Title
Exploiting Symmetry in Subgraph Isomorphism and Formulating Neural Network Constrained Optimization Problems

Author
Yang, Dominic

Publication Date
2023

Peer reviewed|Thesis/dissertation

UNIVERSITY OF CALIFORNIA
Los Angeles

Exploiting Symmetry in Subgraph Isomorphism and Formulating Neural Network Constrained Optimization Problems

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

by

Dominic Tianli Yang

2023

ABSTRACT OF THE DISSERTATION

Exploiting Symmetry in Subgraph Isomorphism
and Formulating Neural Network Constrained
Optimization Problems

by

Dominic Tianli Yang
Doctor of Philosophy in Mathematics
University of California, Los Angeles, 2023
Professor Andrea Bertozzi, Chair

In this dissertation, I will cover two distinct topics. In the first part, I will discuss the subgraph isomorphism problem which has a broad range of applications from pattern recognition and bioinformatics to compilers circuit design. In combinatorial problems such as these, symmetry, if unaccounted for, can confound standard solvers leading to significant amounts of redundant work being done. I will introduce multiple approaches for dynamically exploiting this symmetry to accelerate solvers as well as to compactly characterize the set of solutions. I rigorously establish conditions under which from one subgraph isomorphism, potentially exponentially many more may be generated. I also demonstrate through empirical assessment that incorporating symmetry into subgraph search can provide significant reductions in search time and produce many more solutions.

In the presence of these huge solution spaces, there still remains the work of determining the actual subgraph isomorphism of interest. In an approach inspired by active learning, I will introduce the problem of querying template vertices for more information to uniquely

identify solutions. Multiple different strategies for querying vertices which also incorporate symmetry are established and assessed by rigorous analysis and comprehensive experiments.

In the second part, I will discuss methods for formulating neural network constrained optimization problems. With the advent of machine learning, neural networks have been used as a surrogate for complicated physical models, and the incorporation of these networks into optimization problems is a challenge. I will present a direct embedding approach, a mixed integer program, and a nonlinear program with complementarity constraints as methods for formulating these problems to be used with standard optimization solvers. I establish key properties of each methods, and through experiments, I assess the advantages and disadvantages for each approach.

The dissertation of Dominic Tianli Yang is approved.

Christopher R. Anderson

Deanna M. Hunter

Mason Alexander Porter

Sven Leyffer

Andrea Bertozzi, Committee Chair

University of California, Los Angeles

2023

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I would like to thank first my advisor, Andrea Bertozzi, for her help in the duration of my studies. I would also like to thank all of the other members of my research group who have given me advice, ideas, and support. Thank you specifically to Jacob Moorman, Thomas Tu, and Yurun Ge. I would also like to extend thanks to Sven Leyffer who has offered a great deal of support to me in my research.

I also would like to extend thanks to all of the graduate student friends who I have made during my time at UCLA. In particular, I want to thank Benjamin Bowman, Joel Barnett, Patrick Hiatt, Louis Esser, Jerry, Luo, Xia Li, and Kaiyan Peng. I also want to thank all of my officemates through the years, Eilon Tzur, Nicholas Boschert, and Yotam Yaniv. Thank you also to Jason Brown, James Chapman, Michael Johnson, Allison Schiffman for being good friends. Thank you also to Sandy Kim for being a good friend through Covid and since. Thank you all for keeping me sane in these years.

I want to thank my friends from UC Davis whose friendship I think is invaluable. Thank you to Andrew Jackson, Alejandro Hernandez, Vidush Vishwanath, Kamal Gill, Kyla Broderick, Sheila Kulkarni, Amelia Freije, Daniel Loran, Sam Truong, Hershel Shah, and Bianca Rivera. I am grateful for the many positive memories that I have of you all.

I want to thank especially my mother, my father, my brother, and all of my grandparents. It is only with the support of my family that I could have gotten to the point where I am today. I love you all dearly. I hope that through my life I will be able to show that appreciation.

Chapter 3 is a version of [YGN23] and is joint work with Yurun Ge, Thien Nguyen, Jacob Moorman, Denali Molitor, and Prof. Andrea Bertozzi. Andrea Bertozzi supervised the project. Thien Nguyen introduced a version of candidate equivalence in prior work, and I expanded on these concepts for use in subgraph matching, provided rigorous proofs, created the method for compactly representing the solution space, and adapted another solver for use

in experiments. Yurun Ge developed the method for accelerating subgraph search for solving the all-different aspect of search, and contributed to the code, experiments, and visualizations for the multichannel datasets. Jacob Moorman and Denali Molitor contributed to the code.

Chapter 4 is a version of [GYB23] and is joint work with Yurun Ge and Prof. Andrea Bertozzi. Andrea Bertozzi supervised the project. Yurun Ge formalized the concept of active learning in subgraph matching, provided the proof for NP-completeness, and developed methods and code for solving the active learning query problem and performed several experiments. I established the reduction to the set cover problem, introduced the theoretical variants of the problem, and also devised several methods and experiments for this project.

Chapter 5 is a version of [YBL22] which is joint work with Sven Leyffer and Prasanna Balaprakash. Sven Leyffer supervised the project and provided ideas and developed theory. Prasanna Balaprakash provided a use case for the problem. I contributed to the development of the theory of surrogate optimization and set up the experiments analyzing the various formulations.

The work done in Chapter 3 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. The work done in Chapter 4 is based on research supported by the National Science Foundation (Grant DMS-2027277). The work done in Chapter 5 is based on research supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, under Contract DE-AC02-06CH11357. This work was also supported by the U.S. Department of Energy through grant DE-FG02-05ER25694. This was also supported through an NSF-MSGI fellowship.

VITA

2016-2017	Student Assistant, UC Davis Graduate School of Management, UC Davis, Davis, California.
2017-2019	R & D Intern, Discrete Optimization Group, Sandia National Laboratories, Albuquerque, New Mexico
2018	B.S. (Mathematics) and B.S. (Computer Science), UC Davis.
2018–2023	Teaching Assistant, Mathematics Department, UCLA.
2019–2023	Research Assistant, Mathematics Department, UCLA.
2020–2021	NSF MGSI Intern / Givens Scholar, Argonne National Laboratory, Lemont, Illinois
2022	Applied Machine Learning Fellow, Los Alamos National Laboratory, Los Alamos, New Mexico

PUBLICATIONS

Yurun Ge, Dominic Yang, Andrea Bertozzi. "Iterative Active Learning Strategies for the Subgraph Matching Problem." *Submitted*.

Dominic Yang, Yurun Ge, Thien Nguyen, Jacob Moorman, Denali Molitor, and Andrea Bertozzi. "Structural Equivalence in Subgraph Matching." IEEE Transactions on Network Science and Engineering (2023).

Dominic Yang, Prasanna Balaprakash, and Sven Leyffer. "Modeling design and control problems involving neural network surrogates." Computational Optimization and Applications (2022): 1-42.

Thomas Tu, Jacob Moorman, Dominic Yang, Qinyi Chen, and Andrea Bertozzi. "Inexact Attributed Subgraph Matching." Fourth workshop on Graph Techniques for Adversarial Activity Analytics (GTA3 4.0), IEEE BIG DATA conference, Los Angeles, Dec 10, 2020.

Thomas Tu, Dominic Yang. "Fault-tolerant Subgraph Matching on Aligned Networks." Fourth workshop on Graph Techniques for Adversarial Activity Analytics (GTA3 4.0), IEEE BIG DATA conference, Los Angeles, Dec 10, 2020.

Jesùs A. De Loera, Thomas A. Hogan, Deborah Oliveros, and Dominic Yang. "Tverberg Type Theorems with Altered Intersection Patterns (Nerves)." Discrete & Computational Geometry (2020): 1-22.

Thien Nguyen, Dominic Yang, Yurun Ge, Hao Li, and Andrea Bertozzi. "Applications of Structural Equivalence to Subgraph Isomorphism on Multichannel Multigraphs." Third workshop on Graph Techniques for Adversarial Activity Analytics (GTA3 3.0), IEEE BIG DATA conference, Los Angeles, Dec 9, 2019.

Benjamin A. Rachunok, Andrea Staid, Jean-Paul Watson, David L. Woodruff and Dominic Yang. "Stochastic Unit Commitment Performance Considering Monte Carlo Wind Power Scenarios." 2018 IEEE International Conference on Probabilistic Methods Applied to Power Systems (PMAPS), Boise, June 2018.

CHAPTER 1

Introduction

In this thesis, we will be discussing two distinct topics: efficient algorithms for finding subgraph isomorphisms and the implementation of surrogate neural network models into generic optimization problems. The subgraph isomorphism problem, also known as the subgraph matching problem, refers to the task of discovering a small template graph as a subgraph of a larger world graph. In the past few decades, there has been a great influx of giant datasets of graphs representing all kinds of interactions between entities including communication patterns, social networks, biological structures, and countless other systems. Subgraph isomorphism checking has emerged as a tool for analyzing these graphs to recognize certain patterns of interaction where these patterns might be molecules within a larger protein structure, a criminal ring in a financial transaction network, or segments of an image. As a result of the myriad applications of finding subgraph isomorphisms, the development of efficient algorithms for detecting subgraph isomorphisms has become a burgeoning field of research.

