possible combinations.

Definition 1.2.6 (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*

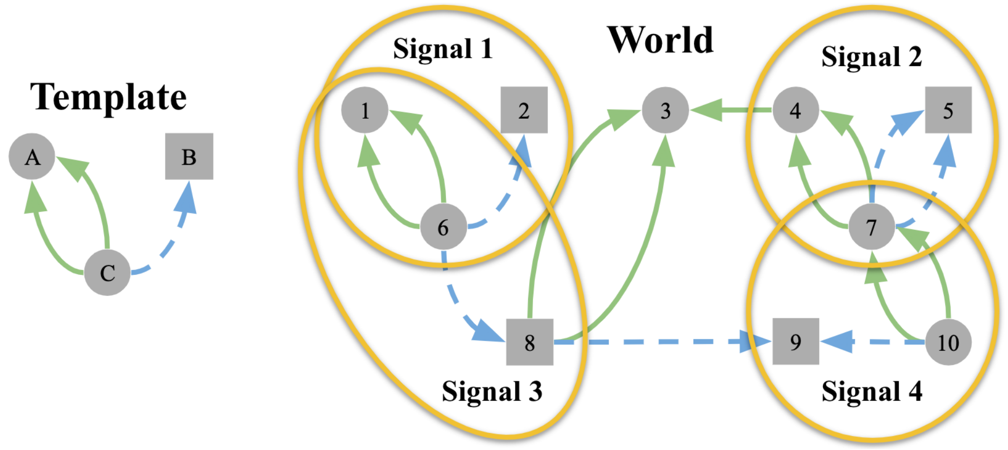


Figure 1.1: In the above multiplex networks, the shapes of the nodes corresponds to their labels (circle or square) and the patterns of the edges correspond to their channel (solid green or dashed blue). Given the template and world networks above, there are four signals consisting of the subgraphs of the world induced by $\{\mathbf{1}, \mathbf{2}, \mathbf{6}\}$, $\{\mathbf{1}, \mathbf{6}, \mathbf{8}\}$, $\{\mathbf{4}, \mathbf{5}, \mathbf{7}\}$, and $\{\mathbf{7}, \mathbf{9}, \mathbf{10}\}$.

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 [GJ79], it has been solved in practice even for some networks with over a billion nodes [SWW12, RKR17]. The challenge of finding all SIs, rather than simply checking whether there are any, is called the *subgraph matching problem (SMP)*.

Definition 1.2.7 (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 1.1 is the set $\mathcal{F}(\mathcal{G}_t, \mathcal{G}_w) = \{f_1, f_2, f_3, f_4\}$ of SIs, where the f_i are described in Table 1.2. 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 1.2: Solutions to SMP and MCSP corresponding to the template and world shown in Figure 1.1.

Since any algorithm for solving the SMP must list all of the SIs $\mathcal{F}(\mathcal{G}_t, \mathcal{G}_w)$, the computation time required, at minimum, must scale with the number $|\mathcal{F}(\mathcal{G}_t, \mathcal{G}_w)|$ of SIs. The number of SIs can be as large as $O(|\mathcal{V}_w|^{|\mathcal{V}_t|})$, which is the number of injective maps from $\mathcal{V}_t$ to $\mathcal{V}_w$. 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 [SW05] or motif counting [ADH08]. This is called the *SI counting problem (SICP).*

Definition 1.2.8 (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,*

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

Another concise summary of the set of SIs is the set of all signal nodes. We define the problem of finding this set of nodes to be the *signal node set problem (SNSP).* This can be useful in application contexts where the goal is to identify unusual or suspicious nodes that fit a pattern, rather than distinct instances of that pattern. The number of signal nodes is bounded above by the total number $|\mathcal{V}_w|$ of world nodes. Though the SNSP lacks the completeness of the full set of SIs, it is much cheaper to calculate.

Definition 1.2.9 (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,*

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

We can also summarize the set of SIs using the *minimal candidate sets.* For each template node, we define the minimal candidate set as 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. From an application perspective, this may be useful when certain template nodes are more important to identify than others, such as the core member of a social group or the leader of an adversarial organization.

Definition 1.2.10 (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 that correspond to $\mathbf{v}$ in at least one signal. That is,*

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

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 computationally expensive as the most expensive among those SIPs.

1.3 Prior Work

Most state-of-the-art algorithms for subgraph matching follow one of three approaches [CFS17, CFS18]: tree-search, constraint-propagation, and graph-indexing. The majority of existing algorithms are specialized to the single-channel case and are not directly applicable to multiplex networks without some adaptation. 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 [Ull76], VF2 [CFS04] and its variants (VF2Plus [CFV15], VF3 [CFS17, CFS18], VF2++ [JM18]), and for specific graphs, RI/RI-DS [BGP13].

Constraint-propagation approaches view SI 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 that indicates which world nodes 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 used to restrict the scope of a tree search for all solutions. Examples of constraint-propagation approaches include McGregor [McG79], nRF+ [LV02], ILF [ZDS10], LAD [Sol10] (and its variants, IncompleteLAD and PathLAD [KMS16]), Glasgow [MP15], and FocusSearch [Ull10].

Graph-indexing approaches seek to retrieve all graphs that match a given subgraph query from a database of graphs. 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, such as neighborhoods of nodes with distinctive features. Upon receiving a new template query, it is broken up into smaller subqueries matching these indexes. The results of these indexed queries are then typically joined together with a Cartesian product, identifying a large set of possible matches. Next, there is a verification step to check whether the joined results match the full pattern; this typically involves running another SI algorithm, such as VF2. Examples of graph-indexing approaches include GraphQL

[HS08], SPath [ZH10], GADDI [ZLY09], QuickSI [SZL08], TurboISO [HLL13], BoostISO [RW15], CFL-Match [BCL16], and CNI-Match [NS17]. Our problem does not involve graph databases and we thus do not construct indices. However, several graph-indexing approaches also involve filtering techniques, which can be used independently of the indices they construct [SL19].

1.3.1 Ullmann's Algorithm

Ullmann's algorithm [Ull76] is a backtracking tree-search with refinement. For each template node $\mathbf{v}$, it creates a list of candidate nodes in the world that 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, a 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 Subsection 2.1.3, is an extension of this constraint to the multiplex network case.

1.3.2 VF2

VF2 [CFS04] generalizes Ullmann's algorithm to directed graphs and extends refinement with additional semantic feasibility rules. It distinguishes matched nodes (i.e. 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.

1.3.3 Constraint-Propagation Approaches

In [LV02], the authors classify existing constraints using two categorizations: binary versus non-binary and forward checking (FC) versus really forward look ahead (RF). Binary constraints map n variables to m values using an $n \times m$ 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 lookahead 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 $\mathbf{v}_w$.

The work [Sol10] 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 *alldifferent constraint* introduced in [Reg94], which can be identified using the Hopcroft–Karp algorithm [HK73]. The alldifferent constraint requires that the matchings between neighborhoods must be one-to-one.

1.3.4 Graph-Indexing Approaches

In GraphQL [HS08], 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 [Sol10], 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 [ZH10], which considers matching k-distance neighborhoods of the template and candidate nodes.

In TurboISO [HLL13], the authors propose 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. They then 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 prioritize searching the smaller candidate subregions. We take a similar approach in Subsection 2.2.1, prioritizing nodes with the fewest candidates.

In CFL-Match [BCL16], 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 nontree 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 structure.

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 to which it is connected. 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.

1.3.5 Multiplex Approaches

Although most subgraph isomorphism algorithms in the literature focus on non-multiplex networks, there are two algorithms that explicitly solve the multiplex SMP. SuMGrA [IIP16] is a graph-indexing and backtracking tree-search approach and RI [BGP13] has been extended to the multiplex case (MultiRI) [MBF20]. 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 2.3. Additionally, the networks considered by SuMGrA and MultiRI differ slightly from Definition 1.2.4. Although 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 non-multiplex algorithms to multiplex networks, one possibility is to use a non-multiplex approach (e.g., VF2 or 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 2.3, we show examples where the number of SIs is too large to solve the multiplex SMP, let alone the non-multiplex SMP.

1.3.6 Review

In Chapter 2, we first discuss our approach for exact subgraph matching via constraint propagation, showing approaches for all five problems defined in Subsection 1.2.1. Next, in Chapter 3, we show an analogous approach for inexact subgraph matching using graph edit distance minimization with cost bounds. In Chapter 4, we address a separate inexact matching problem on aligned networks, which handles potential errors introduced by the network alignment. Lastly, in Chapter 5, we detail a similar constraint propagation approach for identifying constrained path-based templates.

CHAPTER 2

Subgraph Matching on Multiplex Networks*

In this chapter, we introduce a new algorithm for subgraph matching on multiplex networks and discuss some simplifications of the subgraph matching problem. We extend existing constraint satisfaction approaches to operate on multiplex networks. We demonstrate experimentally that these approaches allow us to list or count all SIs for world graphs with up to hundreds of thousands of nodes and templates with up 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.

In the remainder of this section, we define several problems related to subgraph matching and discuss existing approaches to solve these problems. We expand on the details of our approach in Section 2.1 and Subsections 2.2.1 and 2.2.2. In Section 2.3, we perform several experiments to show that the methods we discuss are successful in solving the problems of interest.

*This chapter is adapted from [MTC21].

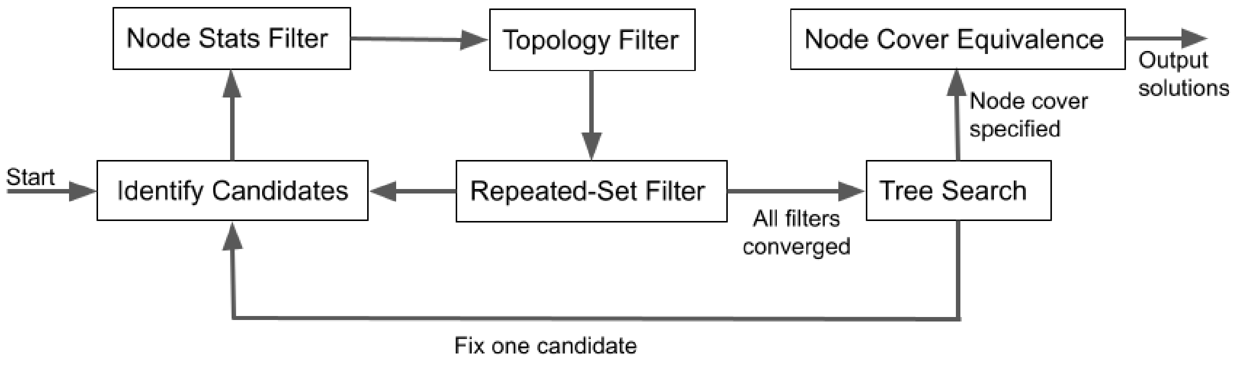


Figure 2.1: Flowchart of the algorithm for subgraph matching

2.1 Filtering

Our algorithms for solving the problems discussed in Subsection 1.2.1 revolve around the use of *filters* to cheaply approximate the solution of the MCSP (Definition 1.2.10). 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 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. We then begin a depth-first tree search for potential matches, rerunning filters at each stage of the tree. For a flowchart of this process, see Figure 2.1

In this thesis, 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 Algorithm 5, change $\geq$ to $==$ each time it appears on Line 8. Additionally, all nodes must be considered as "neighbors" of each other, even if they have no edges between them; this affects Line 2 of Algorithm 3 and Lines 4 and 5 of Algorithm 5.

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 ← False
3: while converged is False
4:     converged ← True                                    ▷ Stop unless progress is made
5:     for filter ∈ 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 ← False                            ▷ Progress was made
9:             Break                                        ▷ Restart from the first filter
10: Output updated candidate set $D(\mathbf{v}_t)$ for each $\mathbf{v}_t \in \mathcal{V}_t$
```

2.1.1 Node Label Filter

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 1.1 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 examples, template node labels cannot be specified exactly. For example, in geospatial applications, template node labels may represent geographic regions such as cities or countries, 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 Subsection 2.3.3.1 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.

2.1.2 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 [Sol10, MBF20]. 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 are 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 multiplex networks.

Algorithm 2 Node-level Statistics Filter

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

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 ℓ. 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. To keep the computation time of our algorithm low, we only use statistics for which the computational cost scales linearly in the number of distinguishable edges (edges whose source, destination, direction, and channel are distinct).

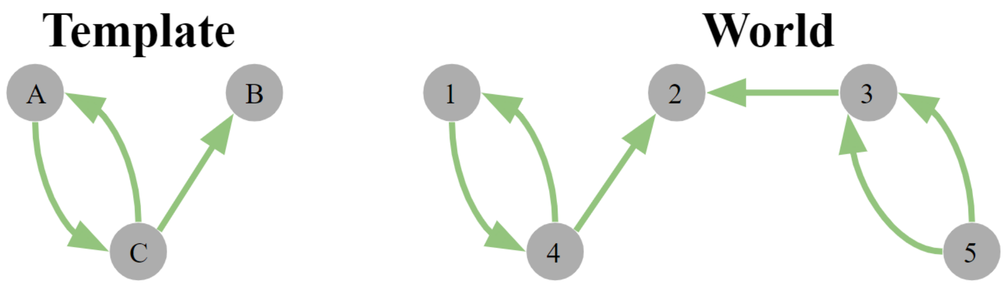


Figure 2.2: In the networks shown, there is only one valid signal for the template: **1** for **A**, **2** for **B**, and **4** for **C**.

	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 2.1: In/out-degree for nodes in the template and world shown in Figure 2.2

As an example, consider applying an in/out-degree filter to the problem in Figure 2.2. The in-degree and out-degree of each template and world node are listed in Table 2.1. Node **A** has one outgoing edge and one incoming edge. Thus, nodes **2** and **5** can be ruled out as candidates since node **2** has no outgoing edges and node **5** has no incoming edges. Similarly, node **B** has one incoming edge, so node **5** can be ruled out as a candidate since it does not have any incoming edges. Finally, node **C** has two outgoing edges and one incoming edge, so all world nodes except **4** are eliminated as candidates since **4** is the only world node with at least two outgoing edges and at least one incoming edge. The candidates for each template node after applying the node-level statistics filter are summarized in the "Statistics" column of Table 2.2.

The node-level statistics filter is most effective when some template nodes have statistic

	Filters applied		
	statistics	statistics, topology	statistics, topology, repeated-sets
Candidates for **A**	$\mathbf{1}, \mathbf{3}, \mathbf{4}$	$\mathbf{1}$	$\mathbf{1}$
Candidates for **B**	$\mathbf{1}, \mathbf{2}, \mathbf{3}, \mathbf{4}$	$\mathbf{1}, \mathbf{2}$	$\mathbf{2}$
Candidates for **C**	$\mathbf{4}$	$\mathbf{4}$	$\mathbf{4}$

Table 2.2: Candidates per template node for the problem shown in Figure 2.2 after various filters have been applied.

values that are 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.

2.1.3 Topology Filter

The topology filter enforces the constraint proposed by [Ull76], extended to multiplex networks. The original constraint, denoted as *AC(Edges)* [Sol10], is that if $\mathbf{v}_w$ is a candidate for template node $\mathbf{v}_t$, then for every template node $\mathbf{u}_t$ that neighbors $\mathbf{v}_t$, there must exist a candidate $\mathbf{u}_w$ for $\mathbf{u}_t$ that neighbors $\mathbf{v}_w$. The natural extension to multiplex networks is that if $\mathbf{v}_w$ is a candidate for template node $\mathbf{v}_t$, then for every template node $\mathbf{u}_t$ that neighbors $\mathbf{v}_t$, there must exist a candidate $\mathbf{u}_w$ for $\mathbf{u}_t$ that 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 $\mathbf{u}_t$.

To clarify the concept, consider applying the topology filter to the problem in Figure 2.2 after applying the node-level statistics filter. Since node **4** is the only candidate for node **C**, nodes **3** and **4** are eliminated as candidates for both nodes **A** and **B** because they do not have neighbors that are candidates for node **C**. These results are shown in Table 2.2.

Algorithm 3 Topology Filter

```
1: **Input** template $\mathcal{G}_t$, world $\mathcal{G}_w$, candidate set $D(\mathbf{v}_t)$ for each $\mathbf{v}_t \in \mathcal{V}_t$
2: for neighboring template nodes $\mathbf{v}_t$ and $\mathbf{u}_t$
3:     for candidate $\mathbf{v}_w \in D(\mathbf{v}_t)$
4:         nbr_cand_found $\leftarrow$ False
5:         for candidate $\mathbf{u}_w \in D(\mathbf{u}_t)$
6:             enough_out $\leftarrow \mathcal{E}_w(\mathbf{v}_w, \mathbf{u}_w) \geq \mathcal{E}_t(\mathbf{v}_t, \mathbf{u}_t)$
7:             enough_in $\leftarrow \mathcal{E}_w(\mathbf{u}_w, \mathbf{v}_w) \geq \mathcal{E}_t(\mathbf{u}_t, \mathbf{v}_t)$
8:             if enough_out and enough_in
9:                 nbr_cand_found $\leftarrow$ True
10:                break
11:        if nbr_cand_found is not True
12:            Remove $\mathbf{v}_w$ from $D(\mathbf{v}_t)$
13: Output updated candidate set $D(\mathbf{v}_t)$ for each $\mathbf{v}_t \in \mathcal{V}_t$
```

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

2.1.4 Repeated-Sets Filter

Here, we apply another constraint, *GAC(AllDiff)* [Hoe01, Sol10]: Generalized Arc Consistency (also known as hyper-arc consistency) for the AllDifferent constraint, the constraint that all nodes mapped to must be different (i.e. injectivity). The *GAC(AllDiff)* constraint 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* [Hoe01]. 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 [Reg94, GMN08]. Some improvements and modifications to the standard algorithms have been explored [GMN08]. In the repeated-sets filter (Algorithm 4), we enforce *GAC(AllDiff)* by directly considering unions of candidate sets to find tight sets. Although the number of candidate set unions grows exponentially with the number of template nodes [Hoe01], 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 the context of the datasets we consider, templates are often hundreds of nodes or smaller, and we do not 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 $|\mathcal{V}_w|$.

To illustrate the application of the repeated-sets filter, consider the example in Figure 2.2. Using the node-level statistics and topology filters, the candidates for the template nodes are narrowed down to those in Table 2.2. Since $\mathbf{A}$ has only one candidate, $\{\mathbf{A}\}$ is a tight set and $\mathbf{1}$ 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 2.2.

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 Subsections 2.3.4.1 and 2.3.4.5 that contain nodes that are indistinguishable.

Algorithm 4 Repeated-Sets Filter

```
1: Input template $\mathcal{G}_t$, world $\mathcal{G}_w$, candidate set $D(\mathbf{v}_t)$ for each $\mathbf{v}_t \in \mathcal{V}_t$
2: unions $\leftarrow$ EmptyMap()
     $\triangleright$ 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.
3: $\mathcal{T} \leftarrow \emptyset$                                  $\triangleright$ Nodes known to belong to tight sets.
4: todo $\leftarrow \{\mathbf{v}_t \in \mathcal{V}_t : |D(\mathbf{v}_t)| < |\mathcal{V}_t|\}$          $\triangleright$ Template nodes with too many candidates
                                                     cannot belong to a tight set.
5: while $|$todo$| > 0$
6:     $\mathbf{v}_t \leftarrow$ element of todo with fewest candidates $D(\mathbf{v}_t)$
7:     todo $\leftarrow$ todo $\setminus \{\mathbf{v}_t\}$
8:     for $(\mathcal{U}_w, \mathcal{U}_t) \in$ unions
9:         $\mathcal{U}_w \leftarrow \mathcal{U}_w \cup D(\mathbf{v}_t)$
10:        if $|\mathcal{U}_w| < |\mathcal{V}_t|$
11:            $\mathcal{U}_t \leftarrow \mathcal{U}_t \cup \{\mathbf{v}_t\}$
12:            if $|\mathcal{U}_t| == |\mathcal{U}_w|$                       $\triangleright$ $\mathcal{U}_t$ is a tight set.
13:                $\mathcal{T} \leftarrow \mathcal{T} \cup \mathcal{U}_t$
14:                $D(\mathbf{u}_t) \leftarrow D(\mathbf{u}_t) \setminus \mathcal{U}_w$ for $\mathbf{u}_t \in \mathcal{V}_t \setminus \mathcal{U}_t$
15:            else
16:                unions$[\mathcal{U}_w] \leftarrow \mathcal{U}_t$
17:    if $D(\mathbf{v}_t) \notin$ unions
18:        if $|D(\mathbf{v}_t)| == 1$                              $\triangleright$ $\{\mathbf{v}_t\}$ is a tight set.
19:            $\mathcal{T} \leftarrow \mathcal{T} \cup \{\mathbf{v}_t\}$
20:            $D(\mathbf{u}_t) \leftarrow D(\mathbf{u}_t) \setminus D(\mathbf{v}_t)$ for $\mathbf{u}_t \in \mathcal{V}_t \setminus \{\mathbf{v}_t\}$
21:        else
22:            unions$[D(\mathbf{v}_t)] \leftarrow \{\mathbf{v}_t\}$
23:    todo $\leftarrow \{\mathbf{v}_t \in$ todo $: |D(\mathbf{v}_t)| < |\mathcal{V}_t \setminus \mathcal{T}|\}$
24: Output updated candidate set $D(\mathbf{v}_t)$ for $\mathbf{v}_t \in \mathcal{V}_t$
```

2.1.5 Neighborhood Filter

In this filter, we extend the *local (LAD)* constraint introduced in [Sol10] to the multiplex network case. In the undirected non-multiplex graph context, the LAD constraint states 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 constraint to the multiplex network context by requiring that $f_\ell(\mathbf{u}_t)$ must have enough edges to $\mathbf{v}_w$ in each channel and direction, i.e.

$$\mathcal{E}_w(\mathbf{v}_w, \mathbf{u}_w) \geq \mathcal{E}_t(\mathbf{v}_t, \mathbf{u}_t) \quad \text{and} \quad \mathcal{E}_w(\mathbf{u}_w, \mathbf{v}_w) \geq \mathcal{E}_t(\mathbf{u}_t, \mathbf{v}_t), \tag{2.1}$$

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 (2.1) 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 $|\mathcal{N}_{\mathbf{v}_t}|$ when 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 [HK73] 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 $\mathcal{B}$.

In the example from Figure 2.2, the node-level statistics, topology, and repeated-sets filters were sufficient to narrow down the candidates until they solve the MCSP (Definition 1.2.10). In Figure 2.3, we give an example where those filters are not sufficient. The resulting candidates before and after neighborhood filter are given in Table 2.3. 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 (the nonzero upper right block of the bipartite

Algorithm 5 Neighborhood Filter