In Chapter 2, we formally define the subgraph matching problem and give an overview of various different approaches to finding subgraph isomorphisms. We elaborate on a central tool for subgraph search, the tree search routine, and we identify specific optimizations that have been made to this approach for enumerating subgraph isomorphisms. We also discuss other distinct approaches: modeling the subgraph matching problem as a constraint program and developing graph indexes which can potentially offer a quick way to filter out problems for which there can be no match. We also briefly discuss the inexact subgraph

matching problem, a related problem where we look for subgraphs which approximately match a template.

In Chapter 3, we analyze the impact of symmetry on standard subgraph search algorithms and assess how we can exploit it to improve algorithm efficiency. The presence of a high degree of symmetry in a subgraph isomorphism problem can result in a combinatorial explosion in the count of matchings, and if unaccounted for, can confound standard subgraph isomorphism solvers. In this chapter, we characterize different notions of symmetry and explain how to exploit symmetry to avoid redundant work, accelerate subgraph search, and compactly represent the set of all subgraph isomorphisms. For these highly symmetric examples, we argue that it is essential to address the impact of symmetry on subgraph search and demonstrate how to easily adapt standard tree search algorithms to incorporate symmetry.

In Chapter 4, with inspiration from the field of active learning, we pose the question: if we could query vertices in our graphs for more information, which queries would help us the most in finding a subgraph isomorphism. This problem arises in contexts where there are an abundance of potential subgraph isomorphisms, and we wish to better characterize our template graph in order to narrow down which of the subgraph isomorphisms we are actually interested in. We formalize this problem and present a collection of criteria for determining which vertices to query so as to determine an isomorphism in the least number of queries. We also demonstrate how we can incorporate symmetry into the query procedure using techniques developed in Chapter 3.

In Chapter 5, we introduce the second topic of our thesis: implementing surrogate neural network models into optimization problems. This problem is motivated by the development of highly computationally intensive models representing complicated physics simulations. Using these models, we would like to solve optimization problems incorporating their behavior. For example, we consider a model of an automobile engine predicting emissions and we wish to pose an optimization problem which designs an engine to minimize these emissions.

We cannot typically incorporate these models directly into optimization problems owing to their high computational costs, and as an approximation, we instead consider neural network surrogates trained using the data from this simulation. We formalize three different approaches of directly incorporating neural networks into optimization problems: via direct embedding, as an integer program, and as a nonlinear program with complementarity constraints.

CHAPTER 2

The Subgraph Isomorphism Problem

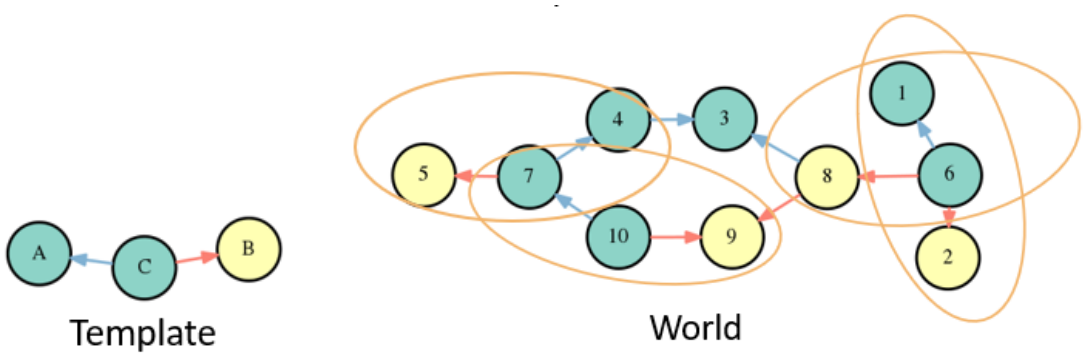

Figure 2.1: Example subgraph matching problem. The colors of nodes and edges correspond to different node labels and channels, respectively. There are four subgraph isomorphisms corresponding to mapping the template nodes (A, C, B) to each of the four circled sets of nodes $(4, 7, 5), (7, 10, 9), (1, 6, 8), (1, 6, 2)$.

2.1 Defining the Problem

The subgraph isomorphism problem (also called the subgraph matching problem) specifies a small graph (the **template**) to find as a subgraph within a larger (**world**) graph. We can see examples of multiple subgraph isomorphisms present in Figure 2.1 where we wish to find nodes which match both the adjacency and label structure of our template graph.

Formally, we have two graphs, a template, $\mathcal{G}_t = (\mathcal{V}_t, \mathcal{E}_t)$, and a world, $\mathcal{G}_w = (\mathcal{V}_w, \mathcal{E}_w)$. In this thesis, unless otherwise stated, we assume the graphs under consideration are simple

directed graphs so that for a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, $\mathcal{E} \subset \mathcal{V} \times \mathcal{V}$, and we forbid any multiple edges or self loops. The concepts discussed in the thesis extend naturally to more complicated graphs, but for simplicity of exposition, we consider solely simple directed graphs. In this context, a subgraph isomorphism is a function $f : \mathcal{V}_t \to \mathcal{V}_w$ that is both injective and respects the adjacency structure of $\mathcal{G}_t$. For the latter property to hold, we require that if $(t_1, t_2) \in \mathcal{E}_t$, then we must have $(f(t_1), f(t_2)) \in \mathcal{E}_t$. If this is true, we say that f is **edge-preserving**. We define subgraph isomorphism as follows:

Definition 1. *Given a template $\mathcal{G}_t = (\mathcal{V}_t, \mathcal{E}_t)$ and a world $\mathcal{G}_w = (\mathcal{V}_w, \mathcal{E}_w)$, a map $f : V_T \to V_W$ is a **subgraph isomorphism** (SI) (or **subgraph matching**) if and only if f is injective and edge-preserving.*

If in addition, the vertices and edges in our template and world also have labels, we also impose the requirement that our mapping f be **label-preserving**. Formally, if we have a vertex label function $\mathcal{L}_V$ and an edge label function $\mathcal{L}_E$ which maps vertices and edges to a label set L, then we require that for any vertex $v \in \mathcal{V}_t$, $\mathcal{L}_V(v) = \mathcal{L}_V(f(v))$, and for any edge $(u, v) \in \mathcal{E}_t$, $\mathcal{L}_E((u, v)) = \mathcal{L}_E((f(u), f(v)))$. If there is at least one subgraph isomorphism, we call the problem **satisfiable**. We refer to the set of all subgraph isomorphisms between a template $\mathcal{G}_t$ and a world $\mathcal{G}_w$ using the notation $\mathcal{F}(\mathcal{G}_t, \mathcal{G}_w)$.

Throughout this thesis, we will use the terms subgraph isomorphism and subgraph matching interchangeably. Also, given $\mathcal{G}_t$ and $\mathcal{G}_w$ and the problem of finding subgraph isomorphism, we will call any individual subgraph isomorphism a solution and we call algorithms for finding them solvers. Related terms are **subgraph homomorphism** or **edge-preserving mapping (EPM)** which relaxes the injectivity requirement, and **induced subgraph isomorphism** which also requires the map to be non-edge-preserving ($(u, v) \notin \mathcal{E}_t$ if and only if $(f(u), f(v)) \notin \mathcal{E}_w$).

Simply finding a subgraph isomorphism is NP-complete [GJ02], suggesting that there is no algorithm that efficiently finds all subgraph isomorphisms on all graphs. In spite of this,

Problem	Description
Subgraph Isomorphism Problem (SIP)	Determine if a SI exists
Signal Node Set Problem (SNSP)	Find all world vertices which participate in a SI
Minimal Candidate Sets Problem (MCSP)	For any $v \in \mathcal{V}_t$, find all $w \in \mathcal{V}_w$ where there exists $f \in \mathcal{F}(\mathcal{G}_t, \mathcal{G}_w)$ with $f(v) = w$
Subgraph Isomorphism Counting Problem (SICP)	Count the number of SIs
Subgraph Matching Problem (SMP)	Enumerate all SIs

Table 2.1: Various problems involving subgraph isomorphisms (from [MTC21])

significant progress has been made in the development of algorithms for detecting subgraph isomorphisms [MPT20, HLL13, BCL16, HDM14].

There are a variety of different problems which involve subgraph isomorphisms as cataloged in [MTC21]. We reproduce these problems and list them roughly in order of difficulty in Table 2.1. In this thesis, we will mostly be interested in the last two problems, the subgraph isomorphism counting problem (SICP) and the subgraph matching problem (SMP). We focus on these problems because we are primarily interested in the characterization of the set of all subgraph isomorphisms, $\mathcal{F}(\mathcal{G}_t, \mathcal{G}_w)$. We reason that for many problems, often there are an abundance of subgraph isomorphisms so that simply finding a single matching provides a very incomplete picture of the space of all solutions. If we had a template representing a network of transactions in a financial crime, knowing that there are millions of matches would inform us that our template is perhaps underspecified and we would need more information to identify the true matching. Had we found only one isomorphism, there would be an extremely high probability that this would be a false positive, and we may have

falsely accused someone of a crime. Having a full characterization of the solution space will establish a better picture of the problem we desire to solve.

2.1.1 Terminology

We now define some related terms from graph theory which will recur throughout this thesis. We say that two vertices u and v are **adjacent** if $(u, v) \in \mathcal{E}$ or $(v, u) \in \mathcal{E}$ and that the edge (u, v) is *incident* to vertices u and v and no other vertices. The **neighborhood** of a vertex v, denoted $N(v)$, is the set of **neighbors** of v, i.e., the vertices adjacent to v. The **degree** of v, $\deg(v)$, is the size of the neighborhood of v, $|N(v)|$. The **indegree** and **outdegree** for v are defined to be the number of other vertices u for which (u, v) is an edge and (v, u) is an edge, respectively. A **leaf** is any vertex with degree 1.

A **path** is a sequence of vertices $v_1, \ldots, v_n$ such that v_i is adjacent to v_{i+1} for all $i = 1, \ldots, n-1$. If $v_1 = v_n$, we say that the path is a **cycle**. A graph is **connected** if for any two vertices u and v, there is a path starting at u and ending at v. A **tree** is a connected graph which has no cycles. A **complete graph** is a graph where every two vertices u and v are adjacent.

We also establish some notation which will be useful for talking about the subgraph isomorphism problem. We call a world vertex w a **candidate** for template vertex t if we believe that there exists a subgraph isomorphism mapping t to w (meaning in an algorithm we have not determined there is no subgraph isomorphism mapping t to w). The set of all candidates for t is the **candidate set** of t and is denoted $C(t)$. To explicitly list specific subgraph isomorphisms, we use the following notation: If we have template vertices $t_1, \ldots, t_n$, and a subgraph isomorphism f, we write the subgraph isomorphism $\{t_1 \to f(t_1), \ldots, t_n \to f(t_n)\}$. For example, the subgraph isomorphism in Figure 2.1 which maps A to 4, B to 5, and C to 7, we will write this as $\{A \to 4, B \to 5, C \to 7\}$.

2.1.2 Applications of Subgraph Isomorphism

There are myriad applications of finding subgraph isomorphisms that have emerged over the years. Some of the most common applications come from pattern recognition where research problems involving subgraph isomorphism have included handwriting recognition [SF83], face recognition [WKK97], biomedical imaging [DGG92], and analyzing geospatial data [WSM14]. More recently, subgraph matching arises as a component in motif discovery [MGF18, RS14], where frequent subgraphs are uncovered for graph analysis in domains including social networks and biochemical data.

One very active area for subgraph isomorphism of which the author has been involves isomorphism algorithm is that of detecting adversarial activity as part of the DARPA-MAA program [KSG18, BJU18, CPM18, MTC21, SPP19, JHW19]. For these problems, the graphs of interest are patterns of communication, transactions, and other interactions between people and the goal of the program was to detect certain specific templates modeling precise interaction patterns.

Some novel applications of subgraph isomorphism involve graphs which represent programs programs and subgraph isomorphism has played a role in malware detection [BMM06], compilation [MF12], and plagiarism detection [LCH06]. Subgraph isomophism has also found a place in solving sudoku problems [MTC21] and generating crossword puzzles [Moo21].

Additionally, subgraph matching is relevant in knowledge graph searches, wherein incomplete factual statements are completed by querying a knowledge database [ABK07, TMY20]. Networks are present in many applications; hence, the ability to detect interesting structures, i.e., subgraphs, apparent in the networks bears great importance.

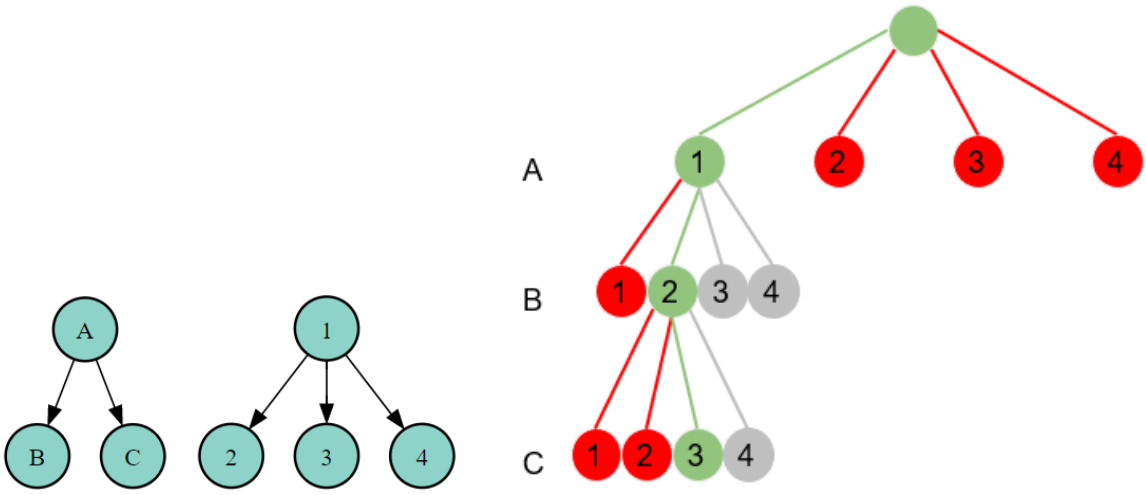

Figure 2.2: A simple template graph (left), world graph (center) and representation of a sample tree search algorithm (right). Each level of the tree on the right corresponds to a stage where one template vertex is assigned to a candidate world vertex. This green path in the tree represents the subgraph isomorphism created by matching A to 1, B to 2, and C to 3. The red nodes in the tree indicate assignments which cannot possibly lead to solutions and may be pruned.

2.2 Approaches for Finding Subgraph Isomorphisms

Since the problem was first established by Ullman in 1976 [Ull76], many different approaches for subgraph isomorphism have been developed. The surveys [FPV14], [CFS07], and [EDS16] explain the broad variety of techniques used for finding the isomorphisms in the fifty years since the problem was formulated. We explore in detail three different classes of subgraph isomorphism finding algorithms: tree search algorithms, constraint programming approaches, and graph indexing approaches. We discuss prominent solvers arising from each of these paradigms.

2.2.1 Tree Search Approaches

A tree search approach to subgraph isomorphism represents perhaps the most natural method to finding subgraph isomorphisms. The general idea behind the method is to gradually build up an assignment between template vertices and world vertices. We first explain the approach by example and then present a general purpose algorithm. If we consider the graphs in Figure 2.2, we may attempt to find a solution by mapping template vertex A to world vertex 1. We then proceed in turn by mapping vertices B to 2 and C to 3 out of all the possible candidate world vertices 1, 2, 3, and 4, and we have found one solution $\{A \to 1, B \to 2, C \to 3\}$. If we want to find another solution, we can unmap C to 3 and instead map C to 4 to produce yet another solution. We can represent this search procedure using a tree structure where each level of the tree corresponds to a stage where a particular template vertex is assigned to a world vertex. The process of finding all subgraph isomorphisms by building up partial mappings can be represented as a traversal of this search tree.

Navigating the entire search tree however is computationally intractable given there are exponentially many nodes in the tree. To reduce the amount of work done in the search, it is beneficial to avoid any assignment of template vertex to world vertex which we know cannot lead to a subgraph matching. For example, once we map A to 1, we cannot map B to 1 as this would violate the injectivity requirement of subgraph isomorphisms. In addition, we cannot map A to any of the vertices 2, 3, or 4 as the degree of these world vertices is smaller than that of A so that there is no way the mapping can satisfy the edge-preserving requirement. This process of ruling out candidate world vertices for a given template vertex is known as **filtering** or **pruning** and is essential for efficient discovery of subgraph isomorphisms.

Other methods of optimizing the search include intelligently choosing which template vertices we assign first. For example, many tree search algorithms [CFS04b, RS14] prioritize mapping high degree vertices first as this typically enables the filtering of more candidate vertices out which minimizes the size of the search space.

A skeleton of a basic tree search algorithm is presented in Algorithm 1. This is naturally represented as a recursive routine and the body of the function represents how we would go through the process of finding an assignment for one template vertex. We start with some representation of a partial match as well as a collection of the candidate world vertices for every given template vertex. In lines 2-4, if we have matched all template vertices, we report a solution and are done. In line 5, we apply filters to eliminate as many candidates as possible to shrink the search space. In lines 6 and 7 we pick a template vertex and generate possible candidate world vertices for the template vertex. Then in lines 8-11, we sequentially try mapping our template vertex to each candidate world vertex, explore the remainder of the search tree with this assignment, and then undoing this match once this search has been completed. In this fashion, we can search the entire tree.

Algorithm 1 Generic routine for a tree search

```
1:  function TreeSearch(Template G_t, World G_w,partial_match, cands)
2:      if MatchComplete(partial_match) then
3:          ReportMatch(partial_match)
4:          return
5:      ApplyFilters(G_t, G_w, partial_match, cands)
6:      Let u = GetNextTemplateVertex(G_t)
7:      Let ws = GenerateWorldVertices(G_w, u, cands)
8:      for v in ws do
9:          partial_match.match(u, v)
10:         TreeSearch(G_t, G_w, partial_match, cands)
11:         partial_match.unmatch(u, v)
12:     return
```

We now give an overview of the different subgraph isomorphism algorithms that have emerged over the years which employ a tree search. The first significant algorithm for finding subgraph isomorphisms proposed by Ullmann in 1976 [Ull76] was a tree search algorithm.