```
1: Input template $\mathcal{G}_t$, world $\mathcal{G}_w$, candidate set $D(\mathbf{v}_t)$ for each $\mathbf{v}_t \in \mathcal{V}_t$
2: for template node $\mathbf{v}_t \in \mathcal{V}_t$
3:     for candidate $\mathbf{v}_w \in D(\mathbf{v}_t)$
4:         $\mathcal{N}_{\mathbf{v}_t} \leftarrow \texttt{Neighborhood}(\mathbf{v}_t)$
5:         $\mathcal{N}_{\mathbf{v}_w} \leftarrow \texttt{Neighborhood}(\mathbf{v}_w)$
6:         $\mathcal{B} \leftarrow \texttt{EmptyBipartiteGraph}(\mathcal{N}_{\mathbf{v}_t}, \mathcal{N}_{\mathbf{v}_w})$
7:         for $(\mathbf{u}_t, \mathbf{u}_w) \in \mathcal{N}_{\mathbf{v}_t} \times \mathcal{N}_{\mathbf{v}_w}$
8:             if $\begin{cases} \mathbf{u}_w \in D(\mathbf{u}_t) \\ \textit{and } \mathcal{E}_w(\mathbf{v}_w, \mathbf{u}_w) \geq \mathcal{E}_t(\mathbf{v}_t, \mathbf{u}_t) \\ \textit{and } \mathcal{E}_w(\mathbf{u}_w, \mathbf{v}_w) \geq \mathcal{E}_t(\mathbf{u}_t, \mathbf{v}_t) \end{cases}$
9:                 Add an edge between $\mathbf{u}_t$ and $\mathbf{u}_w$ in $\mathcal{B}$
10:        $\texttt{max\_match} = \texttt{MaxCardinalityMatching}(\mathcal{B})$
11:        if $|\texttt{max\_match}| < |\mathcal{N}_{\mathbf{v}_t}|$
12:            Remove $\mathbf{v}_w$ from $D(\mathbf{v}_t)$
13: Output updated candidate set $D(\mathbf{v}_t)$ for each $\mathbf{v}_t \in \mathcal{V}_t$
```

adjacency matrix) of $\mathcal{B}$ corresponding to $\mathbf{v}_t = \mathbf{C}$ and $\mathbf{v}_w = \mathbf{4}$ is shown in Table 2.4. Each row of the biadjacency matrix corresponds to a neighbor of $\mathbf{v}_t$ and each column corresponds to a neighbor of $\mathbf{v}_w$. Entries correspond to the conditions on Line 8 of Algorithm 5. Since the maximum cardinality matching on $\mathcal{B}$ has only two elements, the neighborhood filter is able to eliminate node **4** from the candidates of node **C**.

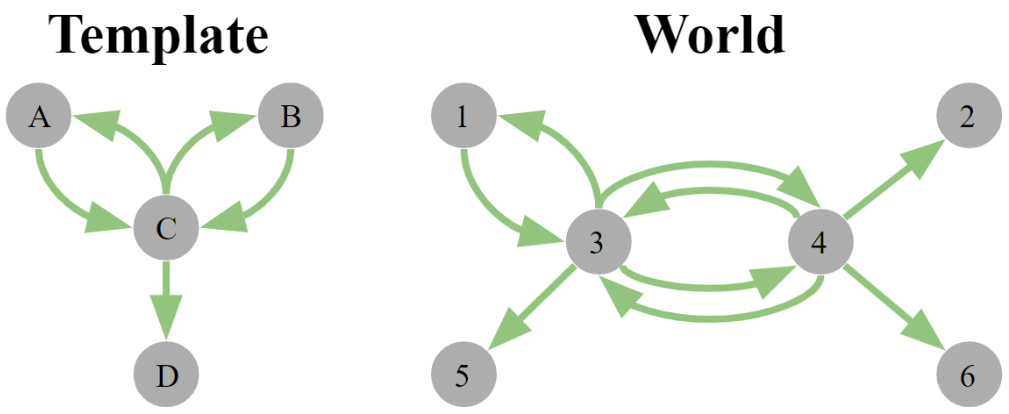


Figure 2.3: An example template and world for which the neighborhood filter plays a role in eliminating candidates.

2.1.6 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 2.4, where we have three cycle

	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 2.3: Candidates per template node for the problem shown in Figure 2.3 after various filters have been applied.

graphs consisting of 3, 4 and 5 nodes respectively.

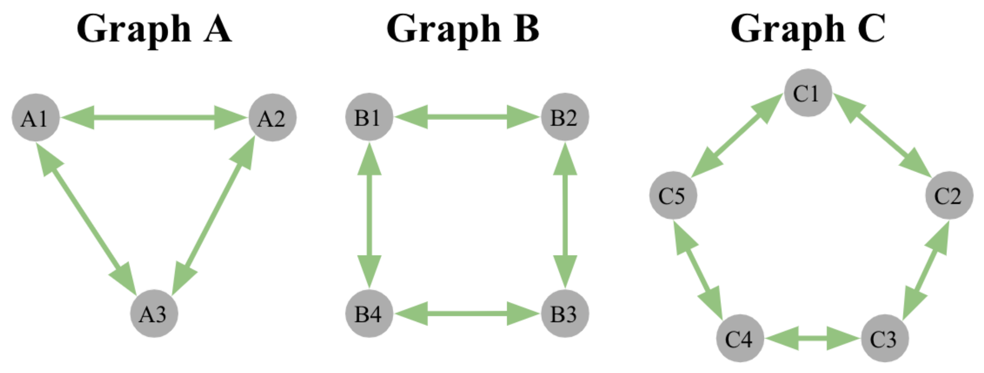


Figure 2.4: Subgraph matching problems consisting of graphs A, B and C.

Consider Graph A in Figure 2.4 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 the elimination filter, we observe 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

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

Table 2.4: Biadjacency matrix of $\mathcal{B}$ used in the matching problem between the neighborhoods of template node **C** and world node **4**.

case, the node-level statistics, topology, and neighborhood filters do not remove any invalid candidates. Only after applying the elimination filter can we conclude that there is no valid signal existing in Graph C. In practice, an elimination filter that iterates over node-level statistics, topology, and repeated-sets filters is often sufficient; iterating over neighborhood filters is expensive. However, in the example above, we may need to apply all of the filters to determine the solution. In Subsection 2.3.4, we show some examples where the elimination filter is useful in eliminating candidates.

2.2 Solving the Problems

We present the details of how to solve the SICP and MCSP using the filters from Section 2.1 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.

2.2.1 Isomorphism Counting

After applying the filters described in Section 2.1, some template nodes may have exactly one candidate. We refer to template nodes that still have multiple candidates as *unspecified* nodes. When an edge exists between two unspecified nodes, we have to enforce that a

Algorithm 6 Elimination Filter

```
1:  Input template $\mathcal{G}_t$, world $\mathcal{G}_w$, candidate set $D(\mathbf{v}_t)$ for each $\mathbf{v}_t \in \mathcal{V}_t$
2:  $S \leftarrow$ copy($D$)                                    ▷ Save the candidate sets
3:  for template node $\mathbf{v}_t \in \mathcal{V}_t$
4:      for candidate $\mathbf{v}_w \in D(\mathbf{v}_t)$
5:          $D(\mathbf{v}_t) \leftarrow \{\mathbf{v}_w\}$                      ▷ Assign $\mathbf{v}_w$ to $\mathbf{v}_t$
6:          Iterate all filters to convergence
7:          if $D(\mathbf{u}_t)$ is empty for any $\mathbf{u}_t \in \mathcal{V}_t$
8:              Remove $\mathbf{v}_w$ from $D(\mathbf{v}_t)$ in $S$
9:          Reset candidate sets to $S$
10: Output updated candidate set $D(\mathbf{v}_t)$ for each $\mathbf{v}_t \in \mathcal{V}_t$
```

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 that, 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 2.2, if all three template nodes have multiple candidates, the minimal node cover would be $\{\mathbf{C}\}$. Since the minimal node cover is NP-hard to compute in general, we settle for a small node cover [BE85]. 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 [Reg94].

To solve the SIP and SMP, we can make the following modifications. For the SIP, the algorithm exits and returns true once a solution is found, i.e. once `CountAllDiff` (Line 4

Algorithm 7 Isomorphism Counting

```
1: function IsoCount
2:     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 $\mathcal{NC}$
3:     if $\mathcal{NC} = \emptyset$
4:         $\texttt{N}_{\texttt{isos}} \leftarrow \texttt{CountAllDiff}(D)$
5:     else
6:         $\texttt{N}_{\texttt{isos}} \leftarrow 0$
7:         $\mathbf{v}_t \leftarrow \mathcal{NC}.\texttt{pop}()$
8:         $S \leftarrow \texttt{copy}(D)$                                ▷ Save the candidate sets
9:         for world node $\mathbf{v}_w \in D(\mathbf{v}_t)$
10:            $D(\mathbf{v}_t) \leftarrow \{\mathbf{v}_w\}$                          ▷ Assign $\mathbf{v}_w$ to $\mathbf{v}_t$
11:            Apply filters
12:            $\texttt{N}_{\texttt{isos}} \leftarrow \texttt{N}_{\texttt{isos}} + \texttt{IsoCount}(\mathcal{G}_t, \mathcal{G}_w, D, \mathcal{NC})$
13:            Reset candidate sets to $S$
14:    Output number of isomorphisms $\texttt{N}_{\texttt{isos}}$
```

of Algorithm 7) returns a nonzero result. For the SMP, replace `CountAllDiff` with an algorithm that returns a list of all solutions to the alldifferent problem, and combine the two lists of solutions instead of adding the counts together.

2.2.2 Validation

A simpler problem than listing or counting all isomorphisms (SMP or SICP) is the minimal candidate sets problem (MCSP) of Definition 1.2.10. 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 signal. We call such pairs *valid*.

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 needed. 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 are validated as taking part in at least one isomorphism. Repeat this process until all pairs are validated or removed from the set of candidates. This ensures that every candidate-template node pairing remaining after validation is valid. This is because to pass validation, it must participate in at least one isomorphism that was found, and to fail, there must not exist an isomorphism containing it.

When using the node cover approach from Subsection 2.2.1, we can make Algorithm 8 more efficient by stopping the search after the node cover has been assigned and validating all remaining candidates. This is because at this point, with no edges between unassigned nodes, the problem of finding signals has been reduced to an alldifferent problem. Because the repeated-sets filter enforces *GAC(AllDiff)*, we know that every remaining candidate must participate in at least one solution to the alldifferent problem, and thus must participate in at least one signal. Therefore, we can simultaneously validate all remaining candidate pairs.

This algorithm can also be modified to solve the SNSP. If, instead of adding $\{f(\mathbf{u}_t)\}$ to

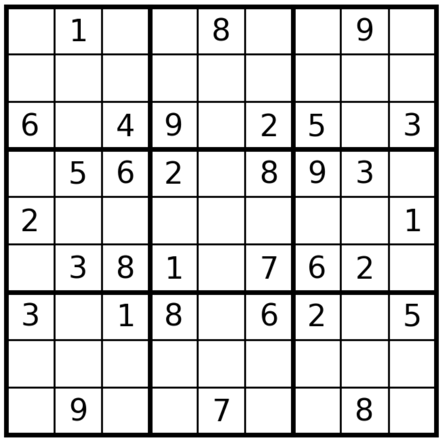
Algorithm 8 Validation