This algorithm implemented this basic recursive algorithm along with a simple filtering rule: if an unmatched template vertex t' is adjacent to a template vertex t which is mapped to world vertex w, its candidates in the world must be adjacent to w. The widely used VF2 algorithm [CFS04b] improves on this by imposing a match ordering that favors template vertices adjacent to already-matched template vertices and also adds filtering rules based on the degrees of vertices. VF2+ [CFV15], VF2++ [JM18], and VF3 [CFS17] are more recent expansions on the VF2 algorithm. The RI algorithm [BGP13] is another tree search approach which proposes a static order for how template vertices are assigned by prioritizing the dense core of the template graph.

Other algorithms which are built on this paradigm include QuickSI [SZL08], GraphQL [HS08], and SPath [ZH10]. QuickSI derives its order for matching template vertices from spanning tree representations of the template and world which have been flattened into arrays, and it uses a modification of the classical Ullman algorithm to implement the tree search. GraphQL develops a query language where a graph is composed of various operations joining nodes together and performs matching in an analogous fashion to database queries. The SPath algorithm establishes a "neighborhood signature" composed of neighborhoods of vertices at a fixed distance from a node using this signature as a tool for filtering out matches. These methods and more are motivated by the creation of large databases of graphs and these algorithms are components of these systems. We talk about graph databases and graph indexing methods for databases in Section 2.2.3.

Some approaches minimize the amount of work done in the tree search by exploiting symmetry of the template and world graphs to avoid redundant work done in the tree search. These techniques rely on the realization that if we have one solution, and we are aware of symmetry in the solution space, we can use this knowledge to produce additional solutions without needing to navigate the full search tree. The TurboISO algorithm [HLL13] established neighborhood equivalence classes which are groups of template vertices which can be interchanged in a subgraph isomorphism to produce a new subgraph isomorphism.

The BoostISO algorithm [RW15] applies a similar concept to the world graph. Symmetry as a tool for greatly accelerating the speed of subgraph search is explored in depth in Chapter 3.

The latest algorithms for subgraph isomorphism using tree search to utilize more information about both template and world to optimize search order. The CFL-match algorithm [BCL16] provides a decomposition of the template into a dense core, a forest of trees and leaves alongside a data structure to store candidates in a spanning tree of a template. The DAF algorithm [HKG19] improves upon this approach using a directed acyclic subgraph of the template to aid in vertex ordering as well as a more complete data structure for storing candidates.

2.2.2 Constraint Programming

In an approach inspired by artificial intelligence, several researchers have posed the problem of finding subgraph isomorphisms as a constraint satisfaction problem (also known as a constraint program (CP)). Constraint programs are satisfaction problems where there is a set of **variables** $\mathcal{X} = \{x_1, \ldots, x_n\}$, a set of **domains** for each variable $\mathcal{D} = \{D_1, \ldots, D_n\}$ which dictate **values** each variable may take, and a set of **constraints** $\mathcal{C} = \{C_1, \ldots, C_m\}$ which further restrict the allowed instantiation of variables. The domains D are directly analogous to the candidate sets discussed in tree search approaches. The goal of any constraint program is to find a solution, i.e., a value $v_i \in D_i$ for each variable x_i which satisfies all constraints. Formally, each constraint is defined for a subset of variables $x_{i_1}, x_{i_2}, \ldots, x_{i_k}$ and specifies which combinations of values from $D_{i_1} \times \cdots \times D_{i_k}$ are allowed, either explicitly as a list or as given by a general rule. The number of variables, k, a constraint C restricts is called the **arity** of C. Constraints with $k = 1$ and $k = 2$ are called unary and binary, respectively.

A standard way to frame the subgraph isomorphism problem as a constraint program involves introducing one variable x_v for each vertex $v \in \mathcal{V}_t$ whose corresponding domain D_v is given by the set of world vertices $\mathcal{V}_w$. Two different sets of constraints are produced to

enforce both the injectivity and edge-preserving conditions for subgraph isomorphisms. To impose that the mapping is injective, we have the following set of binary constraints:

$$\forall u, v \in \mathcal{V}_t, u \neq v \implies x_u \neq x_v. \tag{2.1}$$

This constraint can be generalized to k variables using what is called an *alldifferent* constraint which imposes that the assignments of each of these k variables is to k unique values. To impose that the mapping is edge-preserving, we have the associated set of binary constraints:

$$\forall u, v \in \mathcal{V}_t, (u, v) \in \mathcal{E}_t \implies (x_u, x_v) \in \mathcal{E}_w. \tag{2.2}$$

If our graphs are labelled, we can impose the following set of unary constraints to ensure that the mapping is label-preserving:

$$\forall u \in \mathcal{V}_t, \mathcal{L}(x_u) = \mathcal{L}(u). \tag{2.3}$$

Approaches for solving general constraint programs revolve around a technique called *constraint propagation* which aims to reduce the size of the domains of each variable to values which can participate in a solution. We say that a variable x is **node consistent** if each value in its domain D_x satisfies all unary constraints. In the context of subgraph isomorphism, one such constraint might be that template nodes are mapped to world nodes of the same label, and we would say a variable for a template node v is node consistent if its domain consisted solely of nodes of the same label. A related notion is arc consistency which is defined for binary constraints. We say a variable x_1 is **arc consistent** with respect to x_2 if for each binary constraint C defined for variables x_1 and x_2, for any value $v_1 \in D_{x_1}$, there is a corresponding value $v_2 \in D_{x_2}$ which will satisfy constraint C. More complicated forms of consistency include path consistency which require the consistent instantiation of sequences of variables. Algorithms for solving constraint programs generally proceed by a tree search exploring possible instantiations of each variable (in a very similar fashion as described in the prior section) and then enforce these notions of consistency (using for example, AC-3 algorithm for arc consistency [Mac77]) at each step to reduce the number of candidates to

try in the tree search. Highly efficient and general-purpose solvers (e.g., GeCode [SS04] or IBM's ILOG CP optimizer [LRS18]) have been created for constraint programs given their very general form and broad applicability to a wide variety of problems.

The first approach which invoked the constraint programming paradigm was given by McGregor [McG79] not long after Ullman's original approach. In this work, McGregor directly integrated these methods of constraint propagation for arc and path consistency into a backtracking tree search. He later expands this approach to the maximum common subgraph problem in [McG79]. Since then, the research into subgraph isomorphism algorithms in the constraint programming space has focused on the development of stronger valid constraints for the subgraph isomorphism problem which in turn enables better filters for a quicker traversal of the search tree.

Régin [Reg94] devised a method to impose a generalized form of arc consistency for the alldifferent constraint on k variables. This was followed by the LV algorithm [LV02] which introduced a constraint based on the size of the neighborhoods of template vertices. The ILF algorithm [ZDS10] proposed a method of strengthening constraints on individual vertices to constraints on neighborhoods of these vertices. Solnon [Sol10], in the LAD algorithm, introduced constraints which require that an injective mapping exists from the neighborhood of any template vertex to the neighborhood of a candidate world vertex. The SND algorithm [ALS14] adds a constraint based on the number of k-length paths through a given vertex. Aspects from the LAD and SND algorithms were then combined to produce the PathLAD algorithm [KMS16]. The latest algorithm to emerge in this paradigm is the Glasgow algorithm [MP15, MPT20] which adds constraints defined for supplemental graphs as well as presents a novel method for parallelizing the tree search.

As a whole, constraint programming approaches for subgraph isomorphism bear a strong similarity to the methods presented in Section 2.2.1 in that the methods involve ultimately a tree search and constraint propagation effectively amounts to pruning branches which cannot possibly lead to solutions. What distinguishes these approaches from standard tree search

methods like VF2 is how they describe the problem as less a procedure to attain matching subgraphs via a tree search and more a declarative description of the problem as a set of variables with corresponding domains of which any solution must satisfy a set of constraints. Posed in this manner, even if the resulting algorithm ultimately is a standard tree search, these works can make use of the numerous methods from the constraint programming literature.

2.2.3 Indexing Approaches

The indexing approach to finding subgraph isomorphisms was developed alongside the creation of large databases of graphs and the desire to determine which of these graphs contain a particular template graph of interest. Put formally, given a graph database of world graphs, $D = \{\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_n\}$ and a template graph $\mathcal{G}_t$ (also commonly called a *query*), the goal is to produce the subset of graphs in D which contain $\mathcal{G}_t$ as a subgraph. Indexing approaches have also been commonly used in the setting of having one very large world graph where building an index for this large graph may speed up the subgraph matching process for a collection of small template graphs significantly [ZLY09].

Any of the algorithms mentioned thus far would be suitable for this task, but a central assumption behind these approaches is that applying a tree search approach to enumerating the subgraph isomorphisms for each graph in D would be too costly. Instead, an index for each of the graphs in the database is created which would enable quick filtering out of graphs which world graphs which cannot contain $\mathcal{G}_t$. For the remaining graphs in D, a more costly tree search algorithm (like VF2) is then applied to verify which world graphs do contain $\mathcal{G}_t$. This two-step process has led this approach to be called the *filter-and-verify* algorithm.