```
1: **Input** template $\mathcal{G}_t$, world $\mathcal{G}_w$, candidate set $D(\mathbf{v}_t)$ for each $\mathbf{v}_t \in \mathcal{V}_t$
2: $D'(\mathbf{v}_t) \leftarrow \emptyset$ for each $\mathbf{v}_t \in \mathcal{V}_t$                    ▷ Valid candidates
3: for template node $\mathbf{v}_t \in \mathcal{V}_t$
4:     for candidate $\mathbf{v}_w \in D(\mathbf{v}_t) \setminus D'(\mathbf{v}_t)$
5:         $S \leftarrow \texttt{copy}(D)$
6:         $S(\mathbf{v}_t) \leftarrow \{\mathbf{v}_w\}$                    ▷ Assign $\mathbf{v}_w$ to $\mathbf{v}_t$
7:         Attempt to find an SI $f$ using $S$                    ▷ Solve the SIP
8:         if isomorphism found
9:             $D'(\mathbf{u}_t) \leftarrow D'(\mathbf{u}_t) \cup \{f(\mathbf{u}_t)\}$ for each $\mathbf{u}_t \in \mathcal{V}_t$
10:        else
11:            Remove $\mathbf{v}_w$ from $D(\mathbf{v}_t)$
12: Output minimal candidate set $D'(\mathbf{v}_t)$ for each $\mathbf{v}_t \in \mathcal{V}_t$
```

$D'(\mathbf{u}_t)$ on Line 9, we instead add $\{f(\mathbf{v}_t) | \mathbf{v}_t \in \mathcal{V}_t, f(\mathbf{v}_t) \in D(\mathbf{u}_t)\}$, then the algorithm will validate only that world nodes participate in at least one signal without respect to which template nodes they are candidates for.

2.3 Experiments

Unless otherwise specified, the experiments were run on an HP Z8 G4 Workstation with two 12-core Intel® Xeon® Gold 6136 CPUs using version 0.2.0 of our open-source Python code available on GitHub (`https://github.com/jdmoorman/uclasm/tree/v0.2.0`) [MTC20]. 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 between minutes and hours, depending on the dataset. Validation, if used, typically takes multiple hours.



	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	

Figure 2.5: Sudoku grid with initial clues.

2.3.1 Sudoku

The puzzle game of Sudoku involves a 9×9 grid that is partitioned into nine 3×3 *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 2.5).

One algorithm for solving Sudoku is Donald Knuth's dancing links algorithm [Knu00], 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 [Sim05] (e.g. Peter Norvig's constraint-propagation algorithm [Norb]). Here, the two constraints propagated are that cells in the same row, column, or block 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 constraints of Sudoku are fairly similar to those of subgraph isomorphism. The row, column, and block constraints involve pairwise constraints between two cells, which can be considered analogous to edges. Additionally, the constraint that every cell must be filled by a digit is analogous to injectivity. Here, we show that Sudoku can be written in two different ways as a subgraph isomorphism problem. Though there is some significant computational overhead in performing this conversion, our main goal here is to obtain an interesting example of subgraph isomorphism to evaluate our code on, rather than trying to outperform the best Sudoku solvers.

2.3.1.1 9×9 Representation

Sudoku is equivalent to a 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 3×3 block. We consider the world as a set of 81 nodes, each corresponding to a digit. Each digit is locked to a particular 3×3 block, leading to nine sets of nine digits, each having values 1–9. In the world graph, two nodes are linked if they do *not* have the same value. This world graph has the same number of nodes than the template, but has more edges, so this is a subgraph isomorphism problem. To represent known digits (to start the puzzle), we can restrict the initial candidates. If a cell is filled in with a value, we identify which 3×3 block the cell is in, find the digit in the world that corresponds to that block and has the matching value, and enforce that the only candidate for the cell is that digit.

2.3.1.2 $9 \times 9 \times 3$ Representation

One can also represent row, column or block correspondences as different channels with different edge types, along with another edge type that identifies which cells are the same. In this representation, 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 graph consists of three sets of nine sets of nine digits, each of the three sets corresponding to the row, column, or block channels. As before, there are the same number of world nodes as template nodes. In each channel, sets of nine digits representing the same row/column/block are connected by edges. In these channels, there are the same number of edges as 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 3×3 block only intersects three rows and three columns, digits corresponding to cells in that block will only have "same-cell" edges to digits corresponding to cells in the rows or columns that intersect it. Additionally, the digits must match (e.g. a 9 can only have "same-cell" edges to other 9's.) 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.

2.3.1.3 Results

To test our filters, we use the sets of Sudoku puzzles obtained from Peter Norvig's GitHub [Nora], including those from Project Euler problem 96 [Pro]. This dataset is divided into 3 groups: "50easy", which contains the 50 puzzles from Project Euler problem 96, "top95", which contains 95 puzzles from the magictour website[Ste05], and "hardest", which contains 11 hard puzzles found by Peter Norvig via a search for "hardest Sudoku".

Although our code is not specialized for the task, it still solves all Sudoku examples in the dataset. The time taken to solve each Sudoku example is plotted in Figure 2.6. The mean solution time for the 9×9 representation 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. The mean solution time for the $9 \times 9 \times 3$ representation 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.

Overall, it appears that the 9×9 representation works best on the easier puzzles, while the $9 \times 9 \times 3$ representation is faster on harder puzzles. On the hardest puzzles, the 9×9 representation is faster on average; this is primarily due to outliers for which the $9 \times 9 \times 3$ representation takes much longer.

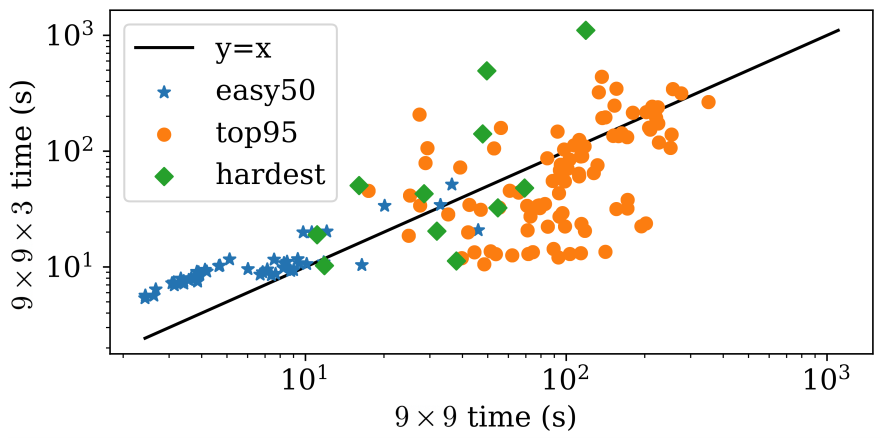


Figure 2.6: Solving Sudoku puzzles as special case of a multiplex subgraph isomorphism problem using the 9×9 and $9 \times 9 \times 3$ representations. Scatter plot of solution times (seconds) on the Sudoku puzzles from Peter Norvig's GitHub [Nora], including those from Project Euler problem 96 [Pro]. A black line is drawn where the representations take an equal amount of time as a visual aid for comparing the two representations.

2.3.2 Multiplex Erdős–Rényi Networks

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 template graphs, $p_t = 0.5$. For the world graphs, the probability p_w is scaled up as the number of channels increases. We do this to maintain a fixed probability of $1 - p_t + p_t^2$ that a randomly generated world edge vector has an edge in every channel that a random template vector does. This gives us the condition

$$\mathbb{P}(\mathcal{E}_t(\mathbf{u}_t, \mathbf{v}_t) \leq \mathcal{E}_w(\mathbf{u}_w, \mathbf{v}_w)) = 1 - p_t + p_t^2,$$

for all template nodes $\mathbf{u}_t$, $\mathbf{v}_t$ and world nodes $\mathbf{u}_w$, $\mathbf{v}_w$. With edge vectors of length $|\mathcal{C}|$, this leads us to the following formula for p_w:

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

Note that when $|\mathcal{C}| = 1$, $p_w = p_t$; this was the motivation for the choice of $1 - p_t + p_t^2$.

For each trial, we generate a template graph with 10 nodes and a world graph with size ranging from 10 nodes to 300 nodes. We then solve both the SIP and SICP. To measure the difficulty of these problems, 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 channels (Figure 2.7). We cap the algorithm runtime at 10,000 seconds per trial and average results over 100 trials.

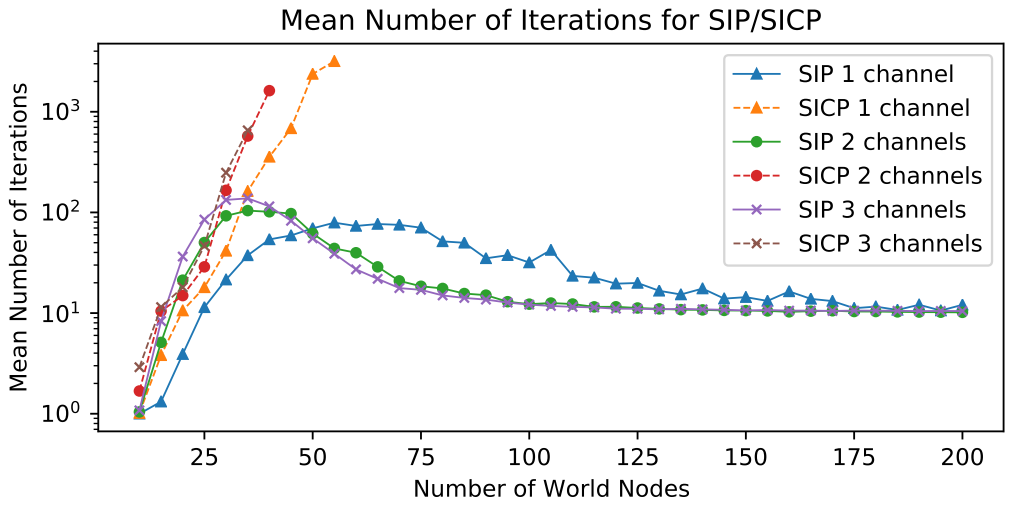


Figure 2.7: Mean number of search iterations (recursive calls to `IsoCount`) taken by Algorithm 7 to solve the SIP and SICP with one, two, and three channels as a function of world size. Results are averaged over 500 trials.

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 [MPS18]. When SIs are unlikely, as when the world is near the size of the template, filtering often rules out all SIs before reaching the bottom of the search tree. When SIs are common, as when the world is large (e.g. over 100 nodes), 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 mean 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.

2.3.3 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 2.5 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 (3 km)	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	15,367,315	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 2.5: Sizes and filtering results for the real-world examples in Sections 2.3.3.1, 2.3.3.2 and 2.3.3.3. The last column records the types of problems stated in Subsection 1.2.1 that we are able to solve. The names of the filters are abbreviated: L = Node Label; S = Node-level Statistics; T = Topology; R = Repeated-Sets; N = Neighborhood; E = Elimination.

2.3.3.1 Great Britain Transportation

The Great Britain Transportation Network [GB15] combines a public transportation dataset available through the United Kingdom open-data program [Dep15] 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 six different transportation methods: air, ferry, railway, metro, coach, and bus. This dataset has 262,377 nodes and 475,502 edges. The original dataset can be found at [GB15].

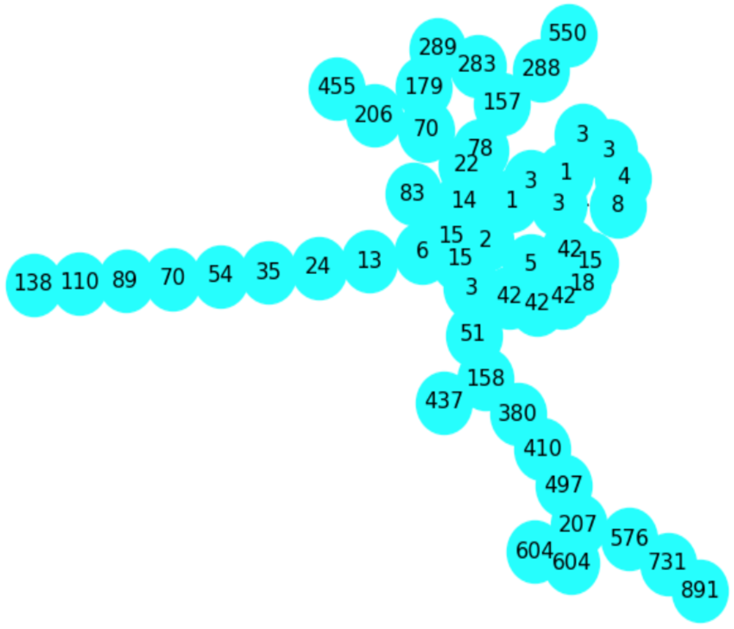


Figure 2.8: Results for the Great Britain Transportation template after applying the node-level statistics, topology, repeated-sets and elimination filters, without node label filtering. The number in each node is the number of remaining candidates for that node after filters.

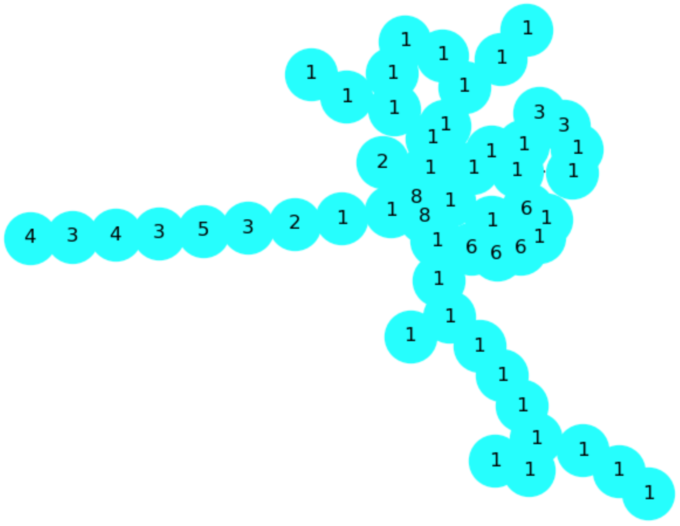


Figure 2.9: Results for the Great Britain Transportation template after applying the node label filter with a 3 km radius in addition to the node-level statistics, topology, repeated-sets, and elimination filters. The number in each node is the number of remaining candidates for that node after filters.

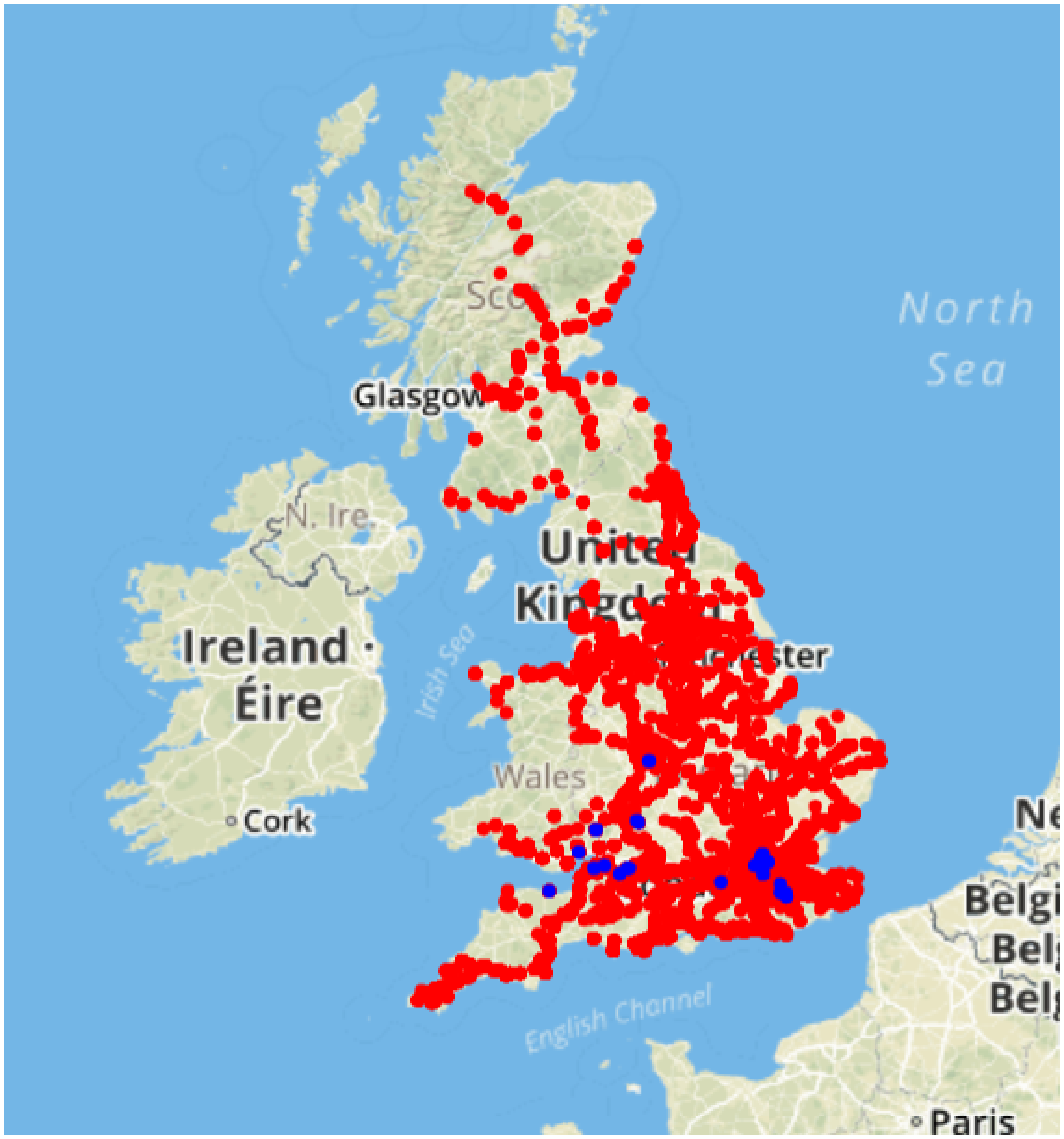


Figure 2.10: All candidates for the Great Britain Transportation dataset: blue nodes represent template nodes and red nodes represent candidates for the SIP. Image was generated using the Mapbox API[Map] with data from Open Street Map.[Ope]

To test our algorithm, we first create a template. We identify a small set of locations that interact with each other through five out of six channels (we exclude air, 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 five channels. We arbitrarily 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. We then repeatedly apply the node-level statistics, topology, and repeated-sets filters. The resulting candidate counts are shown in Figure 2.8.

We can further reduce the number of candidates using the node label filter. The result is shown in Figure 2.9. We observe that all nodes are reduced to fewer than 10 candidates, with the majority having just a single candidate.

In Figure 2.10, we see that the remaining candidates after filtering are geographically distributed across the UK. As previously mentioned in Subsection 2.1.1, these nodes represent 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 graph 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 use 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.

2.3.3.2 Higgs Twitter

The Higgs Twitter dataset [DLM13] records Twitter activities from July 1–7, 2012, during and after the discovery of the Higgs boson particle. It can be represented as a directed, multiplex network with four channels, representing retweets, replies, mentions and follower relationships. The world graph contains 456,626 nodes (Twitter accounts) and 15,367,315 edges (interactions), and the account identities are 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 accounts frequently involved in retweets or replies during the week, and whose induced subgraph is connected and contains edges in all four channels. The template has a total of 2,668 edges.

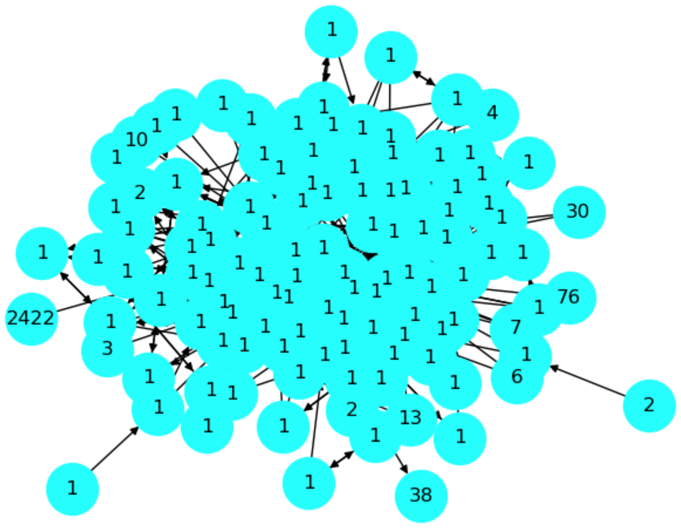


Figure 2.11: Candidate count for each node in the Higgs Twitter template after applying the node-level statistics, topology, and repeated-sets filters.

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 2.11). For the remaining template nodes with multiple candidates, we find that most of the candidates are valid (i.e. they belong to at least one subgraph isomorphism). Using Algorithm 7, we compute that there are 1.03×10^{14} subgraph isomorphisms.

2.3.3.3 Commercial Airlines

We also test our filtering methods on a commercial airlines dataset [CGZ13]. This dataset contains a multiplex airline network in Europe that consists of 37 channels, each representing a different airline. Compared with the other real world examples above, its world graph is relatively small, only containing 450 nodes (airports) and 7,177 directed, unweighted edges (flights).

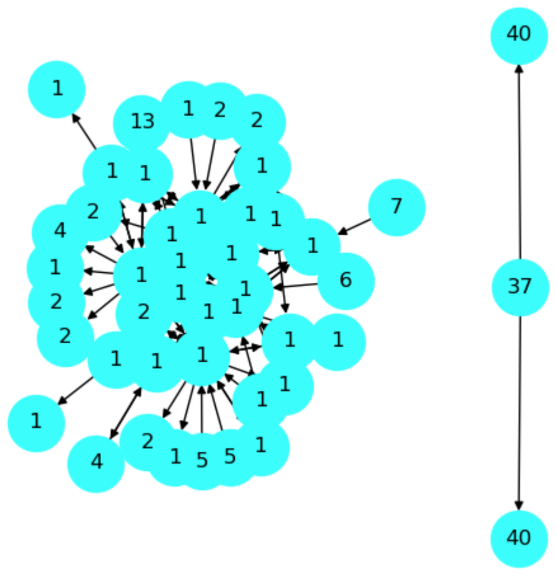


Figure 2.12: Candidate count for each node in the commercial airline template after applying the node-level statistics, topology and repeated-sets filters.

We construct a template that has 37 nodes and 210 edges by taking the induced subgraph of 20 core nodes and introducing some additional peripheral nodes and edges. When creating the template, we also ensure that it has nodes and edges across all channels to test the multiplex 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 2.12. The node-level statistics filter alone manages to find exact matches for 10 template nodes, and the topology filter increases the number of exact matches to 28 nodes. Applying Algorithm 7, we find that there are 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 identities of 28 out of 37 template nodes, as well as the potential identities of the remaining 9 template nodes.

2.3.4 Adversarial Activity

We apply our filtering methods to a series of datasets developed for the Defense Advanced Research Projects Agency program on Modeling Adversarial Activity (DARPA–MAA)[XTL18, DZK18a]. This program uses networks to represent activities (e.g. human trafficking, financial transactions, email communication, phone calls, and distribution of narcotics) some of which may be adversarial. 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 patterns from an activity template to part of a larger dataset of multiplex actions. Here, we present results for 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 world graphs ranging between 1,000–200,000 nodes with one or more roughly 100-node templates that simulate a scenario of activities by a specific group of agents.

Table 2.6 summarizes the sizes and filtering results for 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 signals, we solve the SICP or MCSP. For all of the datasets, we have applied the node-level statistics and topology filters to reduce the number of candidates. When the candidate counts remain high, we apply the neighborhood and elimination filters to further narrow them down.

2.3.4.1 PNNL Version 6

PNNL Version 6 [CPM18] has 12 instances, each consisting of one world and two templates. The worlds have around 23,000 nodes and over 12,000,000 edges each. The templates have 74–81 nodes and 1,200–1,650 edges.

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 2.6: 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 states the types of problems stated in Subsection 1.2.1 that we solve for each instance. The names of the filters are abbreviated: L = Node Label; S = Node-level Statistics; T = Topology; R = Repeated-Sets; N = Neighborhood; E = Elimination.

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 2.6) where we cannot solve the MCSP by applying the previous filters. Under such circumstances, we also apply the elimination filter, which narrows down the candidate counts further as shown in Figure 2.13.

Figure 2.13 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 5,000 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 2.14 shows the comparison of filtering results for instance B1-S1 before and after applying the elimination filter. The sharp contrast in their candidate counts shows that the elimination filter sometimes reduces the number of candidates significantly. After applying all filters, we observe in Figure 2.14 that some of the template nodes are permutable and share the same candidates, which lead to 1,152 SIs in this dataset.

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 for 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.

2.3.4.2 PNNL Real World

This dataset was created by PNNL from a social-media dataset collected by Matteo Magnani and Luca Rossi [CDM10, MR11]. 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 applying validation

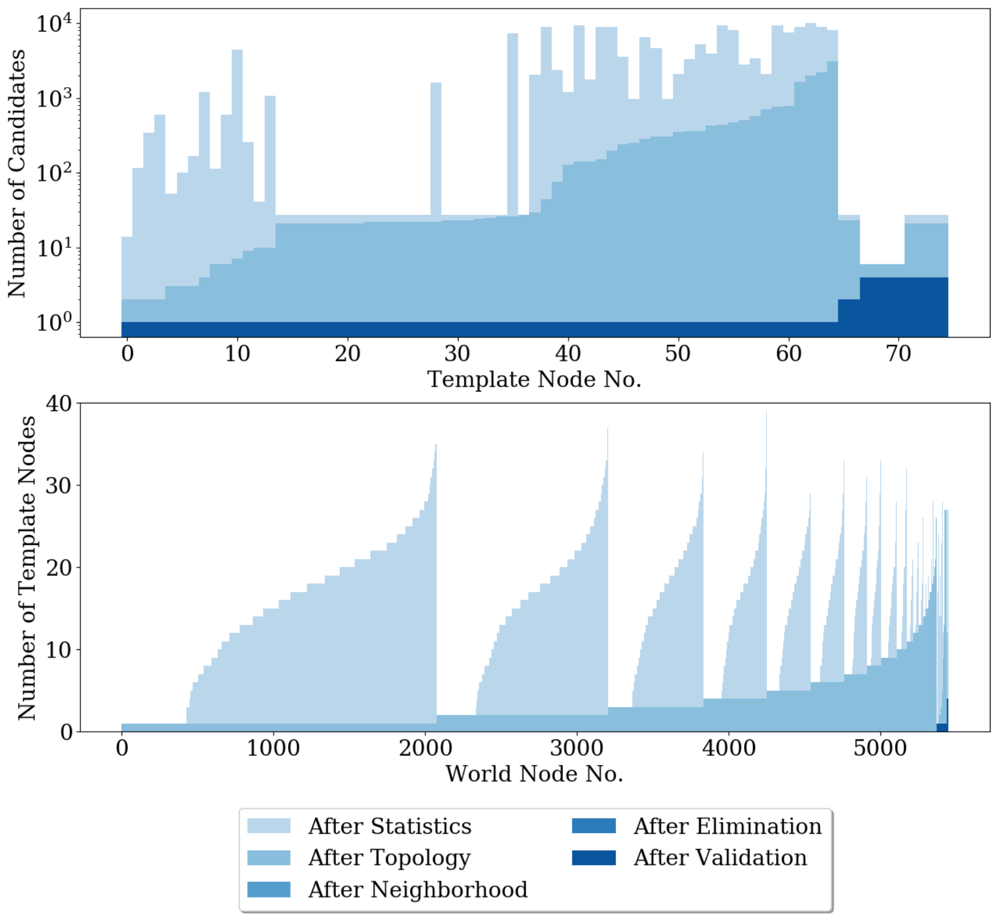
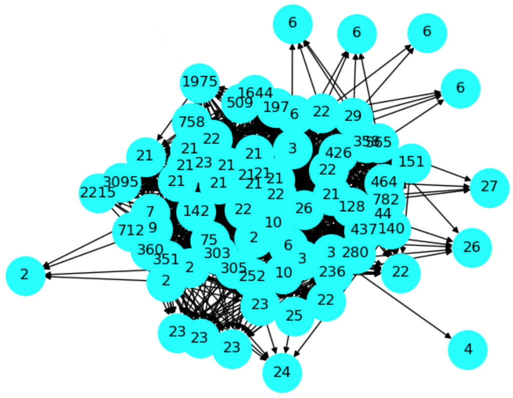


Figure 2.13: (Top): The number of candidates for each template node after different filters are applied to PNNL Version 6 B1-S1. (Bottom): The number of template nodes for which each world node is a candidate. Note that the Validation histogram perfectly overlaps the Elimination histogram and the Neighborhood histogram perfectly overlaps the Topology histogram.


(a) Without elimination filter.

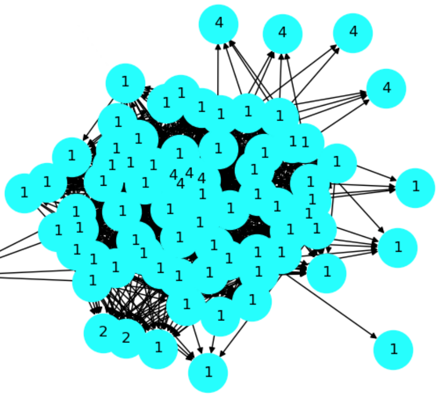

(b) With elimination filter.

Figure 2.14: 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 applying the elimination filter.

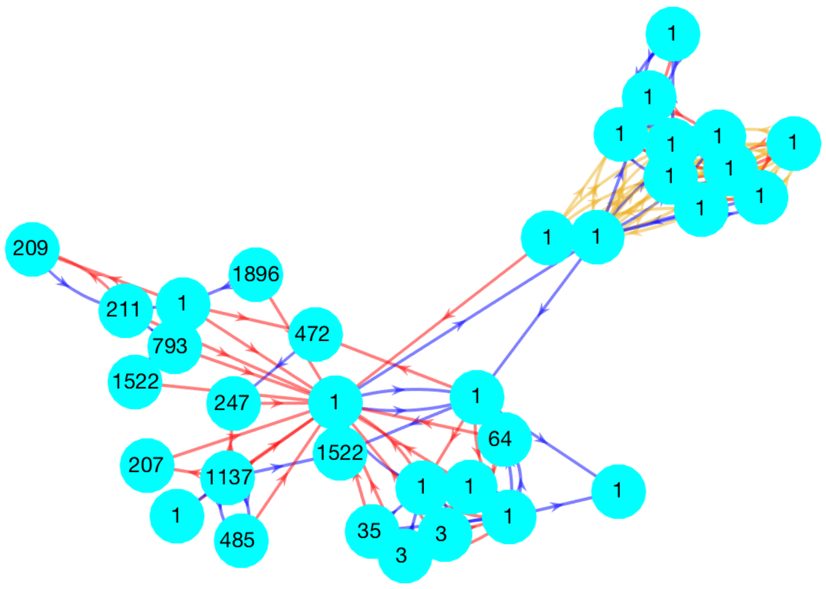


Figure 2.15: Candidate count for each node in the PNNL Real World template after solving the MCSP using validation. Edge colors correspond to channels.

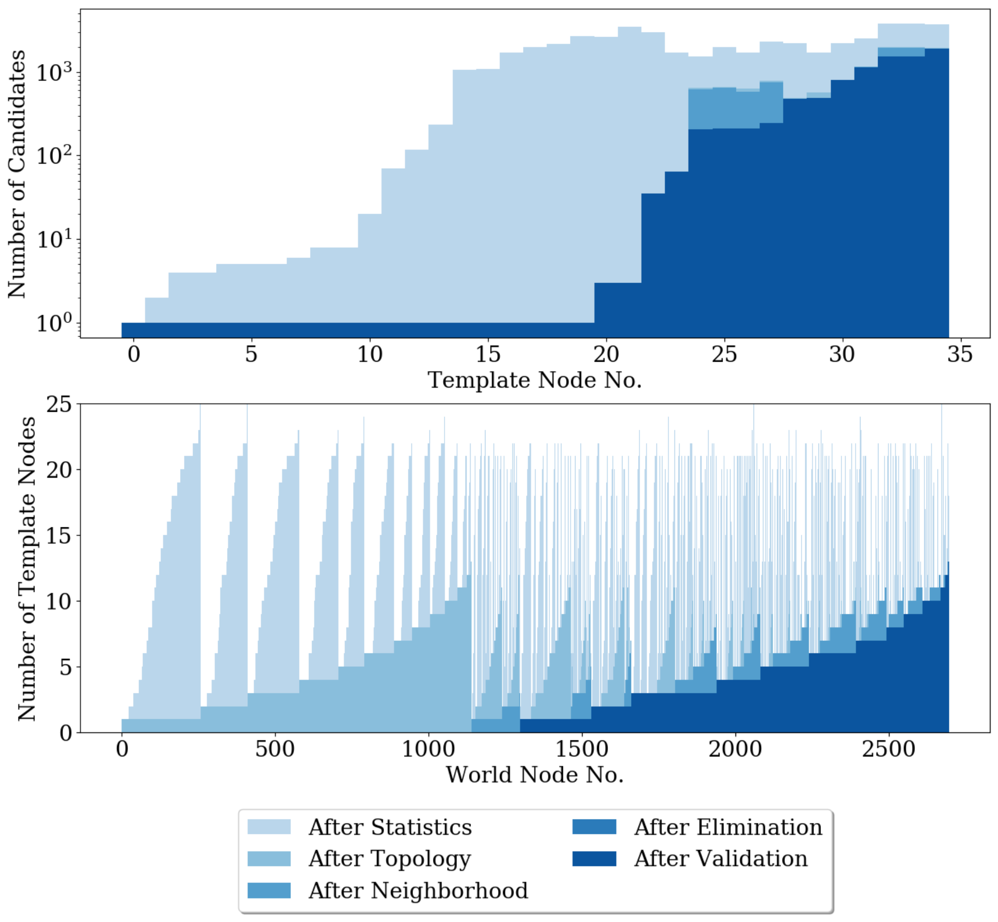


Figure 2.16: (Top): The number of candidates for each template node after different filters are applied to PNNL Real World. (Bottom): The number of template nodes for which each world node is a candidate. Note that the Validation histogram almost perfectly overlaps the Elimination histogram.

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 2.16 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 nearly identical to 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.

2.3.4.3 GORDIAN Version 7

GORDIAN Version 7 [KSG18] has two instances: Batch-1 and Batch-2. The Batch-1 template has 156 nodes and 3,045 edges, while the Batch-2 template has 44 nodes and 715 edges. Both instances have worlds with 190,869 nodes and about 123,265,000 edges. 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 elimination and neighborhood filters, we manage to find the exact match for 129 template nodes out of 156. In Figure 2.17, 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, and 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 also 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.

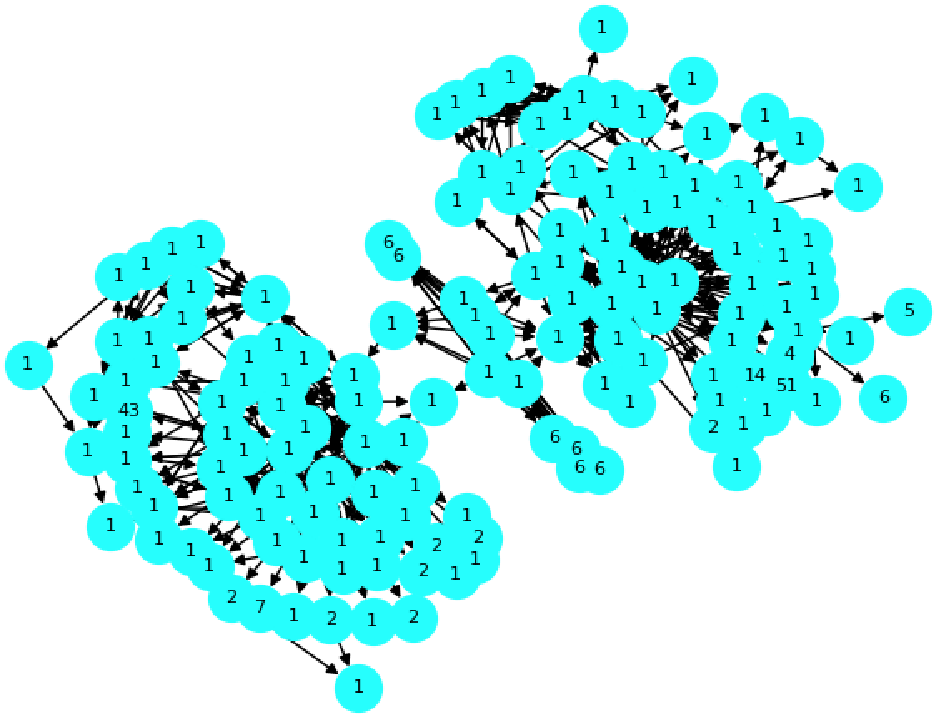


Figure 2.17: Candidate count for each node in the GORDIAN Version 7 Batch-1 template after applying the node-level statistics, topology, repeated-sets, and elimination filters.

2.3.4.4 IvySys Version 7

IvySys Version 7 [BJU18] has three channels corresponding to financial, communication and logistics transactions. Due to the sparsity of the template relative to that of the world, we are unable to identify any template nodes with unique candidates after applying filters, as seen in Figure 2.18. 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. However, the complexity of the solution space makes it unreasonable to proceed to solve the rest of the variants, like SICP, SNSP and MCSP.

2.3.4.5 IvySys Version 11

IvySys Version 11 consists of a single world graph and template, each with five channels. It is somewhat similar to IvySys Version 7, as illustrated in Figure 2.19. The template has 103 nodes and 387 edges, while the world has 1,404 nodes and 5,719,030 edges. 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 2.19 shows the existence of permutable sets of template nodes. This suggests the possibility that we might have already reduced the candidates to the point where almost all of the remaining candidates are part of at least one valid signal.

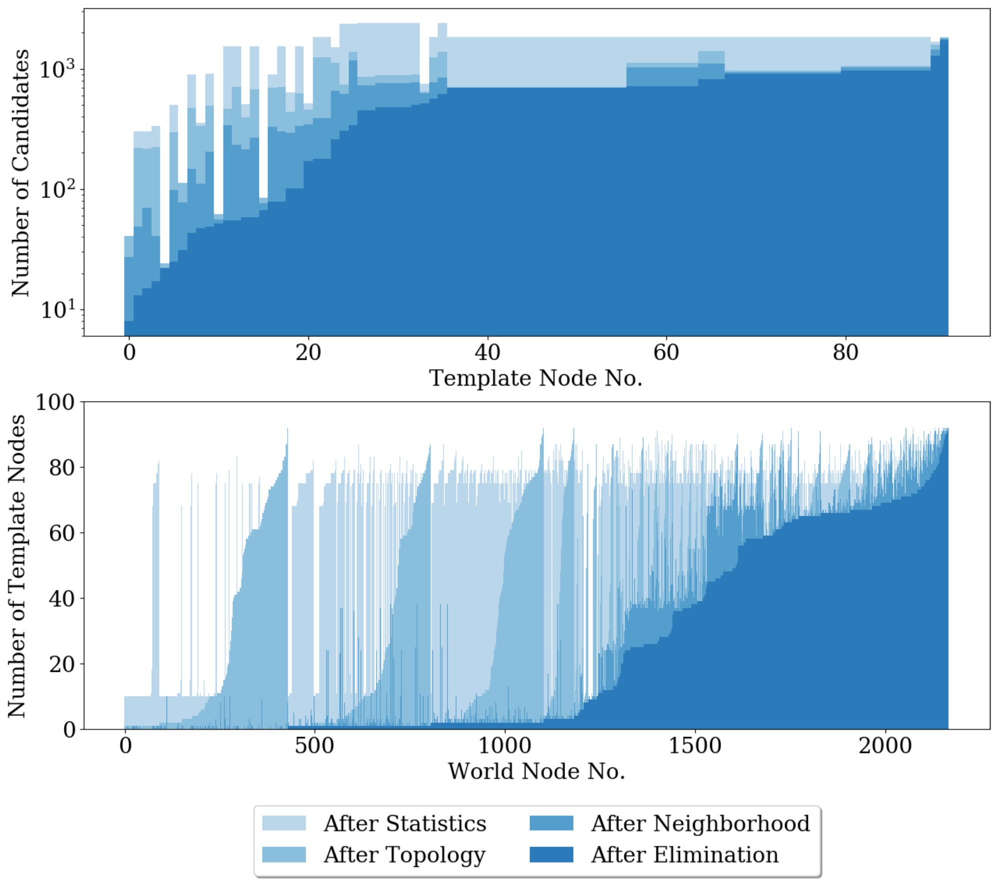


Figure 2.18: (Top): The number of candidates for each template node after different filters are applied to IvySys Version 7. (Bottom): The number of template nodes for which each world node is a candidate.

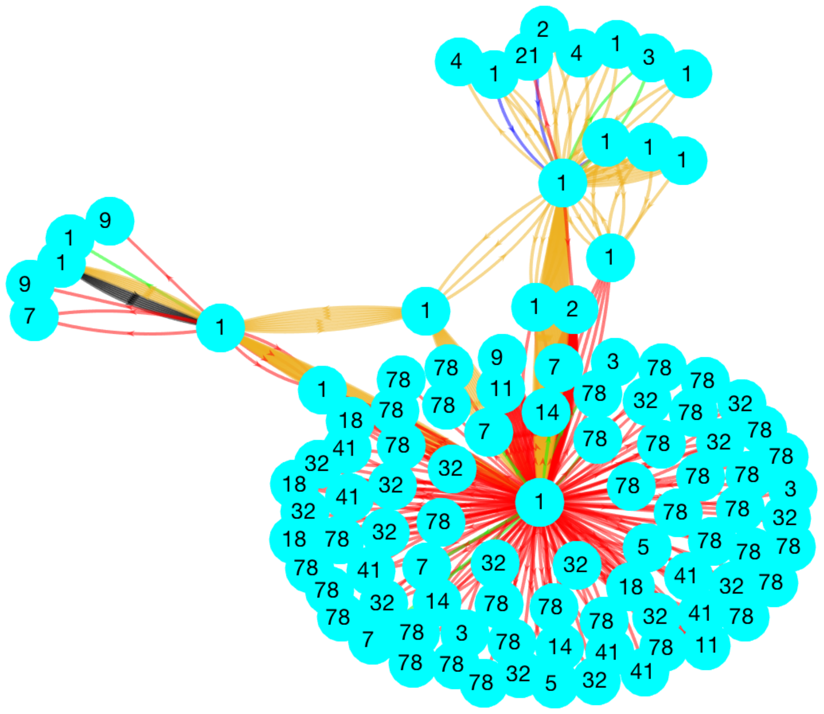


Figure 2.19: Candidate count for each node in the IvySys Version 11 template after applying the node-level statistics, topology, repeated-sets, and elimination filters. The colors of the edges correspond to their different channels.

CHAPTER 3

Inexact Attributed Subgraph Matching[*]

3.1 Introduction

In applications where the graphs have attributes on the nodes and edges, it is common to search for subgraphs of the world that only approximately match the template. This is called *inexact attributed subgraph matching* (also known as approximate attributed subgraph matching[PTT14]).

Definition 3.1.1 (Attributed Graph). *An* attributed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{L}, \mathcal{A})$ *consists of a set $\mathcal{V}$ of nodes, a set $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ of edges, a set $\mathcal{A}$ of attributes, and a map $\mathcal{L} : \mathcal{V} \cup \mathcal{E} \to \mathcal{A}$ from nodes and edges to their attributes.*

Given two attributed graphs, template $\mathcal{G}_t = (\mathcal{V}_t, \mathcal{E}_t, \mathcal{L}_t, \mathcal{A})$ and world $\mathcal{G}_w = (\mathcal{V}_w, \mathcal{E}_w, \mathcal{L}_w, \mathcal{A})$, we are interested in one-to-one maps $f : \mathcal{V}_t \to \mathcal{V}_w$ where the induced subgraph of the image is similar to the template. Such a map f is called an *inexact match* of $\mathcal{G}_t$. The cost function used to measure similarity between the template and the image of f is denoted $C(f; \mathcal{G}_t, \mathcal{G}_w)$ and is described in Subsection 3.2.1.

To parallel the definitions in Subsection 1.2.1, we call the process of finding the map $f^\star$ that minimizes $C(f; \mathcal{G}_t, \mathcal{G}_w)$ inexact *attributed subgraph isomorphism (ASI)*, and we call the process of finding all maps f with distance at most ϵ inexact *attributed subgraph matching (ASM)*. We call the problem of finding the k most similar maps is called *top-k inexact ASM*.

[*]This chapter is adapted from [TMY20].

In the remainder of this section, we discuss existing approaches to ASI/ASM and inexact ASI/ASM, as well as our contributions. In Section 3.2, we describe our approach to solving inexact ASI, inexact ASM, and top-k inexact ASM. In Section 3.3, we evaluate our methods on the AIDA V2.1.2 dataset, which consists of knowledge graphs collected from news articles about political events in the Ukraine.

3.1.1 Related Work

Section 1.3 discusses algorithms for exact subgraph matching. Algorithms for inexact subgraph matching are more diverse than those for exact subgraph matching. Different applications and research areas define the inexact subgraph matching problem in different ways. Some algorithms take tree search, constraint propagation, or graph indexing approaches similar to those in exact subgraph matching [PTT14, JHW19, KX19, LZ17], while other algorithms relax the discrete optimization problem into a continuous one in order to apply traditional continuous optimization techniques such as gradient descent [SPP20].

3.1.2 Our Contributions

We introduce algorithms for inexact attributed subgraph isomorphism and matching to find optimal subgraphs as measured by graph edit distance. We show results for noisy queries on the AIDA version 2.1.2 knowledge graph dataset collected from news articles about political events in the Ukraine, as well as the DBPedia knowledge graph.

3.2 Algorithm

In Subsection 3.2.1, we define the cost function $C(f; \mathcal{G}_t, \mathcal{G}_w)$ that will be minimized in inexact ASI/ASM. In Subsections 3.2.2 and 3.2.3, we explain how to compute lower bounds on $C(f; \mathcal{G}_t, \mathcal{G}_w)$ under the constraint $f(t_c) = w_c$ for each $t_c \in \mathcal{V}_t$ and $w_c \in \mathcal{V}_w$. Using these constrained lower bounds, in Subsection 3.2.4 we describe a tree search procedure for inexact

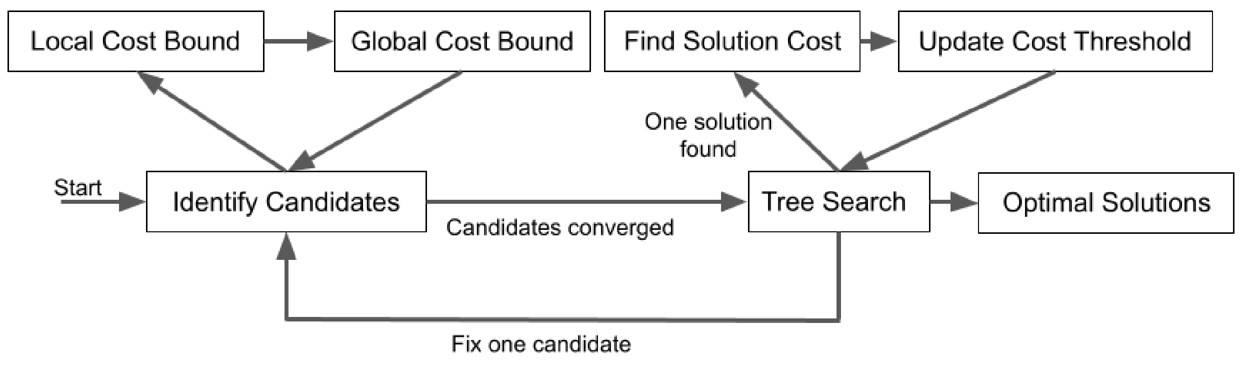


Figure 3.1: Flowchart of the inexact subgraph matching procedure.

ASI/ASM. See Figure 3.1 for a flowchart of this process.

3.2.1 Graph Edit Distance Based Cost Function

The graph edit distance [SF83, GXT10] is a measure of the distance between two graphs, as defined by the total cost of the cheapest sequence of edits that transform one graph into another. The six possible edits usually considered are node addition, node deletion, node substitution, edge addition, edge deletion, and edge substitution. Here, we consider a cost function using only the latter four edits.

The node substitution costs are measured by a user-defined function $D_{\mathcal{V}} : (\mathcal{V}_t \times \mathcal{V}_w) \to \mathbb{R}^+$ that typically compares the labels of the nodes. A common node substitution function is

$$D_{\mathcal{V}}(v_t, v_w) = \mathbb{1}\left[\mathcal{L}_t(v_t) \neq \mathcal{L}_w(v_w)\right],$$

so that $\sum_{t \in \mathcal{V}_t} D_{\mathcal{V}}(t, f(t))$ counts the number of node assignments that do not preserve the node label. If the node labels belong to a normed vector space, then the corresponding norm can be used, i.e.

$$D_{\mathcal{V}}(v_t, v_w) = \|\mathcal{L}_t(v_t) - \mathcal{L}_w(v_w)\|.$$

Likewise, the edge substitution, addition and deletion costs are measured by user-defined functions $D : (\mathcal{E}_t \times \mathcal{E}_w) \to \mathbb{R}^+$, $D^+ : \mathcal{E}_w \to \mathbb{R}^+$ and $D^- : \mathcal{E}_t \to \mathbb{R}^+$. The edge related cost functions are combined into a single function $D_{\mathcal{E}}$ for brevity