A very simple example of one such index for a collection of labeled graphs may simply be a count of the number of vertices with a given label. Then for any template graph, we can similarly count the number of vertices with this label and immediately rule out any world graph which do not have any vertices of this kind. Graph indexes vary in complexity from

simple easy-to-compute quantities as these to counts of more complicated graph substructures. The strength of any given index is measured by how many world graphs are ruled out by the index for a given template graph. There is a natural trade off between how strong an index is and how long an index takes to construct. Let C_q be the set of graphs in D which remain after using index to filter out graphs. If we denote the time for constructing the graph index by T_{index}, the average time for subgraph isomorphism checking as T_{iso}, we can represent the total time for finding which graphs in the database contain $\mathcal{G}_t$ as

$$T = T_{\text{index}} + |C_q| T_{\text{iso}}. \tag{2.4}$$

This approximation for computation time was first presented in [YYH04]. Proponents of indexing approaches reason that T_{iso} is the most costly operation, so a very expensive preprocessing step with a large T_{index} is justifiable if the resulting index can filter out a large number of graphs so that $|C_q|$ is relatively small.

We now present a variety of the different methods for creating the indexes which are used in graph databases. One of the first algorithms, GraphGrep [GS02], uses counts of the number of paths with specific vertex labels as an index and develops a query method by which all subgraph isomorphisms may be found. The algorithm gIndex [YYH04] proposes a very strong index given by a collection of frequent subgraphs of the database graphs with rules to ensure the frequent subgraphs are not redundant and are discriminative (meaning the presence of such a subgraph in a template would rule out many world graphs). In an expansion on gIndex, the authors of FG-index [CKN07] attempt to reduce the computation and memory requirements by instead only indexing what are called δ-tolerance closed frequent subgraphs where δ is a parameter which is used to tune the number of frequent subgraphs stored so as to fit into memory.

The authors of the Tree+Δ algorithm [ZYY07] proposed a slightly weaker index composed of frequent subtrees arguing that this can capture the diversity of subgraphs while not being as costly to compute. The Treepi algorithm [ZHY06] also indexes based on frequent trees

while also providing a novel method for finding subgraph isomorphisms in conjunction with this index based on distances between the centers of subtrees.

In a novel approach, the creators of the Closure-Tree algorithm [HS06] organize the frequent subgraphs indexed in a tree where a substructure $\mathcal{G}_1$ is a descendant of $\mathcal{G}_2$ if one is a subgraph of the other. They compute "pseudo subgraph isomorphisms" which are approximate matches to save on computation time. The GADDI indexing algorithm [ZLY09] defines an index composed of the intersection of neighborhoods of close vertices as its frequent substructures which it also uses in its own subgraph isomorphism algorithm. The Gstring algorithm [JWP06] uses as an index, a decomposition of the world graphs into basic substructures like paths, cycles, and stars.

The Lindex algorithm [YM13] organize the frequent subgraphs in a lattice structured index with graphs being adjacent if one is a subgraph of the other. In this fashion, they aim to be able to quickly find the maximal common subgraph or minimal common supergraph between two graphs. GraphGrepSX [BFG10] is an extension of GraphGrep in that it also records paths of a certain length, but it stores them in a suffix tree for easy access. Then in the subgraph search, maximal paths of a template are compared to the index to filter out graphs based on frequency of these paths. CT-index [KKM11] creates an exhaustive index of paths, trees, and cycles up to a certain size and uses an adaptation of VF2 to perform verification. Grapes [GBB13] is an algorithm which uses a path-based index with additional location information about the paths as well as a design that lends itself well to the parallelization of a subgraph search.

Recent work suggests that the computation time T in (2.4) may be misleading as the world graphs filtered out may be easier subgraph isomorphism problems leaving the hard and costly problems to still be solved in the verification step. The authors of [MPS18] argue that a well-written constraint programming approach should be able to quickly solve these easy problems at low cost and question the need for the costly indexing step.

2.3 Inexact Subgraph Matching

One problem closely related to the subgraph isomorphism problem is the inexact subgraph matching problem. This problem relaxes the edge-preserving and label-preserving conditions of exact subgraph isomorphism and instead introduces a penalty for edge and label mismatches. The goal then is to find a mapping $f : \mathcal{V}_t \to \mathcal{V}_w$ which minimizes this penalty which will be a mapping which should closely approximate a subgraph isomorphism. The exact form of the penalty varies across applications and there are a myriad of approaches many taking inspiration from the exact matching problem. An example model for the cost for a mapping f may involve a cost function imposing a penalty for failing to preserve edges, for mismatching node labels, and for mismatching edge labels. If c_E, $c_{\mathcal{L}_V}$ and $c_{\mathcal{L}_E}$ represent these three functions, respectively, a possible cost for a given mapping $f : \mathcal{V}_t \to \mathcal{V}_w$ may

$$c(f) = \sum_{e \in \mathcal{E}_t, f(e) \notin \mathcal{E}_w} c_E(e) + \sum_{v \in \mathcal{V}_t} c_{\mathcal{L}_V}(\mathcal{L}_V(v), \mathcal{L}_V(f(v))) + \sum_{e \in \mathcal{E}_t} c_{\mathcal{L}_E}(\mathcal{L}_E(e), \mathcal{L}_E(f(e))). \tag{2.5}$$

Many techniques used for finding subgraph isomorphisms can be applied to the inexact subgraph matching problem. The concepts of tree search and filtering can be adapted instead to search for matchings with minimal cost and prune branches in the search tree which can only hold solutions above a certain threshold. Two different approaches [TMY20, KX19] take this branch-and-bound style of approach. These algorithms produce an approximation for the minimal cost of any mapping extending a partial matching by solving a linear assignment problem derived from the cost function. With this bound, they can filter out candidates that lead to mappings of high cost. The algorithm presented in [JHW19] attempts to minimize this cost function using the A^* algorithm.

In the event that our template and world each have n vertices and we only consider missing edges and impose a constant cost for each edge present in the template and not in the world, we can frame this problem in a simpler fashion. In this context, mappings can be represented as permutations of the template vertices. Let A and B be the adjacency

matrices of the template and world graph respectively. Then the problem is to minimize

$$c(P) = \|AP - PB\|_F^2 \tag{2.6}$$

over all permutation matrices P, where $\|\cdot\|_F$ is the Frobenius norm of a matrix. This problem has been studied in great depth (see [CFS04a] for a survey), and typical approaches [VCL15] involve minimizing c over the convex hull of permutation matrices, the set of doubly stochastic matrices. This approach can be extended to graphs of differing sizes by padding the template graph with isolated vertices so that the two graphs are of the same size. This is the approach taken in [SPP19] where the authors analyze different techniques for padding and then apply [VCL15] to find a solution.

A different model for the cost of the of matching a template graph to a subgraph of a world graph is the graph edit distance, which is a measure of how many operations would be required to transform one graph into another. Certain graph operations are allowed (inserting or deleting vertices or edges, and changing labels), and each operation has an associated cost with it. The graph edit distance is then given by cost of the sequence of operations which transform the one graph into another with minimal cost. Computing this distance is known to be NP-hard [GJ79].

Several graph indexing approaches have been developed where the indexes are used to immediately filter out graphs which have a prohibitively high graph edit distance. Similar to indexing approaches for the exact subgraph isomorphism problem, these indexes consider frequent substructures, in this context called q-grams in analogy to q-grams used for approximate string matching [Ukk92]. Frequencies of structures which include subtrees of a distance k from each node [WWY10], paths of certain length [ZXL12], and star graphs [WDT12] have been used to derive lower bounds on graph edit distance to be used as a filter.

CHAPTER 3

Structural Equivalence in Subgraph Matching[1]

3.1 Introduction

In this chapter, we concern ourselves with graph symmetry and its impact on the problem of finding subgraph isomorphisms. As an example of symmetry, observe the template graph in Figure 3.1 which is from a system of biochemical reactions [GFM14] and note that each pair of colored vertices is interchangeable in any subgraph isomorphism as they have the exact same neighbors. As there are 11 such pairs in the graph, for any found isomorphism, we can generate $2^{11} - 1 = 2047$ more solutions simply by interchanging vertices. By avoiding redundant solutions in a subgraph search, we can significantly reduce the search time (potentially by a factor of 2048 or more). This simple form of symmetry where vertices have the exact same set of neighbors is known as **structural equivalence**.

Broader notions of equivalence can be used to further accelerate search. In Figure 3.2, the yellow and blue vertices are each individually structurally equivalent. However, if we proceed by matching A to 1, then we can complete an isomorphism by matching B and C to any of 2, 3, 4, or 5. The presence of additional edges incident to 4 and 5 hides that 4 and 5 may be swapped out for 2 or 3. By identifying when these additional edges may be ignored, we can again dramatically reduce the amount of work. This second notion of equivalence we will refer to as **candidate equivalence**. In the body of this chapter, we will formally define these terms and demonstrate how they can be applied in a tree search

[1]**This chapter is adapted from [YGN23]**

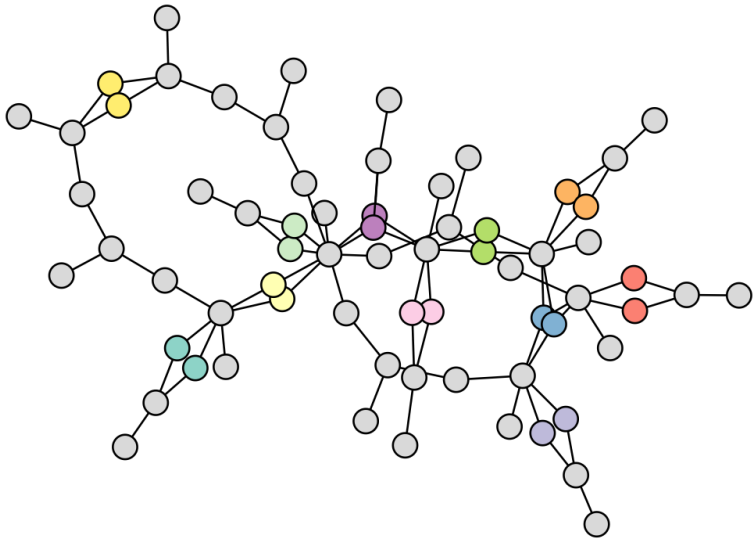

Figure 3.1: Graph representing a system of biochemical reactions from [GFM14]. Non-gray vertices of the same color are structurally equivalent.

algorithm. These two notions of equivalence can be broadly classified into two categories: **static equivalence**, which describes equivalence apparent from the problem description, and **dynamic equivalence**, which describes equivalence uncovered in the search process. Structural equivalence falls into the former category while candidate equivalence belongs to the latter. We note that these forms of equivalence are dependent on the structure of the graphs and cannot be used should we interchange one template graph for another.