$$D_{\mathcal{E}}(e, f(e)) = \begin{cases} D(e, f(e)), & e \in \mathcal{E}_t, f(e) \in \mathcal{E}_w, \\ D^-(e), & e \in \mathcal{E}_t, f(e) \notin \mathcal{E}_w, \\ D^+(f(e)), & e \notin \mathcal{E}_t, f(e) \in \mathcal{E}_w, \\ 0, & e \notin \mathcal{E}_t, f(e) \notin \mathcal{E}_w. \end{cases} \tag{3.1}$$

For convenience, we use the shorthand $f((t_1, t_2)) = (f(t_1), f(t_2))$ so that $f(e)$ is well-defined. Given a template $\mathcal{G}_t$, a world $\mathcal{G}_w$, and a map $f : \mathcal{V}_t \to \mathcal{V}_w$, the cost function $C(f; \mathcal{G}_t, \mathcal{G}_w)$ is defined as

$$C(f; \mathcal{G}_t, \mathcal{G}_w) = \sum_{t \in \mathcal{V}_t} D_{\mathcal{V}}(t, f(t)) + \sum_{e \in \mathcal{V}_t \times \mathcal{V}_t} D_{\mathcal{E}}(e, f(e)). \tag{3.2}$$

3.2.2 Cost Bounds

We now discuss lower bounds on the cost function. The cost function $C(f; \mathcal{G}_t, \mathcal{G}_w)$ can be decomposed as

$$C(f; \mathcal{G}_t, \mathcal{G}_w) = \sum_{t \in \mathcal{V}_t} L(t, f(t); \mathcal{G}_t, \mathcal{G}_w, f), \tag{3.3}$$

where $L(t, f(t); \mathcal{G}_t, \mathcal{G}_w, f)$ are local costs related to each template node t and matching world node $f(t)$. We use decompositions where the local cost L has the form

$$L(t, w; \mathcal{G}_t, \mathcal{G}_w, f) = D_{\mathcal{V}}(t, w) + \frac{1}{2} \sum_{t_o \in \mathcal{V}_t} \Big[D_{\mathcal{E}}\big((t, t_o), (w, f(t_o))\big) + D_{\mathcal{E}}\big((t_o, t), (f(t_o), w)\big) \Big]. \tag{3.4}$$

This way, the local cost $L(t, w; \mathcal{G}_t, \mathcal{G}_w, f)$ captures the cost incurred by assigning world node w to template node t and half of the cost of assigning the edges incident to w to those incident to t. Edge costs are halved[†] so that the local costs sum to the correct total cost $C(f; \mathcal{G}_t, \mathcal{G}_w)$;

[†]Here, for simplicity, we halve the cost, but any non-negative weights that sum to one will work. By adjusting the weights for each edge, we can place more importance on certain nodes; we address this in Section 3.A.

otherwise, edge costs would be counted twice (once for each endpoint). If the edge addition cost were neglected (i.e. $D^+ \equiv 0$), the summation $\sum_{t_o \in \mathcal{V}_t}$ reduces to $\sum_{t_o \in \mathcal{N}_t}$, where $\mathcal{N}_t$ is the set of neighbors of t.

To lower bound the full cost defined in Equation (3.2), we start by bounding the local cost $L(t, w; \mathcal{G}_t, \mathcal{G}_w, f)$ from below by $B(t, w; \mathcal{G}_t, \mathcal{G}_w)$ defined as follows

$$\begin{aligned} B(t, w; \mathcal{G}_t, \mathcal{G}_w) &:= D_{\mathcal{V}}(t, w) + \frac{1}{2} \sum_{t_o \in \mathcal{V}_t} \min_{w_o \in \mathcal{V}_w} \Big(D_{\mathcal{E}}\big((t, t_o), (w, w_o)\big) + D_{\mathcal{E}}\big((t_o, t), (w_o, w)\big) \Big) \\ &\leq L(t, w; \mathcal{G}_t, \mathcal{G}_w, f). \end{aligned} \tag{3.5}$$

Note that the local bound $B(t, w; \mathcal{G}_t, \mathcal{G}_w)$ has no dependence on the map f. In the context of inexact ASI/ASM, the set $\mathcal{V}_w$ iterated over in the minimization can often be reduced for practical purposes; see Section 3.B for the details. We can extend the bound on the local cost to a naive bound on the full cost:

$$\begin{aligned} G_{\text{Naive}}(\mathcal{G}_t, \mathcal{G}_w) &:= \sum_{t \in \mathcal{V}_t} \min_{w \in \mathcal{V}_w} B(t, w; \mathcal{G}_t, \mathcal{G}_w) \\ &\leq \sum_{t \in \mathcal{V}_t} B(t, f(t); \mathcal{G}_t, \mathcal{G}_w) \\ &\overset{\text{Eqn. (3.4)}}{\leq} \sum_{t \in \mathcal{V}_t} L(t, f(t); \mathcal{G}_t, \mathcal{G}_w, f) = C(f; \mathcal{G}_t, \mathcal{G}_w). \end{aligned} \tag{3.6}$$

By leveraging the fact that f must be one-to-one (1-1), we can compute a tighter lower bound:

$$\begin{aligned} G_{\text{LAP}}(\mathcal{G}_t, \mathcal{G}_w) &:= \min_{\substack{g: \mathcal{V}_t \to \mathcal{V}_w \\ \text{s.t. } g \text{ is 1-1}}} \sum_{t \in \mathcal{V}_t} B(t, g(t); \mathcal{G}_t, \mathcal{G}_w) \\ &\leq \sum_{t \in \mathcal{V}_t} B(t, f(t); \mathcal{G}_t, \mathcal{G}_w) \\ &\overset{\text{Eqn. (3.4)}}{\leq} \sum_{t \in \mathcal{V}_t} L(t, f(t); \mathcal{G}_t, \mathcal{G}_w, f) = C(f; \mathcal{G}_t, \mathcal{G}_w). \end{aligned} \tag{3.7}$$

We refer to G_{Naive} and G_{LAP} as “global cost bounds”.

Note that while G_{LAP} is tighter than G_{Naive}, it is more expensive to compute. Computing the global cost bound G_{LAP} is equivalent to solving a rectangular linear assignment problem

(LAP) of size $|\mathcal{V}_t| \times |\mathcal{V}_w|$. Many algorithms exist to solve the LAP and its rectangular variant [Bc99]. The most notable algorithms include the Hungarian algorithm [Kuh55], the Munkres algorithm [Mun57], the Jonker–Volgenant (JV) algorithm [JV87], and the auction algorithm [Ber88]. In our implementation, we apply a modified JV algorithm [Cro16] that is designed to efficiently solve rectangular LAPs. The time complexity of the solver is $O(|\mathcal{V}_t|\,|\mathcal{V}_w|^2)$. For contrast, the time required to compute the naive global cost bound G_{Naive} is only $O(|\mathcal{V}_t|\,|\mathcal{V}_w|)$.

3.2.3 Constrained Cost Bounds

To use the global cost bounds from Equations (3.6) and (3.7) in our algorithm, we must compute the bounds under the constraint $f(t_c) = w_c$ for each $t_c \in \mathcal{V}_t$ and each $w_c \in \mathcal{V}_w$. These constrained global cost bounds will be used as a heuristic for performing a tree search described in Subsection 3.2.4. The corresponding constrained global cost bounds are

$$G^c_{\text{Naive}}(t_c, w_c; \mathcal{G}_t, \mathcal{G}_w) := B(t_c, w_c; \mathcal{G}_t, \mathcal{G}_w) + \sum_{\substack{t \in \mathcal{V}_t \\ t \neq t_c}} \min_{\substack{w \in \mathcal{V}_w \\ w \neq w_c}} B(t, w; \mathcal{G}_t, \mathcal{G}_w) \tag{3.8}$$

for the naive global cost bound and

$$G^c_{\text{LAP}}(t_c, w_c; \mathcal{G}_t, \mathcal{G}_w) := \min_{\substack{g: \mathcal{V}_t \to \mathcal{V}_w \\ \text{s.t. } g \text{ is 1-1} \\ \text{s.t. } g(t_c) = w_c}} \sum_{\substack{t \in \mathcal{V}_t \\ t \neq t_c}} B(t, f(t); \mathcal{G}_t, \mathcal{G}_w) \tag{3.9}$$

for the LAP-based global cost bound.

To compute these constrained global cost bounds, the first step in either case is to compute local bounds $B(t, w; \mathcal{G}_t, \mathcal{G}_w)$ for each $t \in \mathcal{V}_t$ and $w \in \mathcal{V}_w$. This takes $O\left(|\mathcal{V}_t|\,|\mathcal{V}_w| \operatorname{avgtime}(B)\right)$ time, where $\operatorname{avgtime}(B)$ denotes the mean time to compute the local bound for a fixed pair of nodes $t, w \in \mathcal{V}_t \times \mathcal{V}_w$. The remaining calculations to compute Equation (3.8) for each $t_c \in \mathcal{V}_t$ and $w_c \in \mathcal{V}_w$ have time complexity $O(|\mathcal{V}_t|\,|\mathcal{V}_w|)$.

To compute Equation (3.9), we must solve a LAP of size $(|\mathcal{V}_t| - 1) \times (|\mathcal{V}_w| - 1)$ for each $t_c \in \mathcal{V}_t$ and $w_c \in \mathcal{V}_w$. Solving each of these LAPs separately would require $|\mathcal{V}_t|\,|\mathcal{V}_w|$ applications of the $O(|\mathcal{V}_t|\,|\mathcal{V}_w|^2)$ LAP solver for a time complexity of $O(|\mathcal{V}_t|^2\,|\mathcal{V}_w|^3)$. However, since each LAP is a constrained version of the same unconstrained LAP in Equation (3.7), we can instead compute these as perturbations of the unconstrained solution.

To start, we solve the unconstrained LAP in Equation (3.7) to find the map

$$g_\star = \mathop{\arg\min}_{\substack{g:\mathcal{V}_t \to \mathcal{V}_w \\ \text{s.t. } g \text{ is 1-1}}} \sum_{t \in \mathcal{V}_t} B(t, g(t); \mathcal{G}_t, \mathcal{G}_w)$$

in $O(|\mathcal{V}_t|\,|\mathcal{V}_w|^2)$ time. Next, we use the dynamic Hungarian algorithm [KSD07] to enforce the constraint $g(t_c) = w_c$ on the unconstrained map $g_\star$ for each $t_c \in \mathcal{V}_t$ and $w_c \in \mathcal{V}_w$. Enforcing the constraint $g(t_c) = w_c$ frees up $g_\star(t_c)$ to potentially be reassigned to some other template node. When w_c is not in the image of $g_\star$, reassigning $g_\star(t_c)$ costs $O(|\mathcal{V}_t|^2)$ for each $t_c \in \mathcal{V}_t$, independent of w_c, for a total of $O(|\mathcal{V}_t|^3)$. When w_c is in the image of $g_\star$, the constraint $g(t_c) = w_c$ also frees up $g_\star^{-1}(w_c)$ to be reassigned to some other world node.

3.2.4 Searching for Optimal Solutions

From Subsection 3.2.3, we have a procedure for computing lower bounds on $C(f; \mathcal{G}_t, \mathcal{G}_w)$ under the constraint $f(t_c) = w_c$ for each $t_c \in \mathcal{V}_t$ and $w_c \in \mathcal{V}_w$. We treat these lower bounds as a heuristic for performing a greedy depth-first search. For each template node t, we assign $f(t) = w$ for the candidate w with the lowest bound, then recompute the bounds under that additional assignment. We assign candidates to template nodes with the fewest minimum bound candidates first, as this indicates which template nodes have only a few "good" choices. After assigning all template nodes in this way, we obtain a map f.

Although f is usually not the optimal map $f^\star$ that we seek, it can be used to drastically cut down on the list of possible assignments we have to consider going forward. We compute the cost of f to serve as an upper bound on the cost of the optimal map $f^\star$. We keep track of the *cost threshold* U and set it equal to the smallest cost $C_{\min}$ we have seen so far, with $f_{\min}$ the corresponding map. Using $C_{\min}$, the search space is refined by skipping assignments $f(t) = w$ that lead to lower bounds that are greater than or equal to $C_{\min}$, since in inexact ASI we are only interested in the map $f^\star$ that minimizes the cost.

After identifying a new map or eliminating all remaining possibilities due to the cost bounds, we backtrack and try assigning different world nodes to some template nodes. This is done until there are no options left to explore, at which point we have found the optimal map $f^\star = f_{\min}$.

To perform inexact ASM instead of inexact ASI, one can fix $U \equiv \epsilon$ and record all of the maps f that are observed during the search. In this context, we only skip assignments when their cost bound is strictly greater than ϵ, so that we do not accidentally skip matches whose cost is exactly ϵ.

To perform top-k inexact ASM, keep track of $f_1, \ldots, f_k$ and $C_1, \ldots, C_k$ that track the k best maps seen so far and their costs. Use C_k instead of $C_{\min}$ to prune the search space, and only prune when the cost bound is greater than or equal to C_k. When the search is completed, $f_1, \ldots, f_k$ are the k maps with the lowest costs.

One way to speed up the search in inexact ASI and top-k inexact ASM is to set an initial value for U. However, this approach can lead to no solutions being found if the chosen value is lower than the cost of the optimal solution $C(f^\star; \mathcal{G}_t, \mathcal{G}_w)$.

3.3 Experiments

We evaluate our algorithms on two knowledge graph datasets, AIDA V2.1.2 (Subsection 3.3.1) and DBPedia (Subsection 3.3.2). Each dataset consists of a world graph, one or more templates, and an associated graph edit distance. For each dataset and template, we use our algorithms to identify the five cheapest matches (top-k inexact ASM). For AIDA V2.1.2, we also use inexact ASM with thresholding to identify all matches whose cost equal to the lowest possible cost. We run all experiments on an HP Z8 G4 Workstation with two 12-core Intel® Xeon® Gold 6136 CPUs.

3.3.1 AIDA Version 2.1.2

The AIDA Version 2.1.2 dataset was created by Pacific Northwest National Laboratory (PNNL) for the DARPA–MAA program. This dataset consists of a knowledge graph collected from news articles about political events in the Ukraine. Provided also is a measure of distance between attributes, which can be used to construct the corresponding graph edit distance. The world graph has 98,817 nodes and 138,127 edges. Three templates are provided, each with six different variations labeled A–F. These templates are much smaller than the world graph, consisting of 11–33 nodes and 11–40 edges. These templates were created by taking a known "ground truth" subgraph and adding increasing levels of noise. The A template has no noise and is guaranteed to have at least one exact match. Provided also is the ground truth mapping for the A version of each template.

We perform top-k inexact ASM on these templates, with $k = 5$. We use two approaches: the first approach, labeled "Normal" in the table, simply runs the algorithm with no initial cost threshold, while the second uses the ground truth cost bound (labeled "GTCB") from the A template to set an appropriate initial cost threshold for the other templates in each series (1,2,3). To identify this threshold, we first impose the matching from the ground truth, then set our algorithm to find optimal assignments for any remaining nodes that were not included in the ground truth. Using only this initial cost threshold, we discard all other matching information from the ground truth and proceed to optimize over the space of matches with lower cost than the cost of the ground truth. The cost of the ground truth for each template is listed in the "Ground Truth" column under "Cost". In real world contexts, the ground truth is unknown. However, it is not unreasonable that another, possibly suboptimal, approximate match exists that we can use for the purpose of setting the initial cost threshold to restricting the solution space.

The results of the two approaches on the AIDA Version 2.1.2 dataset are shown in Table 3.1. The algorithm was cut off if it failed to complete within 46.5 hours, taking the best match that it had found so far. Although we perform top-k matching, we list only one

Table 3.1: Results for the AIDA Version 2.1.2 dataset, showing time taken and the cost of the best match that was found. The algorithm was cut off if it failed to complete within 46.5 hours, taking the best match that it had found so far.

	Time		Cost		
Template	Normal	GTCB	Normal	Ground Truth	GTCB
1A	27.8 min	1.5 sec	0.000	0.000	0.000
1B	24.5 min	0.2 sec	0.347	0.351	0.347
1C	46.5 hrs	5.48 hrs	8.783	3.610	2.254
1D	46.5 hrs	64.1 min	15.470	1.639	1.636
1E	46.5 hrs	35.9 hrs	16.194	2.642	2.638
1F	46.5 hrs	76.7 min	13.596	2.328	1.772
2A	2.25 min	0.06 sec	0.000	0.000	0.000
2B	3.07 min	0.15 sec	0.307	0.335	0.307
2C	6.35 min	10.6 sec	4.070	4.891	4.070
2D	12.4 min	32.9 sec	3.839	5.942	3.839
2E	4.58 min	22.4 sec	3.848	4.533	3.848
2F	18.5 min	42.3 sec	4.012	4.704	4.012
3A	2.33 min	0.08 sec	0.000	0.000	0.000
3B	2.27 sec	0.15 sec	0.145	0.191	0.145
3C	4.10 min	3.56 sec	2.452	2.452	2.452
3D	10.3 min	2.48 min	2.312	2.585	2.312
3E	20.8 min	10.9 min	2.472	3.471	2.472
3F	4.80 min	0.94 sec	1.314	1.348	1.314

cost for each template; this is because for all templates considered, all 5 matches found were of the same cost. We observe that for all of the A templates, both the normal and GTCB approaches converge to the expected result of an exact match with 0 graph edit distance. For all templates other than 1C–F, both approaches were able to complete within the limit of 46.5 hours and arrive at the optimal solutions. For 1C–F, the normal version hit the runtime limit, yielding the best solutions found within that time. For the GTCB version, only template 1E was aborted early, after 35.9 hours. This was done because, when examining the branching structure of the search algorithm, we deemed the approach unlikely to complete. The other three templates completed and reached optimal solutions.

After identifying optimal solutions (for all templates except 1E), we then apply the approach for inexact ASM discussed in Subsection 3.2.4 to find all optimal solutions by setting the cost threshold to the cost of an optimal solution. The number of optimal solutions found is listed in Table 3.2. In the case of 1E, we set this cost threshold to be the cost of the best solution found; we observe that all found solutions had the same cost.

For templates 1D, 1E, and 1F, the optimal solution search was cut off after 209 hours due to time and memory limitations. We believe that there exist more solutions than those that were found, possibly orders of magnitude more. We have marked these templates with a "$>$" in Table 3.2 to indicate this.

3.3.1.1 Analysis of the Solution Space

It is important to understand the solution space beyond simply finding one match, especially in real-world applications. For example, in security applications, the match may point to specific people or places, some of which could be imposters. Here, we show an example of a map from the template to the full solution space. This visualization can be helpful in trying to understand the connectivity of the template nodes in the full solution space. After using the methods discussed in prior sections to discover subgraph matchings of minimal cost, we can use symmetries in the template graph and world graph to compactly represent the entire solution space.

Symmetry in graph structures has been studied in depth in the context of subgraph matching [HLL13, RW15, BCL16, NYG19]. It is a confounding factor for subgraph discovery that can lead to redundant work while exploring symmetric areas of the graph. The kind of symmetry most often utilized is where nodes are considered *structurally equivalent* if they have the same label and are connected to the same neighbors by edges with the same attributes [Sai79]. For an example of this type of symmetry, see Figure 3.2.

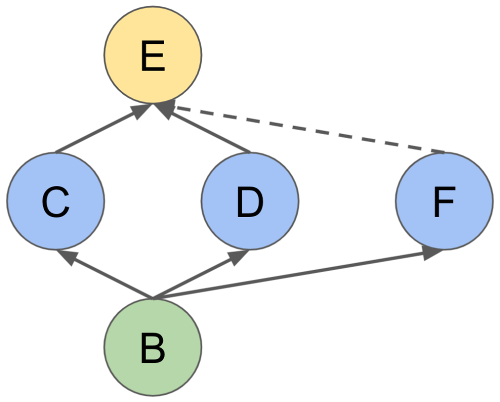


Figure 3.2: Structural Equivalence: Nodes C and D are structurally equivalent and F is structurally equivalent to neither. C, D, and F have the same node label and same set of neighbors. However, the edge connecting F to E has a different label than the edges connecting C and D to E, so F is not structurally equivalent to C or D.

In the matching problems on the AIDA dataset, symmetry in the graph accounts for the combinatorial explosion in solutions for certain template graphs, especially in instances where noise is present. Figure 3.3 demonstrates various forms of symmetry in the world graph and how they affect the structure of the solution space for Template 1B. We present the template graph and its matches to a subgraph of the world graph containing only those nodes appearing in an optimal solution. We color each node in the template graph the same as its candidates in the world graph. The blue and orange nodes in the world graph are two groups of structurally equivalent nodes and can be swapped out arbitrarily in any solution while maintaining a matching of the same cost. The red and lavender groups of world nodes present a slightly more complex form of symmetry (automorphic equivalence). Exploiting it to generate more solutions would require a more intelligent scheme that assigns the red and lavender template nodes as a unit. We leave considering automorphic equivalence for future work and only consider structural equivalence in this chapter. In this example, the problem has 6,120 optimal cost solutions; applying structural equivalence, we can reduce it to 2,520 solutions, from which we can generate the rest.

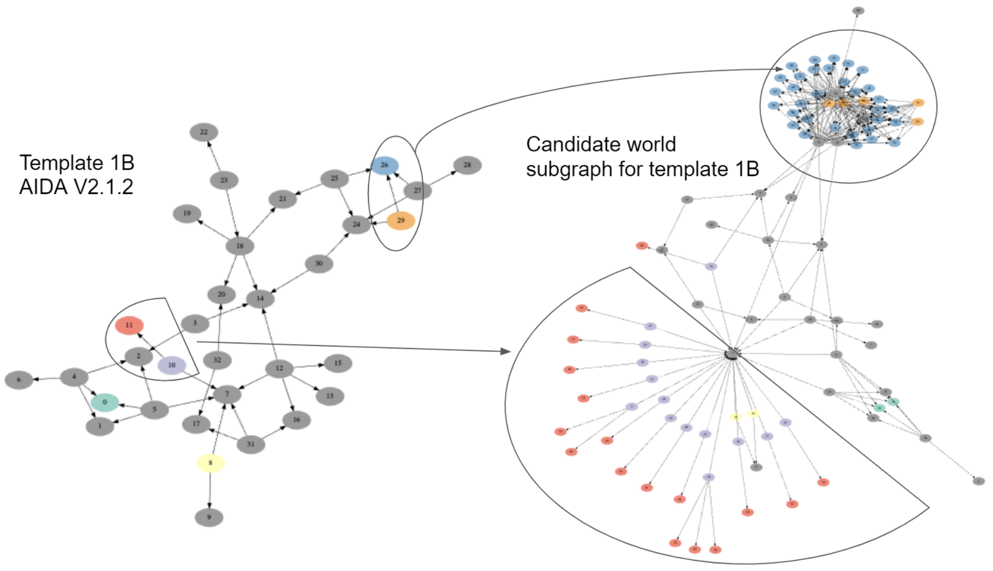


Figure 3.3: Template graph and world subgraph for Problem 1B. Colored sets of nodes in the world graph are candidates for nodes of the same color in the template graph. Two substructures of the template correspond to large sets of candidate world nodes.

An a posteriori analysis of the solutions generated by our search enables us to compress the solution space as well as to illustrate potential ways to significantly speed up the subgraph search. We take the subgraph of world nodes that appear in a minimal cost solution and compute sets of nodes that are structurally equivalent. Then to compress the solution space, we eliminate any solution that can be generated by swapping structurally equivalent world nodes in another mapping. We do this by examining the classes of solutions generated by interchanging structurally equivalent nodes and identifying a representative from each.

Table 3.2 demonstrates the extent to which structural equivalence reduces the description of the solution space to a smaller set. We also list the number of world nodes that appear in a minimal cost solution as well as the number of structural equivalence classes to show the level of equivalence in the solution space. If the number of structural equivalence classes is significantly smaller than the number of world nodes, there is a great deal of equivalence. We observe that in certain cases (e.g. templates 1E and 1F), the size of the representative solution set is nearly ten times smaller than the original. If an algorithm were to efficiently compute these symmetries and incorporate them in a subgraph search, then we might expect speedups of an order of magnitude for these problems.

Analysis through a symmetry lens exposes how introducing noise into a subgraph matching problem impacts the solution space. Broadly, if more information is known about the labels and neighbors of vertices in both the template and world, there will be less symmetry apparent in the matching problem. This can be seen in Table 3.2 with increased noise as we go from A to F, which significantly expands the solution space. Intuitively, having more label information allows nodes to distinguish themselves from each other and break symmetry. If we introduce noise into a problem, say by removing a template node's label, then the labels of the world graph become irrelevant since the label cost will be the same.

The graphs provided in the AIDA datasets have three different labels: "rdf:type", which indicates the semantic type of a node (e.g. Person, Location, Vehicle, etc.), "hasName", which gives the names of entities, and "textValue", which contains miscellaneous text information associated with the node. Table 3.3 lists the number of structural equivalence classes for the first set of templates when considering only the "rdf:type" label as compared to considering all labels. As can be seen, when considering only the type label, we have significantly fewer structural equivalence classes. This is especially apparent in template 1E and 1F, for which the minimal cost solution requires two edge mismatches. This leaves an isolated template node that may match any world graph node, as long as the label matches.

Table 3.2: The number of solutions, representative solutions, candidate world nodes, and structural equivalence classes for each subgraph matching problem from the AIDA Version 2.1.2 dataset. For templates 1D–F, the code was terminated due to runtime constraints before all solutions could be found.

	Template					
	1A	1B	1C	1D	1E	1F
# Solutions	6	6,120	324	>392k	>382k	>400k
# Rep. Sols.	3	2,520	162	>315k	>34k	>58k
# World Nodes	38	99	47	109	3,169	3,141
# Eq. Classes	37	95	46	105	1,686	1,692
	2A	2B	2C	2D	2E	2F
# Solutions	78	1,400	39	13,780	1,248	17,368
# Rep. Sols.	6	60	3	1,160	128	4,160
# World Nodes	29	41	27	45	37	48
# Eq. Classes	17	23	15	30	26	36
	3A	3B	3C	3D	3E	
# Solutions	6	6	36	198	198	
# Rep. Sols.	4	4	16	92	92	
# World Nodes	16	16	21	26	23	
# Eq. Classes	15	15	18	23	23	

3.3.2 DBPedia

DBPedia[LIJ15] is an open source knowledge graph extracted from Wikipedia entries. From the dataset description: “The English version of the DBpedia knowledge base currently describes 3.77 million things, out of which 2.35 million are classified in a consistent Ontology,

Table 3.3: Structural equivalence classes when considering only part of the label and the full label.

Template	1A	1B	1C	1D	1E	1F
# Type Eq. Classes	37	67	45	73	36	67
# Full Eq. Classes	37	95	46	105	1,686	1,692

including 764,000 persons, 573,000 places (including 387,000 populated places), 333,000 creative works (including 112,000 music albums, 72,000 films and 18,000 video games), 192,000 organizations (including 45,000 companies and 42,000 educational institutions), 202,000 species and 5,500 diseases.”

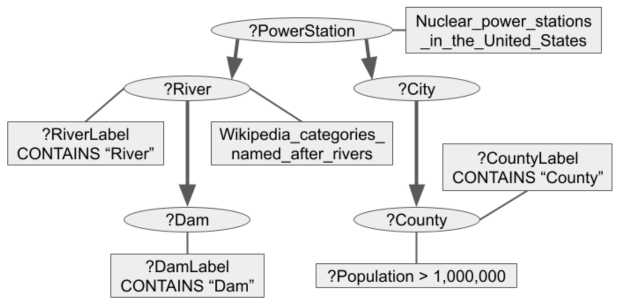
RDF (Resource Descriptive Framework)[RDF14] is a common format for storing knowledge graphs, which can then be queried using SPARQL (SPARQL Protocol and RDF Query Language)[SPA08]. Previous work [TRS13] has applied inexact SPARQL query matching to this knowledge graph using the TruST heuristic for inexact graph matching. Here, to show that our inexact subgraph matching method can be generalized to different measures of graph similarity, we replicate their inexact SPARQL query search using our framework.

The similarity score used in their work consists of three main parts. First, edges are defined to have a similarity of 0.1 if their types match, and 0.0 otherwise; we replicate this using our edgewise cost function, giving edges a cost of 0.1 when unmatched. Second, nodes are defined to have a similarity of 0.1 if they are both properties or both not properties; again, we replicate this using a nodewise cost of 0.1 for matching a property to a non-property. Finally, the APPROXIMATE keyword specifies conditions on nodes to satisfy certain conditions, such as Exact_URI_Match, Value_String_Contains, and Soft_Greater_Than. These each have a similarity of 1.0 if all conditions are met exactly. We again replicate this with a nodewise cost, taking the cost as 1.0 minus the similarity.

We examine the results of the comprehensive query from Section III of [TRS13] on

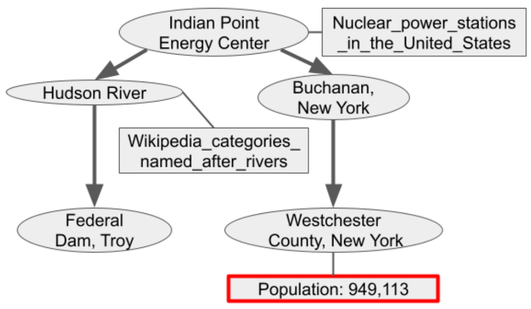
```
@prefix category: <http://dbpedia.org/resource/Category:> .
@prefix dbpedia-owl: <http://dbpedia.org/ontology/> .
@prefix dbpprop: <http://dbpedia.org/property/> .
@prefix dcterms: <http://purl.org/dc/terms/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix skos: <http://www.w3.org/2004/02/skos/core#> .

?PowerStation dcterms:subject
  category:Nuclear_power_stations_in_the_United_States .
?PowerStation dcterms:subject ?River .
?PowerStation dbpprop:locale ?City .
?City dbpedia-owl:isPartOf ?County .
?County rdfs:label ?CountyLabel .
?County dbpedia-owl:populationTotal ?Population .
?River rdfs:label ?RiverLabel .
?River skos:broader
 category:Wikipedia_categories_named_after_rivers .
?Dam dcterms:subject ?River .
?Dam rdfs:label ?DamLabel .

SIMILARITY category:Nuclear_power_stations_in_the_United_States
  Exact_URI_Match .
SIMILARITY category:Wikipedia_categories_named_after_rivers
  Exact_URI_Match .
SIMILARITY ?Population Soft_Greater_Than(1000000,800000) .
SIMILARITY ?CountyLabel Value_String_Contains("County") .
SIMILARITY ?RiverLabel Value_String_Contains("River") .
SIMILARITY ?DamLabel Value_String_Contains("Dam") .
```

Figure 3.4: Modified version of the comprehensive query from [TRS13] used for the DBPedia dataset.



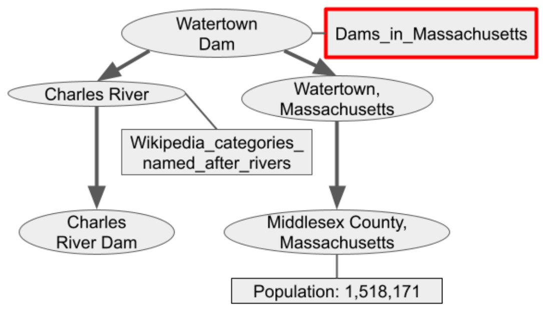
?PowerStation
Nuclear_power_stations _in_the_United_States
?River
?City
?RiverLabel CONTAINS "River"
Wikipedia_categories_ named_after_rivers
?CountyLabel CONTAINS "County"
?Dam
?County
?DamLabel CONTAINS "Dam"
?Population > 1,000,000

(a)



Indian Point Energy Center
Nuclear_power_stations _in_the_United_States
Hudson River
Buchanan, New York
Wikipedia_categories_ named_after_rivers
Federal Dam, Troy
Westchester County, New York
Population: 949,113

(b)



(c)

Figure 3.5: Template and results for the DBPedia dataset in graph format. Highlighted in red for each match are the deviations from the template. (a) Attributed graph template created from the comprehensive query from [TRS13]. (b) Best match for the DBPedia template. The original query asked for a population greater than 1,000,000. (c) Second best match for the DBPedia template. The original query asked for examples of nuclear power stations in the United States, not dams in Massachusetts.

DBPedia version 3.8, which we convert to an attributed graph template with a few minor changes. The original query compares each label to “County”; we believe this to be a typographical error, and instead compare to “Dam”, “River”, or “County” as appropriate. “dbpedia-owl:part_of” was replaced by “dbpedia-owl:isPartOf”, and “dbpprop:totalPop” was replaced by “dbpedia-owl:populationTotal”. We also changed the edge to the river category to “skos:broader” as we were unable to find any edges corresponding to “dcterms:subject”; this could be due to a typographical error, or simply a change in the DBPedia dataset between versions. We show the query used, including the aforementioned changes, in Figure 3.4.

Template	Result 1	Result 2	Result 3	Result 4	Result 5
?PowerStation	Indian Point Energy Center	Watertown Dam	Watertown Dam	Indian Point Energy Center	Indian Point Energy Center
Nuclear power stations in the United States	Nuclear power stations in the United States	Dams in Massachusetts	Buildings and structures in Watertown, MA	Nuclear power stations in the United States	Nuclear power stations in the United States
?River	Hudson River	Charles River	Charles River	Hudson River	Hudson River
?Dam	Federal Dam (Troy)	Charles River Dam	Charles River Dam	Eureka Shipyard	West Shore Railroad
?County	Westchester County, NY	Middlesex County, MA	Middlesex County, MA	Westchester County, NY	Westchester County, NY
?Population	949113	1518171	1518171	949113	949113
Similarity	96%	86%	86%	83%	83%

Table 3.4: Results for the DBPedia dataset. Similarity score is calculated by subtracting the cost from the maximum similarity of 7.4, and is shown as a percentage of the maximum similarity. Variable names are prefixed with a question mark.

We show these results in Table 3.4, taking the top 5 best matches to the provided template. The first match, with the highest similarity score to the query, matches the one found in [TRS13]; due to the population being below the required 1 million, it incurs a cost of 0.254. The second and third results are distinct from the first and almost identical to each other. They both replace the power station with Watertown Dam and ignoring the constraint that it must be a “nuclear power station in the United States”, incurring a cost of 1. Similarly, the fourth and fifth matches are identical to the first, but replace the dam with a shipyard and a railroad, incurring a cost of 1 for ignoring the constraint that the dam's label must contain the word "Dam".

3.A Varying Edgewise Cost Bound Weights

In some situations, the costs of certain nodes are more important than other nodes. For example, when a node has a known assignment, it is less important to know the cost associated with that node, and more important to know the costs of its surrounding nodes, so that they may be assigned candidates using the best possible heuristic.

We repeat the cost bound formulation from Equation (3.4), but replace $\frac{1}{2}$ by a parameter function $\alpha(t, t_o)$:

$$L_\alpha(t, w; \mathcal{G}_t, \mathcal{G}_w, f) = D_\mathcal{V}(t, w) + \sum_{t_o \in \mathcal{V}_t} \alpha(t, t_o)\Big(D_\mathcal{E}\big((t, t_o), (w, f(t_o))\big) + D_\mathcal{E}\big((t_o, t), (f(t_o), w)\big)\Big). \tag{3.10}$$

The new parameter function $\alpha(t, t_o)$ has the property that $\alpha(t, t_o) + \alpha(t_o, t) = 1$ under the constraint $0 \leq \alpha(t, t_o) \leq 1 \forall t, t_o$. If we wish to assign t more importance than t_o, we use $\alpha(t, t_o) = 1$, $\alpha(t_o, t) = 0$.

Similarly, we extend $B(t, w; \mathcal{G}_t, \mathcal{G}_w)$ to $B_\alpha(t, w; \mathcal{G}_t, \mathcal{G}_w)$:

$$B_\alpha(t, w; \mathcal{G}_t, \mathcal{G}_w) := D_\mathcal{V}(t, w) + \sum_{t_o \in \mathcal{V}_t} \alpha(t, t_o) \min_{w_o \in \mathcal{V}_w} \Big(D_\mathcal{E}\big((t, t_o), (w, w_o)\big) + D_\mathcal{E}\big((t_o, t), (w_o, w)\big)\Big).$$

In practice, the modified local cost bound in Equation (3.10) is used during the search algorithm to put less weight on assigned nodes. When a node is given an assignment, it is also given a relative weight of $\alpha = 0$, while its neighbors are given a weight of $\alpha = 1$ with respect to that node. For an edge between two assigned nodes, we default to $\alpha = \frac{1}{2}$.

3.B Restricting Candidates During Minimization

In the context of the search approach detailed in Subsection 3.2.4, at certain points in the algorithm, we have an upper bound U on the cost. At this point in the algorithm, the exact value of the cost bound is not needed if it is greater than or equal to U (or strictly greater in the context of inexact ASM).

Thus, we can instead define the local cost bound as

$$B(t, w; \mathcal{G}_t, \mathcal{G}_w) :=$$
$$\min\Big(U, D_{\mathcal{V}}(t, w) + \sum_{t_o \in \mathcal{V}_t} \alpha(t, t_o) \min_{w_o \in C(t_o)} \Big[D_{\mathcal{E}}\big((t, t_o), (w, w_o)\big) + D_{\mathcal{E}}\big((t_o, t), (w_o, w)\big)\Big]\Big),$$

where $C(t_o)$ is defined to be the set w_o of world nodes for which the known constrained cost bound $G(t_o, w; \mathcal{G}_t, \mathcal{G}_w)$ is strictly less than U (less than or equal to for inexact ASM). Doing so drastically improves computational performance and provides a tighter bound on costs that lie below U. This also refines cost bounds analogously to constraint propagation; as tighter global cost bounds G are found, this leads to tighter local cost bounds, which are then used to compute even tighter global cost bounds until we reach a final set of bounds.

CHAPTER 4

Fault-tolerant Subgraph Matching on Aligned Networks[*]

4.1 Fault-tolerant Subgraph Matching on Aligned Networks

Often, we wish to combine multiple networks together to better express the relationships between their nodes. For example, different datasets representing multiple modes of communication between the same set of entities can be combined into a single multilayer network. To combine these networks, we first need to identify which nodes correspond to the same real world entity and *align* them by combining them into a single node. The problem of identifying this correspondence is the graph alignment problem (also known as the graph matching problem or network alignment problem) and has been well studied in the literature [EKG13, LFF16, ZBV09].

However, in the context of aligned networks, existing subgraph matching methods will often fail if there are errors in the alignment. Though it is desirable to have an aligned network without errors to perform subgraph matching on, this is often impractical or impossible to obtain in a real-world context. Thus, we seek to develop a method for subgraph matching that takes into account the possibility of errors in the alignment.

Now, we consider the subgraph matching problem on aligned networks. We refer to Definition 1.2.1 for the definition of network, and Definition 1.2.3 for the definition of subgraph isomorphism.

[*]This chapter is adapted from [TY20].

Definition 4.1.1 (Alignment)**.** *We define an* alignment $\mathcal{A}$ *on a set of networks* $\mathcal{G}_1, \mathcal{G}_2, ...$ *as a set of disjoint sets of nodes contained within those networks. Each of these sets is referred to as an* individual alignment*, or sometimes just alignment when the context is clear.*

Definition 4.1.2 (Aligned Network)**.** *Given a set of networks* $\mathcal{G}_1, \mathcal{G}_2, ...$ *and an alignment* $\mathcal{A}$*, construct the* aligned network $\mathcal{G}^{aligned}$ *by replacing each set of nodes in the alignment with a single new node whose edges are the edges of the nodes in the set.*

In this chapter, we focus on subgraph matching, and not graph alignment; thus, we do not provide a method for finding alignments. Instead, we assume that one exists and that we have its output, $\mathcal{A}$, as well as the corresponding aligned network, $\mathcal{G}^{\text{aligned}}$.

Importantly, however, we do not assume that this method is perfect, but that it is somewhat prone to errors. Thus, in addition to $\mathcal{A}$, we define the *ground truth alignment* $\mathcal{A}_{GT}$ that represents the true alignment of entities within the real world, and a corresponding *ground truth aligned network* resulting from this alignment. In practice, no method exists that can reliably guarantee one hundred percent accuracy, and in many cases finding the ground truth alignment can be impossible.

Now, we can extend the concept of subgraph isomorphism to aligned networks.

Definition 4.1.3 (Alignable Subgraph Isomorphism)**.** *Given two networks* $\mathcal{G}_t$ *and* $\mathcal{G}_w$ *(referred to as* template *and* world *respectively), we define an* alignable subgraph isomorphism *as a set of two multivalued functions* $f_{nodes} : \mathcal{V}_t \rightarrow P(\mathcal{V}_w)$ *and* $f_{edges} : \mathcal{E}_t \rightarrow P(\mathcal{E}_w)$*, where* $P(S)$ *denotes the power set of* S*, if:*

- $\forall \mathbf{u}_t \in \mathcal{V}_t,\ f_{nodes}(\mathbf{u}_t) \neq \emptyset$
- $\forall E_t \in \mathcal{E}_t,\ f_{edges}(E_t) \neq \emptyset$
- $\forall E_t \in \mathcal{E}_t,\ \forall E_w \in f_{edges}(E_t),\ Src_w(E_w) \in f_{nodes}(Src_t(E_t))$
- $\forall E_t \in \mathcal{E}_t,\ \forall E_w \in f_{edges}(E_t),\ Dst_w(E_w) \in f_{nodes}(Dst_t(E_t))$

A large issue with this definition is that, given any alignable subgraph isomorphism, any candidate can be added to it without violating the conditions. Therefore, every single candidate participates in at least one alignable subgraph isomorphism. To deal with this issue of spurious "unnecessary" candidates, we introduce the concept of minimality.

Definition 4.1.4 (Minimal Alignable Subgraph Isomorphism)**.** *A minimal alignable subgraph isomorphism is a alignable subgraph isomorphism where no candidate can be removed without invalidating the conditions for alignable subgraph isomorphism. That is,* $\forall \mathbf{u}_t \in \mathcal{V}_t$*,* $\nexists \mathbf{v}_w \in f_{nodes}(\mathbf{u}_t)$ *such that*

$$f^s_{nodes}(\mathbf{x}_t) = \begin{cases} f_{nodes}(\mathbf{u}_t) - \{\mathbf{v}_w\}\,, & \mathbf{x}_t = \mathbf{u}_t, \\ f_{nodes}(\mathbf{x}_t) & \textit{elsewhere} \end{cases}$$

$$f^s_{edges}(E_t) = f_{edges}(E_t) - \\ \{E_w \in f_{edges}(E_t) | Src_w(E_w) = \mathbf{v}_w or Dst_w(E_w) = \mathbf{v}_w\}$$

is an alignable subgraph isomorphism.

It is important to recall our original problem context at this point. Within the alignment problem, there is presumed to be a ground truth alignment. Without access to this ground truth, we are performing subgraph matching on an aligned network resulting from a graph alignment algorithm, which often differs from the ground truth. A natural question arises: what is the connection between our results on the (potentially errorful) aligned network and actual subgraphs of the ground truth aligned network?

Although not all (minimal) alignable subgraph isomorphisms correspond to valid subgraph matches for the ground truth, the reverse is true under certain conditions: all subgraph matches on the ground truth are represented by (possibly more than one) alignable subgraph isomorphism.

Theorem 4.1.1. *Given multiple networks* $\{\mathcal{G}_1, \mathcal{G}_2, ...\}$*, their ground truth alignment* $\mathcal{A}_{GT}$*, and any other alignment with no incorrect individual alignments* $\mathcal{A}$*, every subgraph isomorphism* *on the ground truth corresponds to at least one minimal alignable subgraph isomorphism on the aligned network.*

Proof. Denote the subgraph isomorphism on the ground truth as $(f^{GT}_{\text{nodes}}, f^{GT}_{\text{edges}})$, mapping from nodes and edges of the template $\mathcal{G}_t$ to the ground truth aligned world network $\mathcal{G}^{GT}_w$. We then construct a minimal alignable subgraph isomorphism by considering the mapping from each template node $\mathbf{u}_t$ to the component nodes of the world node that it was matched to in the ground truth subgraph isomorphism, $f^{GT}_{\text{nodes}}(\mathbf{u}_t)$. Then, we merge those component nodes according to the given non-ground truth alignment $\mathcal{A}$.

This mapping is clearly an alignable subgraph isomorphism, as completing the remaining merges from $\mathcal{A}^{GT}$ that were not included in $\mathcal{A}$ will result in the ground truth world network and the ground truth subgraph isomorphism that we were given.

To identify a minimal alignable subgraph isomorphism from this alignable subgraph isomorphism, remove candidates in any order until no further candidates can be removed without violating the conditions of alignable subgraph isomorphism. The end result is then a minimal alignable subgraph isomorphism by definition. $\square$

Note that there are two possible kinds of errors in network alignment [DZK18b]: individual alignments that are incorrect but present in the given alignment, and individual alignments that are correct but are missing from the given alignment. Here, we consider only errors of the second kind, and assume that our given alignment contains no incorrect individual alignments. At first, it may seem difficult or impossible for an alignment algorithm to satisfy this property. However, technically, a trivial alignment algorithm that makes no alignments would be guaranteed to make no incorrect ones.

Note also that the converse of Theorem 4.1.1 is not true: there may exist minimal alignable subgraph isomorphisms that do not correspond to any subgraph isomorphism on the ground truth aligned network. For example, a minimal alignable subgraph isomorphism could exist that uses two different world nodes for two different roles in the template, but those world nodes are then merged in the ground truth.

4.2 Filtering

For our method, we build upon the filtering approach described in [MCT18]. For each template node, a list of possible candidates is tracked. We then filter the candidates for each template node based on logical rules. For example, the node-level statistics filter eliminates candidates that do not have as many incoming or outgoing edges as the corresponding template node. The topology filter acts along edges, disqualifying candidates when an edge in the template has no possible match in the corresponding neighborhood of the candidate. After the list of candidates has been filtered down, we then perform a branching tree search, trying candidates and eliminating possibilities until a match is found.

We extend the existing system of filters to the case of minimal alignable subgraph isomorphism. Instead of filtering out candidates that do not participate in any subgraph isomorphisms, we instead filter out candidates that do not participate in any minimal alignable subgraph isomorphisms. This is done by modifying the existing node-level statistics (Algorithm 9) and topology (Algorithm 10) filters.

4.2.1 Fault-tolerant Node-level Statistics Filter

In Chapter 2, the node-level statistics filter enforced that all candidates must have statistic values greater than or equal to those of their corresponding template node. However, in the context of fault-tolerant subgraph matching, this is not always the case. For example, if a template node is represented by multiple world nodes, its edges will be split between them. Thus, each of its candidates may have a lower degree than the original template node, due to only having a portion of the edges.

However, there are still candidates that we can filter out. For example, a node with degree 0 has no edges, and thus will never be a candidate in a minimal alignable subgraph isomorphism, as it will always be removable.

More generally, a candidate must have at least one edge that could correspond to a template edge connected to its corresponding template node. Thus, given the set of nonzero node statistics of the corresponding template node (in/out degree in each channel), at least one of these statistics must be nonzero for the world node to be considered a candidate. The details of this filter can be seen in Algorithm 9.

Algorithm 9 Fault-tolerant Node-level Statistics Filter