Our work identifies and characterizes the structure of redundancies due to symmetry in the matching problem. We exploit these symmetries to produce a compressed version of the solution space which saves space as well as aids in understanding the problem's solutions.

In the literature, there has been significant work done to exploit symmetry to compress graphs and count isomorphisms. TurboIso [HLL13] exploits basic symmetry in the template graph and optimizes the matching order based on a selection of candidate regions and exploration within those regions. BoostIso [RW15] exploits symmetry in the world graph and presents a method by which other tree-search-based approaches are accelerated by using their methodology. The ISMA algorithm [DMF13] exploits basic bilateral and rotational

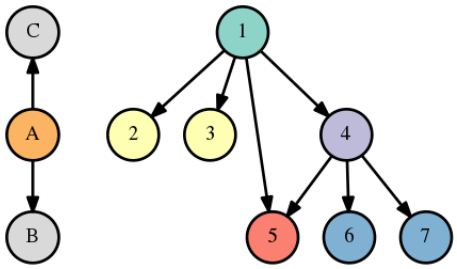

Figure 3.2: Example subgraph isomorphism problem with template on the left and world on the right. Vertices of the same color are structurally equivalent.

symmetry of the template to boost subgraph search and this work was extended into the ISMAGS algorithm [HDM14] to incorporate general automorphic symmetries of the template graph.

3.1.1 Chapter Outline

In this chapter, we demonstrate how tree search-oriented approaches can be accelerated by exploiting both static forms of equivalence, apparent at the start of search, as well as dynamic forms of equivalence, which are uncovered as the search proceeds. In Section 3.2, we formally define structural equivalence and how to incorporate it into a subgraph search. In Section 3.3, we introduce candidate equivalence to demonstrate how to expose not immediately apparent equivalences during a subgraph search. In Section 3.4, we introduce node cover equivalence, an alternate form of equivalence which is easy to calculate, and unify all the notions of equivalence into a hierarchy. In Section 3.5, we adapt the state-of-the-art solver Glasgow [MPT20] to incorporate equivalences and apply it to a set of benchmarks to assess the performance of each of the equivalence levels[2]. In Section 3.6, we demonstrate how to succinctly represent and visualize large classes of solution by incorporating equivalence. In Section 3.7, we extend our algorithm to be able to handle multiplex multigraphs and show

[2]Our implementation of our algorithms can be found at the following repository: `https://github.com/domyang/glasgow-subgraph-solver`.

our algorithm's success in fully mapping out the solution space on a variety of these more structured networks.

This chapter takes inspiration for the general subgraph tree search structure and shares many of the same test cases on multichannel networks as [MTC21]. In a previous work [NYG19], we introduced a simpler notion of candidate equivalence and candidate structure, and tested it on a small selection of multichannel networks. From these prior works, we observed the high combinatorial complexity of the solution spaces necessitating an approach which can exploit symmetry to compress the solution space and accelerate search. In this work, we expand on both papers by introducing several new notions of equivalence, providing a rigorous foundation for their efficacy in subgraph search, establishing a compact representation of the solution space, and empirically assessing these methods on a broad collection of both real and synthetic data sets.

3.2 Structural Equivalence

Structural equivalence is a simple property of networks which, if present, can be exploited to greatly speed up subgraph search. Intuitively, two vertices are structurally equivalent to each other if they can be "swapped" without changing the graph structure. This type of equivalence often occurs in leaves that are both adjacent to the same vertex.

Definition 2. *In a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, we say that two vertices v, w are* ***structurally equivalent*** *(denoted $v \sim_s w$) if:*

1. *For $u \in \mathcal{V}, u \neq v, w$,*
   - *(a)* $(u, v) \in \mathcal{E} \Leftrightarrow (u, w) \in \mathcal{E}$
   - *(b)* $(v, u) \in \mathcal{E} \Leftrightarrow (w, u) \in \mathcal{E}$
2. $(v, w) \in \mathcal{E} \Leftrightarrow (w, v) \in \mathcal{E}$

This definition implies that the neighbors of structurally equivalent vertices (not including the vertices themselves) must coincide. The following proposition verifies that this is an equivalence relation. If the graphs are labeled, then in addition we require that $\mathcal{L}_V(v) = \mathcal{L}_V(w)$ and that the edge labels of each pair of edges above agree.

Proposition 3. *$\sim_s$ is an equivalence relation.*

Proof. Reflexivity and symmetry are both obvious, so we just need to check transitivity. This is immediate by seeing for $u \sim_s v$ and $v \sim_s w$, it is clear that any z that is an in-neighbor or out-neighbor for u, is one for v and therefore is one for w. Similarly, if $(u, w) \in \mathcal{E}$, then $(v, w) \in \mathcal{E}$ implying $(w, v) \in \mathcal{E}$ and therefore $(w, u) \in \mathcal{E}$. Hence this relation is transitive. □

Using this relation, we can partition the vertices of any graph into structural equivalence classes, and interchange members of each class without changing the essential structure of the graph. Checking for equivalence between two vertices simply amounts to comparing neighbors in an $O(|V|)$ operation in the worst case, but is generally faster for sparse graphs. Computing the classes themselves can be found by pairwise comparison of vertices resulting in $O(|V|^3)$ operations in the worst case. Algorithm 2 demonstrates how one could implement a breadth first search algorithm to take advantage of the sparsity of a graph to accelerate the computation. Since for each vertex v visited, it takes $O(deg(v)^2)$ to partition the neighbors of v into equivalence classes, and so in the worst case the algorithm takes $O(\sum_v deg(v)^2) \approx O(|V|\overline{deg(v)^2})$ where $\overline{deg(v)^2}$ denotes the average over $deg(v)^2$ for all v. For sparse graphs, $deg(v) \ll |V|$, so this will be significantly faster than a naive pairwise check.

3.2.1 Interchangeability and Isomorphism Counting

We now show that given any subgraph isomorphism, we can interchange any two vertices in the template graph and still retain a subgraph isomorphism. Before we do this, we formally define what we mean by interchangeability.

Algorithm 2 Routine for computing equivalence classes

```
1: function FindEqClasses(G = (V, E))
2:     Let Q be a queue
3:     Pick first vertex v to put in Q
4:     Let EQ = {}
5:     Let visited = {}
6:     while Q not empty do
7:         Dequeue v from Q
8:         Add v to visited
9:         Partition N(v) into equivalence classes, to EQ
10:        Add representatives from neighbor classes to Q
11:    Check if first vertex v is in any class and add if so
12:    Else add it to its own class
13:    return EQ
```

Definition 4. *Two template graph vertices $v, w \in \mathcal{V}_t$ are **interchangeable** if for all subgraph isomorphisms $f : \mathcal{V}_t \to \mathcal{V}_w$, the mapping g given by interchanging v and w:*

$$g(u) = \begin{cases} f(w) & u = v \\ f(v) & u = w \\ f(u) & otherwise \end{cases}$$

is also a subgraph isomorphism.

*Two world graph vertices $v', w' \in \mathcal{V}_w$ are **interchangeable** if for all subgraph isomorphisms f, if both v', w' are in the image of f with preimages v, w, the mapping g:*

$$g(u) = \begin{cases} w' & u = v \\ v' & u = w \\ f(u) & otherwise \end{cases}$$

is an isomorphism. If only one, say v', is in the image, then h given by

$$h(u) = \begin{cases} w' & u = v \\ f(u) & otherwise \end{cases}$$

is also an isomorphism.

To handle more complicated notions of interchangeability, we will qualify this definition later by restricting the interchangeability only to certain subsets of isomorphisms. The proposition affirming template vertex interchangeability under template structural equivalence follows:

Proposition 5. *Given graphs $\mathcal{G}_t = (\mathcal{V}_t, \mathcal{E}_t)$, $\mathcal{G}_w = (\mathcal{V}_w, \mathcal{E}_w)$, if $v, w \in \mathcal{V}_t$ are structurally equivalent, then they are interchangeable in any subgraph isomorphism.*

Proof. Obviously g is still injective, so we need only check that it is edge-preserving. Suppose $(x, y) \in \mathcal{E}_t$. We consider multiple cases. If neither x, y are v or w, then $(g(x), g(y)) = (f(x), f(y)) \in \mathcal{E}_w$ as f is a subgraph isomorphism. If $x = v$ and $y = w$, then $(g(x), g(y)) = (f(w), f(v))$. Now as $v \sim_s w$ and $(v, w) \in \mathcal{E}_t$, so is (w, v) and using that f is a subgraph isomorphism, $(f(w), f(v)) \in \mathcal{E}_w$. Now suppose one of x, y is one of v, w. Without loss of generality let $x = v$. Then $(g(x), g(y)) = (f(w), f(y))$. As $v \sim_s w$, y is also an out-neighbor of w, and so $(g(x), g(y)) = (f(w), f(y)) \in \mathcal{E}_w$. This verifies that g is a subgraph isomorphism. □

Hence, interchanging the images of two template vertices preserves subgraph isomorphism. As transpositions generate the full set of permutations, we have the following result:

Proposition 6. *If we can partition $\mathcal{V}_t = C_1, \ldots, C_n$ into structural equivalence classes, and there exists at least one subgraph isomorphism, then there are at least*

$$\prod_{i=1}^{n} |C_i|!$$

subgraph isomorphisms.

We can also apply this structural equivalence to the world graph to demonstrate a similar kind of interchangeability.

Proposition 7. *If $v', w' \in \mathcal{V}_w$ are structurally equivalent, then in any subgraph isomorphism f, they are interchangeable.*

Proof. The proofs of this proposition for the two types of interchanges in Definition 4 are very similar, so we just prove the first one. g is obviously injective, so we just check that it is edge-preserving. Given $(x, y) \in \mathcal{E}_t$, we consider three cases. In the first case, neither of x and y are v or w, $\{x, y\} \cap \{v, w\} = \varnothing$. Then $(g(x), g(y)) = (f(x), f(y)) \in \mathcal{E}_w$. In the second case, one of x and y equals one of v or w; either $x \in \{v, w\}$ or $y \in \{v, w\}$. Without loss of generality, we take $x = v$, then $(g(x), g(y)) = (w', f(y))$. As $w' \sim_s v'$ and $(f(x), f(y)) = (v', f(y)) \in \mathcal{E}_w$, we have $(w', f(y)) \in \mathcal{E}_w$. For the final case, $\{x, y\} = \{v, w\}$. Without loss of generality we take $x = v, y = w$. Then $(g(x), g(y)) = (w', v') = (f(y), f(x))$. As $(f(v), f(w)) = (v', w') \in \mathcal{E}_w$, the reverse edge $(w', v') \in \mathcal{E}_w$ since $v' \sim_s w'$. This completes the proof. □

If we apply both template and world structural equivalence to our problem, it is natural to ask how many solutions we can now generate from a single solution. This is given by the following proposition:

Proposition 8. *Let $f : \mathcal{V}_t \to \mathcal{V}_w$ be a subgraph isomorphism. Suppose we partition the template graph $\mathcal{V}_t = \bigcup_{i=1}^{n} C_i$ and the world graph $\mathcal{V}_w = \bigcup_{j=1}^{m} D_j$ into structural equivalence classes. Let $C_{i,j} = C_i \cap f^{-1}(D_j)$ represent the set of template vertices in C_i that map to world vertices in the equivalence class D_j. Then there are*

$$\prod_{i=1}^{n} |C_i|! \prod_{j=1}^{m} \prod_{i=1}^{n} \binom{|D_j| - \sum_{k=1}^{i-1} |C_{k,j}|}{|C_{i,j}|}$$

isomorphisms generated by interchanging equivalent template vertices or world vertices using Propositions 5 and 7.

Proof. As before, the first factor comes from all permutations on the equivalence classes. Once we fix a permutation, from each world equivalence class D_j, we need to choose $|C_{i,j}|$

elements to be the values for the elements of $C_{i,j}$. For $C_{1,j}$, we have $\binom{|D_j|}{|C_{1,j}|}$ elements, for $C_{2,j}$, we have $\binom{|D_j|-|C_{1,j}|}{|C_{2,j}|}$ as we have already used $|C_{1,j}|$ elements, and so on. Taking the product over j gives the second factor. □

3.2.2 Application to Tree Search

We now demonstrate how to adapt any tree-search algorithm to incorporate equivalence. A tree-search algorithm proceeds by constructing a partial matching of template vertices to world vertices, at each step extending the matching by assigning the next template vertex to one of its candidate world vertices. If at any point, the match cannot be extended (due to a contradiction or finding a complete matching), the last assigned template vertex is reassigned to the next candidate vertex. Each possible assignment of template vertex to world vertex corresponds to a node in the tree, and a path from the root of the tree to a leaf corresponds to a full mapping of vertices.

Algorithm 3 demonstrates how to incorporate equivalence into the tree search given by Algorithm 1. Template equivalence can significantly accelerate the tree search. From Proposition 5 we can swap the assignments of equivalent template vertices to find another isomorphism. If we have a partial match, template vertices $u_1 \sim_s u_2$, and we have just considered candidate w for u_1, we can ignore branches where u_2 is mapped to w since we can generate those isomorphisms by taking one where u_1 is mapped to w and swapping. Lines 16 and 17 demonstrate how we can incorporate this idea into a tree search (without these, we would have a standard tree search).

To incorporate world equivalence into the search, we modify the search so that we only assign any template vertex to one representative of an equivalence class in the search. This can be done by modifying GenerateWorldVertices to pick only one representative vertex of each equivalence class out of the candidates for the current template vertex.

Note that after performing the tree search, the solutions found will represent classes of

Algorithm 3 Tree Search Adapted For Equivalence

```
1: function TREE SEARCH(Template G_t, World G_w, partial_match, cands)
2:     if MatchComplete(partial_match) then
3:         ReportMatch(partial_match)
4:         return
5:     ApplyFilters(G_t, G_w, partial_match, cands)
6:     Let u = GetNextTemplateVertex(G_t)
7:     Let ws = GenerateWorldVertices(G_w, u, cands)
8:     if Using World Equivalence then
9:         RecomputeEquivalence(G_t, G_w, partial_match, cands)
10:    Let ws = GenerateWorldVertices(G_w, u, cands, eq)
11:    for v in ws do
12:        partial_match.match(u, v)
13:        TreeSearch(G_t, G_w, partial_match, cands)
14:        partial_match.unmatch(u, v)
15:        if Using Template Equivalence then
16:            for unmatched u' ~ u do
17:                Set v as not a candidate for u'
18:    if Using World Equivalence then
19:        RestoreEquivalence(eq)
20:    return
```

solutions that can be generated by swapping. Some bookkeeping is needed to determine what assignments can be swapped to count the number of distinct solutions. We call the solutions that are actually found (and are not produced by interchanging vertices) **representative solutions**. The set of solutions that can be generated by interchanging equivalent vertices for a given representative solution is a **solution class**. The ability to represent large solution classes with a sparse set of solutions is what allows us to compactly describe massive solution spaces.

3.3 Candidate Equivalence

The equivalence discussed in the prior section is a static form of equivalence, only taking into account information provided at the start of the subgraph search. However, as a subgraph search proceeds, we may be able to discard additional vertices and edges based on information derived from the assignments already made. For example, in Figure 3.2, after assigning A to 1, we may discard vertices 6 and 7 and edge (4, 5), as it is impossible for them be included in a match if A and 1 are matched. After these vertices are discarded, we discover that with respect to the matches already made, 2, 3, 4, and 5 are effectively interchangeable. In order to make use of this dynamic form of equivalence, we need to introduce an auxiliary structure that takes into account our knowledge of the candidates of each vertex u, $C(u)$.

Definition 9. *Given template graph $\mathcal{G}_t = (\mathcal{V}_t, \mathcal{E}_t)$, world graph $\mathcal{G}_w = (\mathcal{V}_w, \mathcal{E}_w)$, and candidate sets $C(u) \subset \mathcal{V}_t$ for each $u \in \mathcal{V}_t$, the **candidate structure** is the directed graph $\mathcal{G}_C = (\mathcal{V}_C, \mathcal{E}_C)$ where the vertices $\mathcal{V}_C = \{(u, c) : u \in \mathcal{V}_t, c \in \mathcal{V}_w\}$ are template vertex-candidate pairs and $((u_1, c_1), (u_2, c_2)) \in \mathcal{E}_C$ if and only if $(u_1, u_2) \in \mathcal{E}_t$ and $(c_1, c_2) \in \mathcal{E}_w$.*

The candidate structure represents both the knowledge of candidates for each template vertex and how template adjacency interplays with world adjacency. It removes extraneous information to expose equivalences not apparent when looking at the original graphs. We note that this data structure is similar to the compact path index (CPI) introduced in

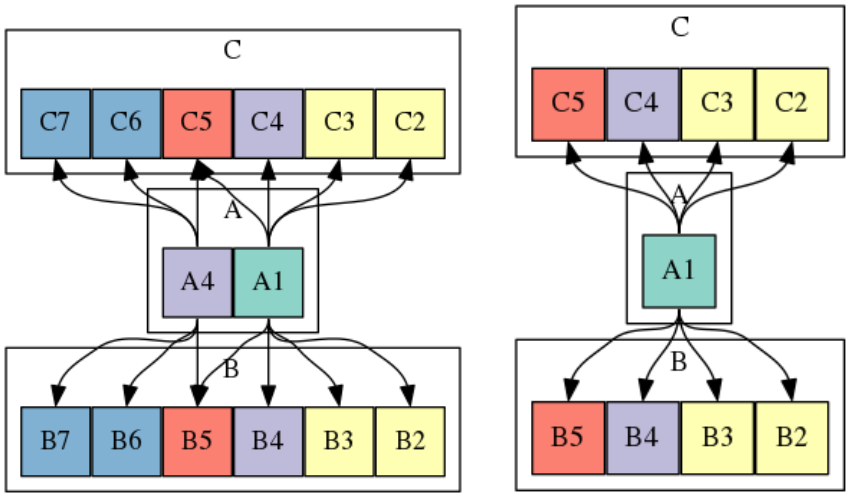

Figure 3.3: Candidate structure for the graphs in Figure 3.2 before and after assigning template vertex A to world vertex 1.