```
1:  Input template $\mathcal{G}_t$, world $\mathcal{G}_w$, candidate sets $D(\mathbf{v}_t)$
2:  for template node $\mathbf{v}_t \in \mathcal{V}_t$
3:      for candidate $\mathbf{v}_w \in D(\mathbf{v}_t)$
4:          any_nonzero $\leftarrow$ False
5:          for statistic $\in$ statistics
6:              if statistic($\mathbf{v}_t$) $> 0$
7:                  if statistic($\mathbf{v}_w$) $> 0$
8:                      any_nonzero $\leftarrow$ True
9:          if any_nonzero is not True
10:             Remove $\mathbf{v}_w$ from $D(\mathbf{v}_t)$
11: Output updated candidate set $D(\mathbf{v}_t)$ for each $\mathbf{v}_t \in \mathcal{V}_t$
```

4.2.2 Fault-tolerant Topology Filter

We can also similarly modify the topology filter to obtain a version that lets us find minimal alignable subgraph isomorphisms. As in the topology filter, we check the neighbors of each template node, and look at their candidates. For regular subgraph isomorphism, we would enforce that all edges must have a corresponding matching edge to a neighboring candidate. However, in this case, we enforce only that at least one neighbor must have a candidate with at least one matching edge. The details of this filter can be seen in Algorithm 10.

Algorithm 10 Fault-tolerant Topology Filter

```
1: Input template $\mathcal{G}_t$, world $\mathcal{G}_w$, candidate sets $D(\mathbf{v}_t)$
2: for template node $\mathbf{v}_t \in \mathcal{V}_t$
3:     for candidate $\mathbf{v}_w \in D(\mathbf{v}_t)$
4:         nbr_cand_found $\leftarrow$ False
5:         for template node $\mathbf{u}_t$ in neighbors of $\mathbf{v}_t$
6:             for candidate $\mathbf{u}_w \in D(\mathbf{u}_t)$
7:                 if $\mathcal{E}_w(\mathbf{v}_w, \mathbf{u}_w) > 0$ and $\mathcal{E}_t(\mathbf{v}_t, \mathbf{u}_t) > 0$
8:                     nbr_cand_found $\leftarrow$ True
9:                 if $\mathcal{E}_w(\mathbf{u}_w, \mathbf{v}_w) > 0$ and $\mathcal{E}_t(\mathbf{u}_t, \mathbf{v}_t) > 0$
10:                    nbr_cand_found $\leftarrow$ True
11:        if nbr_cand_found is not True
12:            Remove $\mathbf{v}_w$ from $D(\mathbf{v}_t)$
13: Output updated candidate set $D(\mathbf{v}_t)$ for each $\mathbf{v}_t \in \mathcal{V}_t$
```

4.2.3 Handling Nodes With Known Alignments

In some contexts, we know *a priori* that certain template nodes are aligned correctly. For example, in a communications network, nodes representing entities like people could have alignment errors, but nodes representing individual phone numbers, strings of text, or other non-entity based information may not. Similarly, in certain contexts, nodes that represent well-known entities that are easily aligned can also be treated as having a known alignment.

When some of the alignments are known, we can apply filters separately to different parts of the template. This is trivial for the node-level statistics filter, as we simply apply the regular node-level statistics filter to the nodes that have known alignments.

The topology filter can also be modified for known alignment in the following way. For template nodes with known alignments, but which connect to nodes with possible alignment errors, we first total the number of edges over all the candidates for the neighbor before checking against the template. In the other direction, for template nodes with possible errors connecting to those with known alignment, the existing version of the fault-tolerant topology filter still applies. The details of this modified topology filter to handle known alignments can be seen in Algorithm 11.

4.3 Recursive Merging

Once enough candidates have been removed via filtering, we begin a branching tree search, much as we would when not dealing with alignment errors. However, at each branch of the tree, in addition to performing an assignment, we also perform a single alignment, merging a subset of the candidates for that template node.

This branching can be done for every single possible subset of candidates. However, we can branch more efficiently by eliminating possibilities in order of increasing merge size. If a set has a valid match, then every superset of that set will have an equivalent match and will be non-minimal, and thus we can skip trying that merge. This helps to limit the computational complexity of the algorithm. When branching for every subset, for a node with n candidates, there are $2^n - n - 1$ possible subsets to consider. With the subset restriction, this decreases to $\binom{n}{2}$ in the best case where only 2-node merges need to be considered, with the rest eliminated. In practice, the computational complexity can also be heavily limited by restricting the maximum size of the merge, decreasing from $O(2^n)$ to $O(2^M)$, where M is the maximum merge size. This may also be useful in application contexts where large merges carry little intrinsic meaning and can be safely ignored.

4.4 Experiments

All experiments were run on an HP Z8 G4 Workstation with two 12-core Intel® Xeon® Gold 6136 CPUs.

Algorithm 11 Topology Filter For Known Alignments

```
1: **Input** template $\mathcal{G}_t$, world $\mathcal{G}_w$, candidate sets $D(\mathbf{v}_t)$
2: for template node $\mathbf{v}_t \in \mathcal{V}_t$
3:     for candidate $\mathbf{v}_w \in D(\mathbf{v}_t)$
4:         nbr_cand_found ← False
5:         for template node $\mathbf{u}_t$ in neighbors of $\mathbf{v}_t$
6:             if $\mathbf{v}_t$ has known alignment
7:                 if $\mathbf{u}_t$ has known alignment
8:                     for candidate $\mathbf{u}_w \in D(\mathbf{u}_t)$
9:                         if $\mathcal{E}_w(\mathbf{v}_w, \mathbf{u}_w) \geq \mathcal{E}_t(\mathbf{v}_t, \mathbf{u}_t)$ and $\mathcal{E}_w(\mathbf{u}_w, \mathbf{v}_w) \geq \mathcal{E}_t(\mathbf{u}_t, \mathbf{v}_t)$
10:                            nbr_cand_found ← True
11:                else
12:                    in_tot ← 0, out_tot ← 0
13:                    for candidate $\mathbf{u}_w \in D(\mathbf{u}_t)$
14:                        in_tot ← in_tot + $\mathcal{E}_w(\mathbf{v}_w, \mathbf{u}_w)$
15:                        out_tot ← out_tot + $\mathcal{E}_w(\mathbf{u}_w, \mathbf{v}_w)$
16:                    if in_tot $\geq \mathcal{E}_t(\mathbf{v}_t, \mathbf{u}_t)$ and out_tot $\geq \mathcal{E}_t(\mathbf{u}_t, \mathbf{v}_t)$
17:                        nbr_cand_found ← True
18:            else
19:                for candidate $\mathbf{u}_w \in D(\mathbf{u}_t)$
20:                    if $\mathcal{E}_w(\mathbf{v}_w, \mathbf{u}_w) > 0$ and $\mathcal{E}_t(\mathbf{v}_t, \mathbf{u}_t) > 0$
21:                        nbr_cand_found ← True
22:                    if $\mathcal{E}_w(\mathbf{u}_w, \mathbf{v}_w) > 0$ and $\mathcal{E}_t(\mathbf{u}_t, \mathbf{v}_t) > 0$
23:                        nbr_cand_found ← True
24:        if nbr_cand_found is not True
25:            Remove $\mathbf{v}_w$ from $D(\mathbf{v}_t)$
26: Output updated candidate set $D(\mathbf{v}_t)$ for each $\mathbf{v}_t \in \mathcal{V}_t$
```

4.4.1 AIDA Version 2.0.2

We perform minimal alignable subgraph isomorphism on the AIDA Version 2.0.2 dataset provided to us by Pacific Northwest National Laboratory (PNNL) [CPM18, Hov20] as part of the DARPA–MAA program [Sch18]. This dataset consists of a knowledge graph in RDF format describing political events in the Ukraine. Templates also provided by PNNL describe the assassination of politician Oleksandr Peklushenko, sometimes under various aliases such as "Aleksandr Peklushenko" or "Pelushenko".

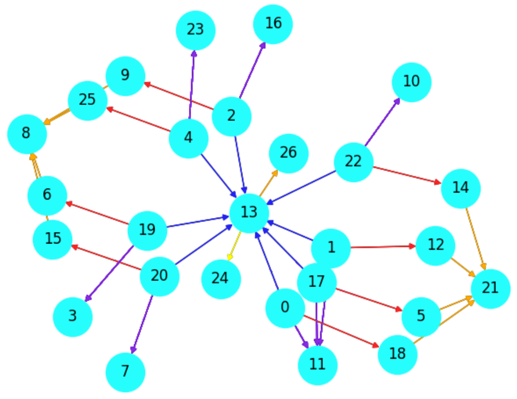


Figure 4.1: A picture of AIDA Version 2.0.2 template 2. Colors on edges represent different edge types, and the number in each node is a unique ID number. The central node, labeled 13, is the entity that represents Oleksandr Peklushenko.

To simulate a specific alignment error, we consider the case where these aliases are identified as separate entities in the world graph, and as a single entity in the template. This would be the result of a missed alignment. Since the remaining nodes in the template correspond to specific terms within the ontology (e.g. "PER.Politician"), which we assume have no possibility of incorrect alignment, we focus our efforts on the "Oleksandr Peklushenko" entity and assume all other nodes have known alignments.

To cut down on the number of required merges, we impose that entities to be merged must have names relatively close to "Oleksandr Peklushenko" or to its component tokens "Oleksandr" and "Peklushenko". To compare strings, we use the Levenshtein distance [Lev66],

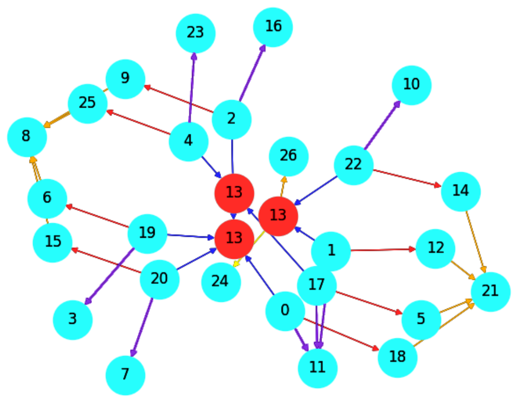


Figure 4.2: A picture of a minimal alignable subgraph isomorphism for AIDA Version 2.0.2 template 2. Colors on edges represent different edge types, and the number in each node is a unique ID number. The three nodes that can be merged together to form node 13 are highlighted in red.

which is the minimum number of string edits (additions, deletions, and substitutions) required to turn one string into another. We tested maximum Levenshtein distances between 3 and 6, with no observable differences in the results. However, at a distance of 7 or higher, the number of results increases greatly. In this context, a distance of 7 is extremely large, as the component tokens have length 9 and 11.

A dataset specific restriction that we can apply is that all entities to be merged must have died (i.e. they must be the target of an edge with the type Life.Die_Victim). In the context of a real-world situation, this may be a reasonable step to take, if the entity in question is known to have died. Even without this restriction, we can find all minimal alignable subgraph isomorphisms; however, the runtime increases significantly, and the number of possible matches also increases.

Running our algorithm, we find 228 minimal alignable subgraph isomorphisms. Although we know from the ground truth alignment (provided to us by PNNL as part of the dataset) that a 3-node merge should have been performed, we do not impose this condition on the algorithm. Thus, the algorithm produces results corresponding to four possible 3-node alignments, six possible 4-node alignments, and four possible 5-node alignments. Restricting only to 3-node alignments, we have just 36 minimal alignable subgraph isomorphisms. These are split between the four possible 3-node alignments: two of them have 12 each, and the other two have 6 each.

These solutions are visualized in Fig. 4.3. We depict the template network and a subgraph of all world nodes that participate in a solution. We color-code each template node with its associated group of candidate nodes in the world network to illustrate how to map the template network onto the world network [NYG19]. The boxed nodes in the world network are those nodes that participate in 3-node, 4-node, or 5-node merges in order for an isomorphism to possible. Observe how without such a merge, we could not guarantee connections to each of the pink, lavender, light green, and dark green sets of nodes that are necessary, as seen in the template network.

We also note that the world network in this example demonstrates possible alignment errors that do not preclude the possibility for a solution. Note that template nodes 15 and 20 (in purple and pink, respectively) can be mapped to either nodes 30 and 14 or nodes 41 and 32 and still maintain an isomorphism. Similar cases can be seen with the blue and lavender groups and the red and orange groups. Our algorithm does not merge these groups, as it is not strictly needed, but more sophisticated algorithms that detect automorphic similarity could possibly be used to merge these other groups, which would lead to a simplified description of the solution space.

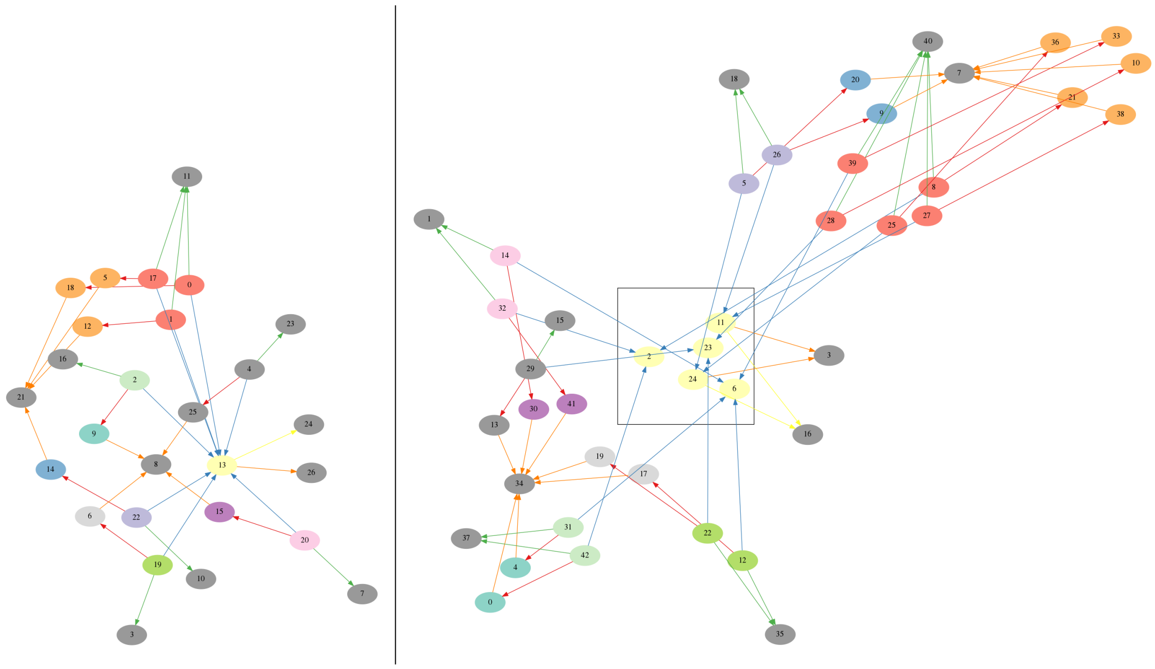


Figure 4.3: AIDA V2.0.2 template network (left) and subgraph of world network with nodes participating in a minimal alignable subgraph isomorphism (right). The number in each node is a unique ID number. Groups of nodes in the world network of one color (that is not dark gray) are the candidates for template nodes of the same color. The boxed group in the world network are the nodes that are involved in a merging.

CHAPTER 5

A Constraint Propagation Approach for Identifying Biological Pathways in COVID-19 Knowledge Graphs

5.1 Introduction

Due to the ever-increasing pace of scientific advancement and discovery, there have been several proposed approaches [SR21, WTD19, NMG10] for automated extraction of facts and knowledge from the scientific literature. Using natural language processing, logical statements and causal relationships can be extracted from a source text. These can then be assembled into a structured knowledge base, which can assist domain experts in answering important questions.

In this chapter, we examine one such knowledge base, a knowledge graph dataset on Coronavirus Disease 2019 (COVID-19) created by PNNL as part of the DARPA–MAA program. Per PNNL's technical description, this dataset represents causal knowledge about biological entities which was automatically extracted from the COVID-19 Open Research Dataset [WLC20] using the INDRA knowledge assembler [GBS17], the Blender knowledge graph [WLW20], and the Comparative Toxicogenomics Database [DWW21].

Within this context, we seek to identify biological activation pathways that answer questions about COVID-19, such as how Severe Acute Respiratory Syndrome Coronavirus 2 (SARS-COV-2) induces a cytokine storm in severely ill COVID-19 patients [ZPM21]. To identify such pathways, we first model our biological query as a *path-based graph template* with constraints. We then construct a constraint propagation scheme similar to that in Chapter 2, taking into account the variable length of the path as well as the specified constraints. Finally, we perform a depth-first search for candidate paths that match the template.

5.2 Problem Statement

We define the problem context as follows: given an attributed graph $\mathcal{G}$, a set of possible start nodes S and a set of possible end nodes E, identify all directed paths leading from a node in S to a node in E. To prevent a enormous number of paths from arising, we also specify that the paths must obey some of the following constraints.

First, we constrain the length of the path to contain at least l_{min} edges and at most l_{max} edges, inclusive. This constraint is extremely important, as setting an upper bound on the path length ensures the number of paths is bounded.

Second, we constrain each path to only contain nodes or edges of certain allowed types (A_v for nodes and A_e for edges). This constraint is simple to implement, as it is equivalent to querying a subgraph taken over all nodes/edges belonging to the allowed types. However, it can be very important in applications, as it allows researchers to specify which types of nodes or edges should and should not occur in paths of interest.

Finally, we impose a constraint on the number of edges of certain types that may occur in paths. For example, one could say that a path may only contain two, three, or five edges of type A, eliminating paths containing exactly zero, one, or four edges of type A (and six or more).

In our application, this constraint is used by the provided template to enforce parity in the number of inhibitive edges. These edges represent inhibition relationships, where the activation of one node causes the inhibition of another (inhibition being the opposite of activation). In the context of our application, we consider a biological activation pathway to be a chain of causal relationships (represented by directed edges in the causal knowledge graph) where the activation of the start of the pathway leads to a corresponding activation at the end. This pathway may, however, contain inhibition relationships. Here, we assume that inhibiting a node that inhibits another node gives a net result of activation. Thus, we seek to identify paths that contain an even number of inhibitive edges.

Putting these conditions together, we have the following definition of a path-based graph template.

Definition 5.2.1 (Path-based Graph Template)**.** *A path-based graph template (S, E, l_{min}, l_{max}, A_v, A_e, C_e, N_c) consists of the following constraints on a path:*

- *A set of allowed start nodes S and a set of allowed end nodes E*
- *A minimum length l_{min} and maximum length l_{max}*
- *A set of allowed node labels A_v and a set of allowed edge labels A_e*
- *Optionally, a restriction on the count of edges in the path whose labels belong to the set C_e. This count must belong to the set N_c for the constraint to be met.*

5.3 Algorithm

To identify these paths, we implement a constraint propagation approach similar to Chapter 2. As before, we seek to identify candidates for our given template nodes. However, these template nodes no longer represent a fixed graph, but instead represent *roles* within the path that may or may not be needed, as the path has variable length.

To demonstrate this approach, let us ignore for now the third constraint on edge type counts, and instead focus on just the path length and allowed type constraints. We construct the template as a series of nodes, with the first representing the source, the second representing the second node in the path, the third representing a possible third node in the path, and so on, up to the maximum path length.

The candidate refinement process is then as follows. Given a candidate for a template role, if it is not the source, first check that it has an edge leading to it from a candidate for the previous node. If there is no such node, then this candidate cannot participate in a path, and so is eliminated. If the role is the source, instead check that it is one of the specified possible sources.

Similarly, for the outgoing edge, we check one of two conditions: either the candidate must be one of the possible end nodes, or it must have an outgoing edge to a candidate for the next node in the path. If neither of these conditions are met, the candidate is eliminated. Note that if the template role in question is at the maximum possible length, there is no outgoing edge, so its only candidates are the possible end nodes. Pseudocode for checking these conditions can be found in Algorithm 12.

We repeat these checks until no more candidates can be eliminated. At that point, we perform a depth-first tree search on the remaining candidates to identify all possible paths, starting from the source and going until an endpoint is found. At each step in the tree search, we choose a candidate for the current template role, then rerun the constraint propagation repeatedly until the remaining candidates are unchanged. Once an endpoint has been identified or all candidates are eliminated, we backtrack up the tree and choose another candidate, until all possible choices are exhausted. The details of this algorithm can be seen in Algorithm 13.

To extend this approach to edge counts, we introduce an augmented set of template roles. Now, instead of each template role representing a possible position in the path, each role represents both a position and the number of edges used to reach that position that match the edge type filter.

Now, instead of checking constraints only on the previous and next roles in the path, each role must check constraints on up to four roles: if the template role (p, n) has position p and edge count n, it must check both roles $(p-1, n-1)$ and $(p-1, n)$ for edges leading to (p, n), and it must check $(p+1, n)$ and $(p+1, n+1)$ for edges going from (p, n) (or (p, n) must be a valid endpoint). Additionally, edges between roles with different edge counts must match the edge type filter, since this implies that the edge in question has changed the edge count.

Algorithm 12 Constraint Enforcement (Without Edge Counts)

```
1:  function Filter
2:      Input start nodes $S$, end nodes $E$, minimum path length $l_{\min}$, maximum path length $l_{\max}$, allowed node labels $A_v$, allowed edge labels $A_e$, world $\mathcal{G}_w$, candidate set $D(\mathbf{t}_i)$ for each $\mathbf{t}_i$, $0 \leq i \leq l_{\max}$
3:      for $i = 0; i \leq l_{\max}; i{+}{+}$
4:          $D(\mathbf{t}_i) \leftarrow \{\mathbf{w} | \mathbf{w} \in D(\mathbf{t}_i), \mathcal{L}(\mathbf{w}) \in A_v\}$        ▷ Filter on node labels
5:      $D_{\text{old}} \leftarrow$ null
6:      while $D \neq D_{\text{old}}$
7:          $D_{\text{old}} \leftarrow D$
8:          for $i = 0; i \leq l_{\max}; i{+}{+}$
9:              if $i == l_{\max}$
10:                 $D(\mathbf{t}_i) \leftarrow D(\mathbf{t}_i) \cap E$
11:             else
12:                 $D(\mathbf{t}_i) \leftarrow \{\mathbf{w} | \mathbf{w} \in D(\mathbf{t}_i), \exists e = (\mathbf{w}, \mathbf{w_o}) \in \mathcal{E}_w, \mathbf{w_o} \in D(\mathbf{t}_{i+1}), \mathcal{L}(e) \in A_e\}$
13:                 if $i \geq l_{\min}$
14:                     $D(\mathbf{t}_i) \leftarrow D(\mathbf{t}_i) \cup E$
15:             if $i == 0$
16:                 $D(\mathbf{t}_i) \leftarrow D(\mathbf{t}_i) \cap S$
17:             else
18:                 $D(\mathbf{t}_i) \leftarrow \{\mathbf{w} | \mathbf{w} \in D(\mathbf{t}_i), \exists e = (\mathbf{w_o}, \mathbf{w}) \in \mathcal{E}_w, \mathbf{w_o} \in D(\mathbf{t}_{i-1}), \mathcal{L}(e) \in A_e\}$
19:     Output updated candidate set $D(\mathbf{t}_i)$ for each $\mathbf{t}_i$, $0 \leq i \leq l_{\max}$
```

Algorithm 13 Pathfinding (Without Edge Counts)

```
1:  function PathSearch
2:      Input start nodes $S$, end nodes $E$, minimum path length $l_{\text{min}}$, maximum path length $l_{\text{max}}$, allowed node labels $A_v$, allowed edge labels $A_e$, world $\mathcal{G}_w$, candidate set $D(\mathbf{t}_i)$ for each $\mathbf{t}_i$, $0 \leq i \leq l_{\text{max}}$, current path position $i$, current path $p_{\text{curr}}$, list of completed paths $P$
3:      $D_{\text{orig}} \leftarrow D$                                    ▷ Save original candidates
4:      for world node $\mathbf{v}_w \in D(\mathbf{t}_i)$, $\mathbf{v}_w \notin p_{\text{curr}}$
5:          $D(\mathbf{t}_i) \leftarrow \{\mathbf{v}_w\}$                  ▷ Assign $\mathbf{v}_w$ to $\mathbf{t}_i$
6:          $D \leftarrow$ Filter($S$, $E$, $l_{\text{min}}$, $l_{\text{max}}$, $A_v$, $A_e$, $\mathcal{G}_w$, $D$)      ▷ Check constraints
7:          if $\mathbf{v}_w \in E$ and $i \geq l_{\text{min}}$
8:              Append $p_{\text{curr}}$ to $P$              ▷ Append the finished path to the pathlist
9:          if $i < l_{\text{max}}$
10:             PathSearch($S$, $E$, $l_{\text{min}}$, $l_{\text{max}}$, $A_v$, $A_e$, $\mathcal{G}_w$, $D$, $i+1$, $p_{\text{curr}} + \mathbf{v}_w$, $P$)
11:         $D \leftarrow D_{\text{orig}}$                   ▷ Reset candidate set to original values
12:     Output List of completed paths $P$
```

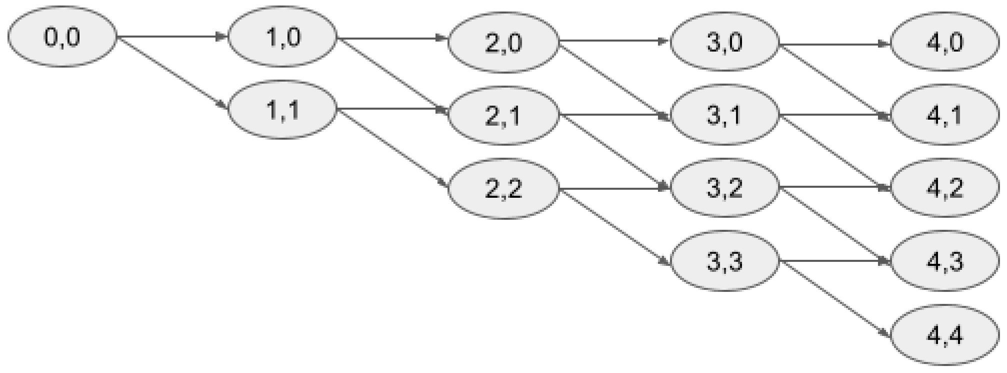


Figure 5.1: Template diagram for pathfinding with edge counts. Each cell represents a possible template role that could appear in the path. The first number in each cell represents the position in the path, with 0 being the start. The second number represents the number of edges used that match the edge type filter. Arrows are drawn where one state can lead to another through the addition of an edge to the path.

Similarly, edges between roles of the same edge count must not match the edge type filter. The details of the extended constraint enforcement are shown in Algorithm 14.

Similarly, we extend the path search algorithm to accommodate branching along possible paths, as shown in Algorithm 15.

5.4 Results

We evaluate our results on version 3.2.1 of the COVID Indra–Blender dataset created by PNNL. This dataset has 85,938 nodes and 3,138,683 edges. There are four types of nodes: 13,377 nodes of type "Chemical", 2634 "Disease", 46445 "Gene", and 23,482 "Protein". The edges are more varied, being broken down into 184 types. All code was run on an HP Z8 G4 Workstation with two 12-core Intel® Xeon® Gold 6136 CPUs.

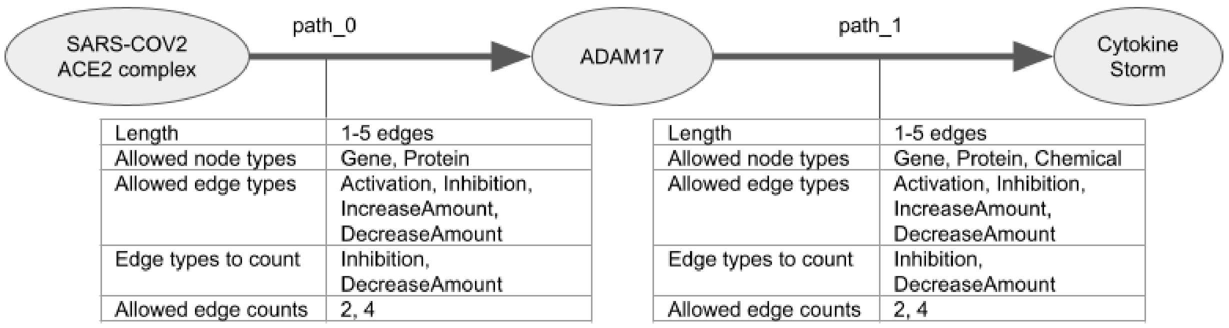
We examine the provided template 1 (see Figure 5.2), which is a structural template describing IL-6 amplification, a proposed mechanism for how COVID-19 induces cytokine

Algorithm 14 Constraint Enforcement (With Edge Counts)

```
1: function FILTER
2:     Input start nodes $S$, end nodes $E$, minimum path length $l_{\min}$, maximum path length $l_{\max}$, allowed node labels $A_v$, allowed edge labels $A_e$, edge labels to count $C_e$, valid edge counts $N_c$, world $\mathcal{G}_w$, candidate set $D(\mathbf{t}_{i,j})$ for each $\mathbf{t}_{i,j}$, $0 \leq j \leq i \leq l_{\max}$
3:     for $i = 0; i \leq l_{\max}; i{+}{+}$
4:         for $j = 0; j \leq i; j{+}{+}$
5:             $D(\mathbf{t}_{i,j}) \leftarrow \{\mathbf{w} | \mathbf{w} \in D(\mathbf{t}_{i,j}), \mathcal{L}(\mathbf{w}) \in A_v\}$        ▷ Filter on node labels
6:     $D_{\text{old}} \leftarrow$ null
7:     while $D \neq D_{\text{old}}$
8:         $D_{\text{old}} \leftarrow D$
9:         for $i = 0; i \leq l_{\max}; i{+}{+}$
10:            for $j = 0; j \leq i; j{+}{+}$
11:                if $i == l_{\max}$
12:                    if $j \in N_c$
13:                        $D(\mathbf{t}_{i,j}) \leftarrow D(\mathbf{t}_{i,j}) \cap E$
14:                    else
15:                        $D(\mathbf{t}_{i,j}) \leftarrow \varnothing$
16:                else
17:                    $D_1 \leftarrow \{\mathbf{w} | \mathbf{w} \in D(\mathbf{t}_{i,j}), \exists e = (\mathbf{w}, \mathbf{w_o}) \in \mathcal{E}_w, \mathbf{w_o} \in D(\mathbf{t}_{i+1,j}), \mathcal{L}(e) \in A_e, \mathcal{L}(e) \notin C_e\}$
18:                    $D_2 \leftarrow \{\mathbf{w} | \mathbf{w} \in D(\mathbf{t}_{i,j}), \exists e = (\mathbf{w}, \mathbf{w_o}) \in \mathcal{E}_w, \mathbf{w_o} \in D(\mathbf{t}_{i+1,j+1}), \mathcal{L}(e) \in A_e, \mathcal{L}(e) \in C_e\}$
19:                    $D(\mathbf{t}_{i,j}) \leftarrow D_1 \cup D_2$
20:                    if $i \geq l_{\min}$ and $j \in N_c$
21:                        $D(\mathbf{t}_{i,j}) \leftarrow D(\mathbf{t}_{i,j}) \cup E$
22:                if $i == 0$
23:                    $D(\mathbf{t}_{i,j}) \leftarrow D(\mathbf{t}_{i,j}) \cap S$
24:                else
25:                    $D_1 \leftarrow \{\mathbf{w} | \mathbf{w} \in D(\mathbf{t}_{i,j}), \exists e = (\mathbf{w_o}, \mathbf{w}) \in \mathcal{E}_w, \mathbf{w_o} \in D(\mathbf{t}_{i-1,j}), \mathcal{L}(e) \in A_e, \mathcal{L}(e) \notin C_e\}$
26:                    $D_2 \leftarrow \{\mathbf{w} | \mathbf{w} \in D(\mathbf{t}_{i,j}), \exists e = (\mathbf{w_o}, \mathbf{w}) \in \mathcal{E}_w, \mathbf{w_o} \in D(\mathbf{t}_{i-1,j-1}), \mathcal{L}(e) \in A_e, \mathcal{L}(e) \in C_e\}$
27:                    $D(\mathbf{t}_{i,j}) \leftarrow D_1 \cup D_2$
28:     Output updated candidate set $D(\mathbf{t}_{i,j})$ for each $\mathbf{t}_i$, $0 \leq j \leq i \leq l_{\max}$
```

Algorithm 15 Pathfinding (With Edge Counts)

```
1: function PathSearch
2:     Input start nodes $S$, end nodes $E$, minimum path length $l_{\min}$, maximum path length $l_{\max}$, allowed node labels $A_v$, allowed edge labels $A_e$, edge labels to count $C_e$, valid edge counts $N_c$, world $\mathcal{G}_w$, candidate set $D(\mathbf{t}_i)$ for each $\mathbf{t}_i$, $0 \leq i \leq l_{\max}$, current path position $i$, current edge count $j$, current path $p_{\text{curr}}$, list of completed paths $P$
3:     $D_{\text{orig}} \leftarrow D$                                              ▷ Save original candidates
4:     for world node $\mathbf{v}_w \in D(\mathbf{t}_{i,j})$, $\mathbf{v}_w \notin p_{\text{curr}}$
5:         $D(\mathbf{t}_{i,j}) \leftarrow \{\mathbf{v}_w\}$                                  ▷ Assign $\mathbf{v}_w$ to $\mathbf{t}_{i,j}$
6:         $D \leftarrow$ Filter($S$, $E$, $l_{\min}$, $l_{\max}$, $A_v$, $A_e$, $C_e$, $N_c$, $\mathcal{G}_w$, $D$)          ▷ Check constraints
7:         if $\mathbf{v}_w \in E$ and $i \geq l_{\min}$ and $j \in N_c$
8:             Append $p_{\text{curr}}$ to $P$                    ▷ Append the finished path to the pathlist
9:         if $i < l_{\max}$
10:            PathSearch($S$, $E$, $l_{\min}$, $l_{\max}$, $A_v$, $A_e$, $C_e$, $N_c$, $\mathcal{G}_w$, $D$, $i+1$, $j$, $p_{\text{curr}} + \mathbf{v}_w$, $P$)
11:            PathSearch($S$, $E$, $l_{\min}$, $l_{\max}$, $A_v$, $A_e$, $C_e$, $N_c$, $\mathcal{G}_w$, $D$, $i+1$, $j+1$, $p_{\text{curr}} + \mathbf{v}_w$, $P$)
12:        $D \leftarrow D_{\text{orig}}$                              ▷ Reset candidate set to original values
13:    Output List of completed paths $P$
```



Length	1-5 edges
Allowed node types	Gene, Protein
Allowed edge types	Activation, Inhibition, IncreaseAmount, DecreaseAmount
Edge types to count	Inhibition, DecreaseAmount
Allowed edge counts	2, 4

Length	1-5 edges
Allowed node types	Gene, Protein, Chemical
Allowed edge types	Activation, Inhibition, IncreaseAmount, DecreaseAmount
Edge types to count	Inhibition, DecreaseAmount
Allowed edge counts	2, 4

Figure 5.2: Template 1 of the PNNL COVID V3.2.1 dataset. The template describes a path from the SARS-COV-2 ACE2 receptor to cytokine storms via activation of ADAM17.

storms[HUT20]. This template takes the form of two paths. The first path is an activation pathway from two possible nodes representing the SARS-CoV-2 ACE2 complex to ADAM metallopeptidase domain 17/10 (ADAM17), and is constrained to have between one and five edges (inclusive), with either two or four edges being of type “Inhibition” or “DecreaseAmount”. The nodes in the first path are constrained to be of type “Gene” or “Protein”. The second path is another activation pathway from ADAM17 to various representations of cytokine storm in the dataset; it is also constrained to have between one and five edges, with two or four inhibitive edges. However, the nodes in the second path, in addition to “Gene” and “Protein”, are also allowed to be of type “Chemical”. Note that the template does not allow a count of zero inhibitive edges, despite being a valid option for an activation pathway; this was later found by PNNL to be an issue with the provided template, but was not fixed for this version of the dataset.

To cut down on the number of possible paths, as well as to ensure that only activation relationships are represented in the path, we imposed an additional constraint that did not appear in the template, namely that activating edges must be of type “Activation” or “IncreaseAmount” to parallel the inhibiting edge types of “Inhibition” and “DecreaseAmount”. The original template did not place any constraint on which types could be considered activating, except that they not be “Inhibition” or “DecreaseAmount”. Thus, initial attempts often

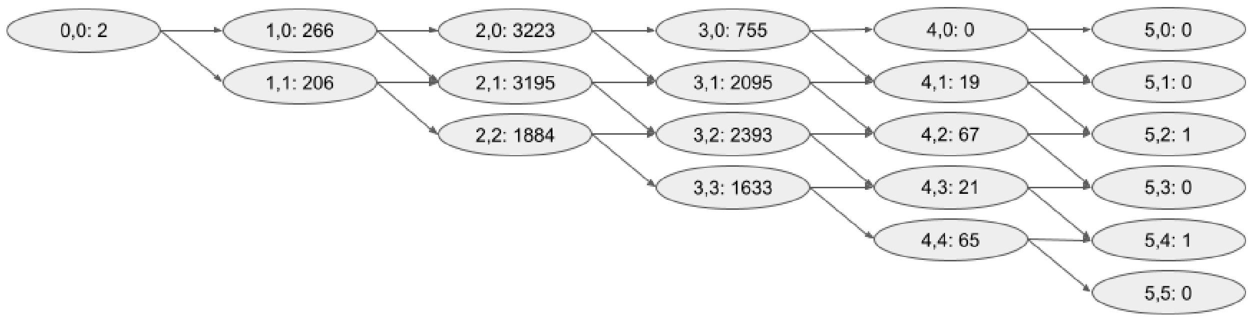


Figure 5.3: Candidate counts for the first path in template 1 of the COVID V3.2.1 dataset. The first number in each cell represents the position, the second represents the number of edges used that match the edge type filter, and the third is the number of candidates for that template role.

included paths with non-activation edge types treated as activation, such as “affectsˆbinding”, “affectsˆcotreatment”, or “decreasesˆreaction”.

We find 5,591,288 matches for the first path constraint in the template and 13,529,145 matches for the second path constraint. Although this solution space is very large, it results from a surprisingly small number of candidates: the largest number of candidates for a given template role is only on the order of a few thousand. The huge number of solutions is thus not due to a large number of disjoint matches, but rather a large number of overlapping matches. In figures Figures 5.3 and 5.4, we show the candidate spaces for each path constraint by counting the number of candidates for each template role. We also observe that the largest number of candidates appear in the middle of the path, in positions 2 and 3. This makes sense, as the ends of the path are more constrained, being fixed to one of only a few possible values.

Due to the enormity of the solution space, it is unlikely that all results for this template provide reasonable alternatives for explaining the link between SARS-COV-2 and cytokine storm formation. However, the relatively small size of the candidate set gives hope that, with some additional constraints on the path, a concise representation of the solution space can be identified and presented to a domain expert for further investigation.

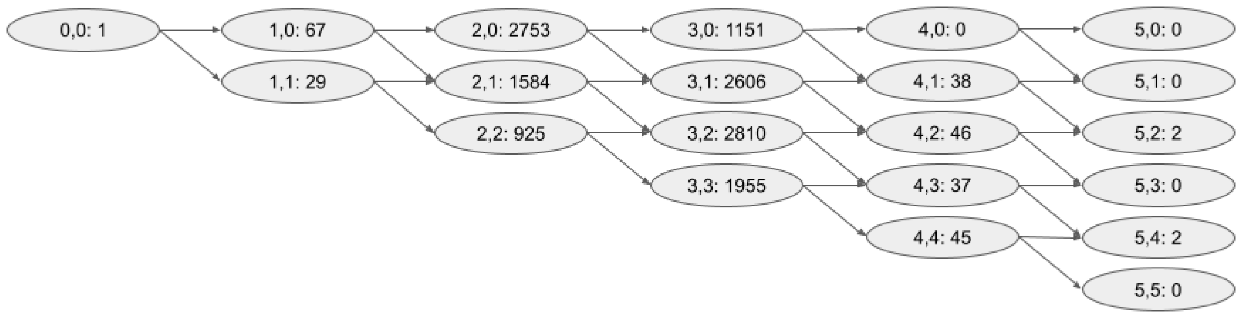


Figure 5.4: Candidate counts for the second path in template 1 of the COVID V3.2.1 dataset. The first number in each cell represents the position, the second represents the number of edges used that match the edge type filter, and the third is the number of candidates for that template role.

CHAPTER 6

Conclusion

In this thesis, we detail various subgraph matching algorithms and approaches, and apply them to various problems in adversarial activity detection, network analysis, knowledge graph querying, and other applications.

In Chapter 2, we introduce a series of filtering methods that can be used to solve subgraph isomorphism problems in large-scale multiplex networks consisting of hundreds of thousands of nodes. Our filtering approaches reduce the search space using the node attributes, node-level statistics, topology, repeated-set, neighborhood, and elimination filters. 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 1.2.8). We can also validate whether each world node participates in at least one subgraph isomorphism, hence solving the Signal Node Set Problem (Definition 1.2.9).

We 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 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 of the proposed subgraph isomorphism problems. However, even in settings where we cannot solve the problems, such as the Great Britain Transportation and IvySys Versions 7 and 11, our methods reduced the size of the search space. Our results show that further narrowing down the candidates of template nodes is often hindered by an abundance of subgraph isomorphisms. Under such circumstances, leveraging more node attributes such as geospatial coordinates or timestamps can assist in making further progress, as shown in the Great Britain Transportation example.

Chapter 3 presents an analogous approach for the inexact attributed subgraph matching problem by defining the problem in terms of the graph edit distance. To search for optimal solutions, we first identify lower bounds on the cost of assigning individual template nodes to their candidates in order to restrict the search space. Using an approach analogous to constraint propagation, we first identify lower bounds on the cost of individual assignments. We then combine the cost bounds using linear sum assignment to obtain global cost bounds, which we use to guide a complete depth-first search for optimal solutions. We apply this approach to two attributed knowledge graph datasets, the PNNL AIDA V2.1.2 dataset describing political events in the Ukraine and the DBPedia knowledge graph database.

Chapter 4 extends the approach in Chapter 2 to handle potential errors in network alignment. We define the concepts of alignable subgraph isomorphism and minimal alignable subgraph isomorphism as they apply to aligned graphs. We modify the existing filtering system to tolerate candidates that fulfill the conditions of alignable subgraph isomorphism and alter the branching search to perform recursive alignments. Finally, we evaluate our algorithm on the AIDA Version 2.0.2 dataset.

One further research direction would be to extend the other filters from Chapter 2 to the aligned case, including the permutation, neighborhood, elimination, and validation filters. Another possibility is an extension of the inexact matching scheme described in Chapter 3 to the aligned graph context, defining alignment errors as part of the overall graph edit distance.

Finally, in Chapter 5, we describe a constraint propagation approach for path-based graph queries involving various constraints. To handle the variability in potential template size, we introduce the concept of template roles and detail the logical rules for performing constraint propagation among them. We show results on the PNNL COVID V3.2.1 dataset, identifying potential pathways connecting the SARS-COV-2 ACE2 receptor with cytokine storms.

This approach can be applied as a subroutine within a more general subgraph matching routine, where a single edge of the graph may instead be specified to be a constrained path. This is extremely important in the context of adversarial activity detection, as several activities, such as money transfers, material shipments, and communication chains, may be more generally represented as paths rather than fixed length chains of individual nodes or actors. One could also extend our approach to more general structures, such as connected groups or communities of nodes, which would allow us to more generally represent the role(s) of a group without having to specify the number of nodes.

REFERENCES

[ADH08] 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, 2008.

[ANR15] 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*, pp. 1–10. IEEE, 2015.

[BBB16] 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.

[Bc99] R. E. Burkard and E. Çela. "Linear assignment problems and extensions." In *Handbook of Combinatorial Optimization*, pp. 75–149. Springer, 1999.

[BCL16] 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*, pp. 1199–1214. ACM, 2016.

[BE85] R. Bar-Yehuda and S. Even. "A local-ratio theorem for approximating the weighted vertex cover problem." In *Analysis and Design of Algorithms for Combinatorial Problems*, volume 109 of *North-Holland Mathematics Studies*, pp. 27–45. Elsevier, 1985.

[Ber88] D. P. Bertsekas. "The auction algorithm: A distributed relaxation method for the assignment problem." *Annals of Operations Research*, **14**:105–123, 1988.

[BGP13] 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.

[BJU18] 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*, pp. 3986–3990. IEEE, 2018.

[CDM10] F. Celli, F. M. L. Di Lascio, M. Magnani, B. Pacelli, and L. Rossi. "Social network data and practices: the case of FriendFeed." In *Advances in Social Computing*, pp. 346–353. Springer, 2010.

[CFS03] D. Conte, P. Foggia, C. Sansone, and M. Vento. "Graph matching applications in pattern recognition and image processing." In *International Conference on Image Processing*, volume 2, pp. II–24. IEEE, 2003.

[CFS04] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento. “A (sub)graph isomorphism algorithm for matching large graphs.” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **26**(10):1367–1372, 2004.

[CFS17] V. Carletti, P. Foggia, A. Saggese, and M. Vento. “Introducing VF3: A new algorithm for subgraph isomorphism.” In *Graph-Based Representations in Pattern Recognition*, pp. 128–139. Springer, 2017.

[CFS18] 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, 2018.

[CFV15] V. Carletti, P. Foggia, and M. Vento. “VF2 plus: An improved version of VF2 for biological graphs.” In *Graph-Based Representations in Pattern Recognition*, pp. 168–177. Springer, 2015.

[CGZ13] 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**, 2013. Art. no. 1344.

[CPM18] 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*, pp. 3947–3950. IEEE, 2018.

[Cro16] D. F. Crouse. “On implementing 2D rectangular assignment algorithms.” *IEEE Transactions on Aerospace and Electronic Systems*, **52**(4):1679–1696, 2016.

[Dep15] Department for Transport. “National public transport data repository.” `https://data.gov.uk/dataset/d1f9e79f-d9db-44d0-b7b1-41c216fe5df6/national-public-transport-data-repository-nptdr`, 2015. Accessed: 2021-05-30.

[DLM13] M. De Domenico, A. Lima, P. Mougel, and M. Musolesi. “The anatomy of a scientific rumor.” *Scientific Reports*, **3**, 2013. Art. no. 2980.

[DWW21] A. P. Davis, T. C. Wiegers, J. Wiegers, C. J. Grondin, R. J. Johnson, D. Sciaky, and C. J. Mattingly. “CTD anatomy: Analyzing chemical-induced phenotypes and exposures from an anatomical perspective, with implications for environmental health studies.” *Current Research in Toxicology*, **2**:128–139, 2021.

[DZK18a] J. Douglas, B. Zimmerman, A. Kopylov, J. Xu, D. Sussman, and V. Lyzinski. “Metrics for evaluating network alignment.” In *11th ACM International Conference on Web Search and Data Mining, Workshop on Graph Techniques for Adversarial Activity Analytics*. ACM, 2018a.

[DZK18b] J. Douglas, B. Zimmerman, A. Kopylov, J. Xu, D. Sussman, and V. Lyzinski. “Metrics for evaluating network alignment.” In *GTA3 at WSDM*, 2018b.

[EKG13] A. Egozi, Y. Keller, and H. Guterman. "A probabilistic approach to spectral graph matching." *IEEE Trans. Pattern Anal. Mach. Intell.*, **35**(1):18–27, 2013.

[Fan12] W. Fan. "Graph pattern matching revised for social network analysis." In *Proceedings of the 15th International Conference on Database Theory*, pp. 8–21. ACM, 2012.

[GB15] R. Gallotti and M. Barthelemy. "The multilayer temporal network of public transport in Great Britain." *Scientific Data*, **2**, 2015. Art. no. 140056.

[GBS17] B. M. Gyori, J. A. Bachman, K. Subramanian, J. L. Muhlich, L. Galescu, and P. K. Sorger. "From word models to executable models of signaling networks using automated assembly." *Molecular Systems Biology*, **13**(11):954, 2017.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman and Co, 1979.

[GMN08] I. P. Gent, I. Miguel, and P. Nightingale. "Generalised arc consistency for the AllDifferent constraint: An empirical survey." *Artificial Intelligence*, **172**(18):1973–2000, 2008.

[GXT10] X. Gao, B. Xiao, D. Tao, and X. Li. "A survey of graph edit distance." *Pattern Analysis and Applications*, **13**:113–129, 2010.

[HK73] J. E. Hopcroft and R. M. Karp. "An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs." *SIAM Journal on Computing*, **2**(4):225–231, 1973.

[HLL13] W. Han, J. Lee, and J. Lee. "Turbo$_{\text{iso}}$: Towards ultrafast and robust subgraph isomorphism search in large graph databases." In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, pp. 337–348. ACM, 2013.

[Hoe01] W.-J. van Hoeve. "The AllDifferent constraint: A survey." http://www.andrew.cmu.edu/user/vanhoeve/papers/alldiff.pdf, 2001. Accessed: 2021-05-30.

[Hov20] E. Hovy. "Active interpretation of disparate alternatives (AIDA)." https://www.darpa.mil/program/active-interpretation-of-disparate-alternatives, 2020. Accessed: 2020-10-6.

[HS08] 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*, pp. 405–418. ACM, 2008.

[HUT20] S. Hojyo, M. Uchida, K. Tanaka, R. Hasebe, Y. Tanaka, M. Murakami, and T. Hirano. "How covid-19 induces cytokine storm with high mortality." *Inflammation and regeneration*, **40**(1):1–7, 2020.

[HWP03] 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*, pp. 549–552. IEEE, 2003.

[IIP16] V. Ingalalli, D. Ienco, and P. Poncelet. "SuMGra: Querying multigraphs via efficient indexing." In *Database and Expert Systems Applications*, pp. 387–401. Springer, 2016.

[JHW19] H. Jin, X. He, Y. Wang, H. Li, and A. L. Bertozzi. "Noisy subgraph isomorphisms on multiplex networks." In *2019 IEEE International Conference on Big Data*, pp. 4899–4905. IEEE, 2019.

[JM18] A. Jüttner and P. Madarasi. "VF2++— an improved subgraph isomorphism algorithm." *Discrete Applied Mathematics*, **242**:69–81, 2018.

[JV87] R. Jonker and A. Volgenant. "A shortest augmenting path algorithm for dense and sparse linear assignment problems." *Computing*, **38**:325–340, 1987.

[KAB14] M. Kivelä, A. Arenas, M. Barthelemy, J. P. Gleeson, Y. Moreno, and M. A. Porter. "Multilayer networks." *Journal of Complex Networks*, **2**(3):203–271, 2014.

[KMS16] L. Kotthoff, C. McCreesh, and C. Solnon. "Portfolios of subgraph isomorphism algorithms." In *Learning and Intelligent Optimization*, pp. 107–122. Springer, 2016.

[Knu00] D. E. Knuth. "Dancing links." *Millennial Perspectives in Computer Science*, pp. 187–214, 2000.

[KSD07] G. A. Korsah, A. Stentz, and M. B. Dias. "The dynamic Hungarian algorithm for the assignment problem with changing costs." Technical Report CMU-RI-TR-07-27, Carnegie Mellon University, 2007.

[KSG18] 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*, pp. 3959–3967. IEEE, 2018.

[Kuh55] H. W. Kuhn. "The Hungarian method for the assignment problem." *Naval Research Logistics Quarterly*, **2**(1-2):83–97, 1955.

[KX19] A. Kopylov and J. Xu. "Filtering strategies for inexact subgraph matching on noisy multiplex networks." In *2019 IEEE International Conference on Big Data*, pp. 4906–4912. IEEE, 2019.

[Lev66] V. I. Levenshtein. "Binary codes capable of correcting deletions, insertions, and reversals." **10**(8):707–710, 1966.

[LFF16] 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, 2016.

[LIJ15] J. Lehmann, R. Isele, M. Jakob, A. Jentzsch, D. Kontokostas, P. N. Mendes, S. Hellmann, M. Morsey, P. van Kleef, S. Auer, and C. Bizer. "DBpedia - A large-scale, multilingual knowledge base extracted from Wikipedia." *Semantic Web*, **6**:167–195, 2015.

[LV02] J. Larrosa and G. Valiente. "Constraint satisfaction algorithms for graph pattern matching." *Mathematical Structures in Computer Science*, **12**(4):403–422, 2002.

[LZ17] Y. Liang and P. Zhao. "Similarity search in graph databases: A multi-layered indexing approach." In *2017 IEEE 33rd International Conference on Data Engineering*, pp. 783–794. IEEE, 2017.

[Map] Mapbox. "Mapbox." `https://www.mapbox.com/`.

[MBF20] G. Micale, V. Bonnici, A. Ferro, D. Shasha, R. Giugno, and A. Pulvirenti. "MultiRI: Fast subgraph matching in labeled multigraphs." *arXiv preprint arXiv:2003.11546*, 2020.

[McG79] J. J. McGregor. "Relational consistency algorithms and their application in finding subgraph and graph isomorphisms." *Information Sciences*, **19**(3):229–250, 1979.

[MCT18] J. D. Moorman, Q. Chen, T. K. Tu, Z. M. Boyd, and A. L. Bertozzi. "Filtering methods for subgraph matching on multiplex networks." In *2018 IEEE International Conference on Big Data (Big Data)*, pp. 3980–3985, 2018.

[Moo21] J. Moorman. *Stochastic Optimization and Subgraph Search*. PhD thesis, UCLA, 2021.

[MP15] C. McCreesh and P. Prosser. "A parallel, backjumping subgraph isomorphism algorithm using supplemental graphs." In *Principles and Practice of Constraint Programming*, pp. 295–312. Springer, 2015.

[MPS18] 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**(1):723–759, 2018.

[MR11] M. Magnani and L. Rossi. "The ML-model for multi-layer social networks." In *2011 International Conference on Advances in Social Networks Analysis and Mining*, pp. 5–12. IEEE, 2011.

[MTC20] J. D. Moorman, T. K. Tu, Q. Chen, D. Yang, and X. He. "jdmoorman/uclasm: v0.2.0." Zenodo. `https://doi.org/10.5281/zenodo.4052353`, 2020.

[MTC21] J. D. Moorman, T. Tu, Q. Chen, X. He, and A. L. Bertozzi. "Subgraph matching on multiplex networks." *IEEE Transactions on Network Science and Engineering*, **8**(2):1367–1384, 2021.

[Mun57] J. Munkres. "Algorithms for the assignment and transportation problems." *Journal of the Society for Industrial and Applied Mathematics*, **5**(1):32–38, 1957.

[NMG10] A. Nuzzo, F. Mulas, M. Gabetta, E. Arbustini, B. Zupan, C. Larizza, and R. Bellazzi. "Text mining approaches for automated literature knowledge extraction and representation." In *MEDINFO 2010*, pp. 954–958. IOS Press, 2010.

[Nora] P. Norvig. "pytudes GitHub repository." `https://github.com/norvig/pytudes`, a. Accessed: 2021-05-30.

[Norb] P. Norvig. "Solving every Sudoku puzzle." `http://www.norvig.com/sudoku.html`, b. Accessed: 2021-05-30.

[NS17] C. Nabti and H. Seba. "Compact neighborhood index for subgraph queries in massive graphs." *arXiv preprint arXiv:1703.05547*, 2017.

[NYG19] T. Nguyen, D. Yang, Y. Ge, H. Li, and A. L. Bertozzi. "Applications of structural equivalence to subgraph isomorphism on multichannel multigraphs." In *2019 IEEE International Conference on Big Data*, pp. 4913–4920. IEEE, 2019.

[Ope] OpenStreetMap. "Openstreetmap." `https://www.openstreetmap.org/`.

[PPP17] S. Pilosof, M. A. Porter, M. Pascual, and S. Kéfi. "The multilayer nature of ecological networks." *Nature Ecology & Evolution*, **1**, 2017. Art. no. 0101.

[Pro] "Project Euler problem 96: Su Doku." `https://projecteuler.net/problem=96`. Accessed: 2021-05-30.

[PTT14] R. Pienta, A. Tamersoy, H. Tong, and D. H. Chau. "MAGE: Matching approximate patterns in richly-attributed graphs." In *2014 IEEE International Conference on Big Data*, pp. 585–590. IEEE, 2014.

[RDF14] "RDF 1.1 Concepts and Abstract Syntax." `https://www.w3.org/TR/rdf11-concepts/`, 2014. Accessed: 2021-8-17.

[Reg94] J. Régin. "A filtering algorithm for constraints of difference in CSPs." In *Proceedings of the 12th National Conference on Artificial Intelligence*, volume 1, pp. 362–367. AAAI, 1994.

[RKR17] 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*, pp. 1–12. IEEE, 2017.

[RW15] 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.

[Sai79] L. D. Sailer. "Structural equivalence: Meaning and definition, computation and application." *Social Networks*, **1**(1):73–90, 1978–1979.

[Sch18] C. Schwartz. "Modeling Adversarial Activity (MAA)." https://www.darpa.mil/program/modeling-adversarial-activity, 2018. Accessed: 2018-10-2.

[SF83] A. Sanfeliu and K. Fu. "A distance measure between attributed relational graphs for pattern recognition." *IEEE Transactions on Systems, Man, and Cybernetics*, **SMC-13**(3):353–362, 1983.

[Sim05] H. Simonis. "Sudoku as a constraint problem." In *Proceedings of the 4th International Workshop on Modelling and Reformulating Constraint Satisfaction Problems*, pp. 13–27, 2005.

[SL19] S. Sun and X. Luo. "Scaling up subgraph query processing with efficient subgraph matching." In *2019 IEEE 35th International Conference on Data Engineering*, pp. 220–231. IEEE, 2019.

[Sol10] C. Solnon. "AllDifferent-based filtering for subgraph isomorphism." *Artificial Intelligence*, **174**(12–13):850–864, 2010.

[SPA08] "SPARQL Query Language for RDF." `https://www.w3.org/TR/rdf-sparql-query/`, 2008. Accessed: 2021-8-17.

[SPP20] 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*, **42**(11):2887–2900, 2020.

[SR21] P. Shetty and R. Ramprasad. "Automated knowledge extraction from polymer literature using natural language processing." *iScience*, **24**(1):101922, 2021.

[Ste05] G. Stertenbrink. "top95." `http://magictour.free.fr/top95`, 2005. Accessed: 2021-09-06.

[SW05] T. Schank and D. Wagner. "Finding, counting and listing all triangles in large graphs, an experimental study." In *Experimental and Efficient Algorithms*, pp. 606–609. Springer, 2005.

[SWW12] Z. Sun, H. Wang, H. Wang, B. Shao, and J. Li. "Efficient subgraph matching on billion node graphs." *Proceedings of the VLDB Endowment*, **5**(9):788—799, 2012.

[SZL08] 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.

[TMY20] T. K. Tu, J. D. Moorman, D. Yang, Q. Chen, and A. L. Bertozzi. "Inexact attributed subgraph matching." In *2020 IEEE International Conference on Big Data*, pp. 2575–2582. IEEE, 2020.

[TRS13] G. Tauer, R. Rudnicki, and M. Sudit. "Approximate SPARQL for error tolerant queries on the DBpedia knowledge base." In *Proceedings of the 16th International Conference on Information Fusion*, pp. 850–856. IEEE, 2013.

[TY20] T. K. Tu and D. Yang. "Fault-tolerant subgraph matching on aligned networks." In *2020 IEEE International Conference on Big Data*, pp. 2569–2574. IEEE, 2020.

[Ull76] J. R. Ullmann. "An algorithm for subgraph isomorphism." *Journal of the ACM*, **23**(1):31–42, 1976.

[Ull10] J. R. Ullmann. "Bit-vector algorithms for binary constraint satisfaction and subgraph isomorphism." *Journal of Experimental Algorithmics*, **15**, 2010. Art. no. 1.6.

[Ver79] L. M. Verbrugge. "Multiplexity in adult friendships." *Social Forces*, **57**(4):1286–1309, 1979.

[WLC20] L. L. Wang, K. Lo, Y. Chandrasekhar, R. Reas, J. Yang, D. Eide, K. Funk, R. Kinney, Z. Liu, W. Merrill, et al. "CORD-19: The COVID-19 Open Research Dataset." *ArXiv*, 2020.

[WLW20] Q. Wang, M. Li, X. Wang, N. Parulian, G. Han, J. Ma, J. Tu, Y. Lin, H. Zhang, W. Liu, et al. "COVID-19 literature knowledge graph construction and drug repurposing report generation." *arXiv preprint arXiv:2007.00576*, 2020.

[WTD19] L. Weston, V. Tshitoyan, J. Dagdelen, O. Kononova, A. Trewartha, K. A. Persson, G. Ceder, and A. Jain. "Named entity recognition and normalization applied to large-scale information extraction from the materials science literature." *Journal of Chemical Information and Modeling*, **59**(9):3692–3702, 2019. PMID: 31361962.

[XTL18] J. Xu, H. Tong, T. Lu, J. He, and N. Bliss. "GTA^3 2018: Workshop on graph techniques for adversarial activity analytics." In *Proceedings of the 11th ACM International Conference on Web Search and Data Mining*, p. 803. ACM, 2018.

[YG13] L. Yartseva and M. Grossglauser. "On the performance of percolation graph matching." In *Proceedings of the First ACM Conference on Online Social Networks*, pp. 119–130. ACM, 2013.

[YH02] X. Yan and J. Han. "gspan: Graph-based substructure pattern mining." In *2002 IEEE International Conference on Data Mining*, pp. 721–724. IEEE, 2002.

[ZBV09] 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, 2009.

[ZDS10] S. Zampelli, Y. Deville, and C. Solnon. "Solving subgraph isomorphism problems with constraint programming." *Constraints*, **15**:327–353, 2010.

[ZH10] P. Zhao and J. Han. "On graph query optimization in large networks." *Proceedings of the VLDB Endowment*, **3**(1–2):340–351, 2010.

[ZL17] M. Zitnik and J. Leskovec. "Predicting multicellular function through multi-layer tissue networks." *Bioinformatics*, **33**(14):190–198, 2017.

[ZLY09] 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*, pp. 192–203. ACM, 2009.

[ZMC11] 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.

[ZPM21] J. Zucker, K. Paneri, S. Mohammad-Taheri, S. Bhargava, P. Kolambkar, C. Bakker, J. Teuton, C. T. Hoyt, K. Oxford, R. Ness, et al. "Leveraging structured biological knowledge for counterfactual inference: A case study of viral pathogenesis." *IEEE Transactions on Big Data*, **7**(1):25–37, 2021.