[BCL16]. However, the CPI is only defined for a given rooted spanning tree of the template graph whereas our candidate structure takes into consideration all edges of the template graph. For our toy example in Figure 3.2, if we assume that the candidate sets are reduced to the minimal candidate sets so that $C(A) = \{1, 4\}$ and $C(B) = \{2, 3, 4, 5, 6, 7\}$. Then the candidate structure for these graphs is as shown on the left in Figure 3.3. At this point, there is no apparent equivalence to exploit from the candidate structure. However once we decide to map vertex A to 1, the candidate structure reduces to the right graph in Figure 3.3. It is visually clear that vertices 2, 3, 4, and 5 are structurally equivalent as candidates of B and C. Similarly, if we assigned A to 4, vertices 5, 6, and 7 will be structurally equivalent in the candidate structure. We want to determine under which circumstances this will ensure interchangeability. We introduce the following definition:

Definition 10. *Given a candidate structure* $\mathcal{G}_C = (\mathcal{V}_C, \mathcal{E}_C)$, $c_1, c_2 \in \mathcal{V}_w$, *we say that* c_1 *is* ***candidate equivalent*** *to* c_2 *with respect to* $u \in \mathcal{V}_t$, *denoted* $c_1 \sim_{c,u} c_2$, *if and only if* $c_1, c_2 \notin C(u)$ *or* $c_1, c_2 \in C(u)$ *and* $(c_1, u) \sim_s (c_2, u)$.

It is easy to show that if the candidate sets are *complete* (for each template vertex u, if there is a matching which maps u to world vertex v, then $v \in C(u)$), then if $c_1 \sim_s c_2$, then $c_1 \sim_{c,u} c_2$ for all template vertices u.

The exact criteria for interchangeability is a little more complicated. For example, in Figure 3.2, 4 appears as a candidate for both A and for B and C, so that we cannot simply swap 4 with vertices that are candidate equivalent to 4 with respect to B. To address a more complex notion of interchangeability, we introduce some terms. We say that a subgraph isomorphism f is derived from a candidate structure $\mathcal{G}_C$ if for any $v \in \mathcal{V}_t, (v, f(v)) \in \mathcal{V}_C$ (i.e., $f(v)$ is a candidate of v). We say that world vertices w_1, w_2 are **$\mathcal{G}_C$-interchangeable** if for all isomorphisms derived from the candidate structure $\mathcal{G}_C$, w_1 and w_2 can be interchanged and preserve isomorphism.

A simple criterion for interchangeability is provided in the following proposition:

Proposition 11. *Suppose that given a specific candidate structure $\mathcal{G}_C = (\mathcal{V}_C, \mathcal{E}_C)$, we have that $c_1, c_2 \in C(u)$ and $c_1 \sim_{c,u} c_2$ for some template vertex u. Suppose that c_1 and c_2 are not candidates for any other vertex. Then c_1 and c_2 are $\mathcal{G}_C$-interchangeable.*

Proof. This is clearly injective, and for any edge (v, w) which doesn't include u, g agrees with f, so it preserves those edges. If we have $(u, v) \in \mathcal{E}_t$, we have $(f(u), f(v)) = (c_1, f(v)) \in \mathcal{E}_w$. Since $c_1 \sim_{c,u} c_2$, and we have $((u, c_1), (v, f(v)) \in \mathcal{E}_C$, we must have $((u, c_2), (v, f(v)) \in \mathcal{E}_C$ which implies $(c_2, f(v)) \in \mathcal{E}_w$. □

This proposition suggests a simple method for exploiting candidate equivalence. In our tree search, when we generate candidate vertices for a given vertex u, we find representatives, for each candidate equivalence class, that do not appear as candidates for other vertices. If a class has a vertex appearing in other candidate sets, then we cannot exploit equivalence and must check each member of the class. Furthermore, as we continue to make matches and eliminate candidates, more world vertices will become equivalent, so it is advantageous

to recompute equivalence before every match as is done in line 9 of Algorithm 3. Upon unmatching, we need to restore the prior equivalence as is done in line 19.

If we have that $f(v) = c_1$ and $f(w) = c_2$, and we want to swap c_1 for c_2, we need a stronger condition; namely, we need that they are equivalent with respect to both v and w. In the process of a tree search, we do not know exactly what each vertex will be mapped to so instead we consider an even stronger condition:

Definition 12. *Given a candidate structure $\mathcal{G}_C = (\mathcal{V}_C, \mathcal{E}_C)$, we say that $c_1 \in \mathcal{E}_w$ is **fully candidate equivalent** to $c_2 \in \mathcal{V}_w$, denoted $c_1 \sim_c c_2$ if for all $u \in \mathcal{V}_t$, $c_1 \sim_{c,u} c_2$.*

Note that if $c_1 \sim_{c,u} c_2$ for some u, and c_1, c_2 are not candidates for any other vertices, then $c_1 \sim_c c_2$. This condition enables us to interchange world vertices and still maintain the subgraph isomorphism conditions. This is established by the following proposition:

Proposition 13. *Suppose that given a specific candidate structure $\mathcal{G}_C = (\mathcal{V}_C, \mathcal{E}_C)$, we have that $c_1, c_2 \in \mathcal{V}_w$ and $c_1 \sim_c c_2$. Then, c_1 and c_2 are $\mathcal{G}_C$-interchangeable.*

Proof. If $(x, y) \in \mathcal{E}_t$, and neither is u_1 or u_2, then g agrees with f and preserves the edge. If one is u_1 or u_2, without loss of generality take $x = u_1$, then $(g(x), g(y)) = (c_2, f(y))$. As f is a subgraph isomorphism $(c_1, f(y)) \in \mathcal{E}_w$ and since $c_1 \sim_c c_2$, $(c_2, f(y)) \in \mathcal{E}_w$ as well. If $x = u_1, y = u_2$, then $(g(x), g(y)) = (c_2, c_1) = (f(y), f(x))$ and this edge is in $\mathcal{E}_w$ since $c_1 \sim_c c_2$ and $(c_1, c_2) \in \mathcal{E}_w$. □

3.4 Node Cover Equivalence

An alternate notion of equivalence, introduced in [MTC21], involves the use of a node cover. A **node cover** is a subset of vertices whose removal, along with incident edges, results in a completely disconnected graph. The approach in [MTC21] is to build up a partial match of all the vertices in the node cover followed by assigning all the vertices outside the node cover. After reducing the candidate sets of all the vertices outside the cover to those that

have enough connections to the vertices in the cover, what remains is to ensure that they are all different.

We formalize this with some definitions. A **partial match** is a subgraph isomorphism from a subgraph of the template graph to the world graph. We list out the mapping as as a list of ordered pairs $M = \{(v_1, w_1), \ldots, (v_n, w_n)\}$. A template vertex - candidate pair (v, c) is **joinable** to a partial match M if for each $(v_i, w_i) \in M$, if $(v_i, v) \in \mathcal{E}_t$, then $(w_i, c) \in \mathcal{E}_w$ and if $(v, v_i) \in \mathcal{E}_t$, then $(c, w_i) \in \mathcal{E}_w$. If two world vertices w_1, w_2 are interchangeable in any subgraph isomorphism extending a partial match M, we say that w_1 and w_2 are M**-interchangeable**.

Since the problem is significantly simpler, it is easier to obtain a form of equivalence on the vertices.

Definition 14. *Let M be a partial match M on a node cover N of $\mathcal{V}_t$ and suppose that for all $u \in \mathcal{V}_t \setminus N$, the candidate set $C(u)$ is comprised entirely of all world vertices joinable to M. Two world vertices w_1, w_2 are* ***node cover equivalent*** *with respect to M, denoted $w_1 \sim_{N,M} w_2$, if for all $u \in \mathcal{V}_t \setminus N$, $w_1 \in C(u)$ if and only if $w_2 \in C(u)$.*

For example, consider the template and world in Figure 3.4. Once vertices B and D in the node cover are mapped to 2 and 5, the remaining vertices have candidates that have the associated color in the world graph. We then simply group each of these candidates together into equivalence classes. Note that the edges depicted in red are what prevent structural equivalence, and the node cover approach effectively ignores these edges to expose the equivalence of these vertices.

Proposition 15. *Suppose we have a node cover of the template graph N, and a partial matching M on N and two world vertices w_1, w_2 not already matched satisfy $w_1 \sim_{N,M} w_2$. Then w_1 and w_2 are M-interchangeable.*

Proof. Let f be such an isomorphism which maps u_1 to w_1 and u_2 to w_2 and g interchanges w_1 and w_2. Consider $(x, y) \in \mathcal{E}_t$. If neither are u_1, u_2, then g agrees with f and so the edge is

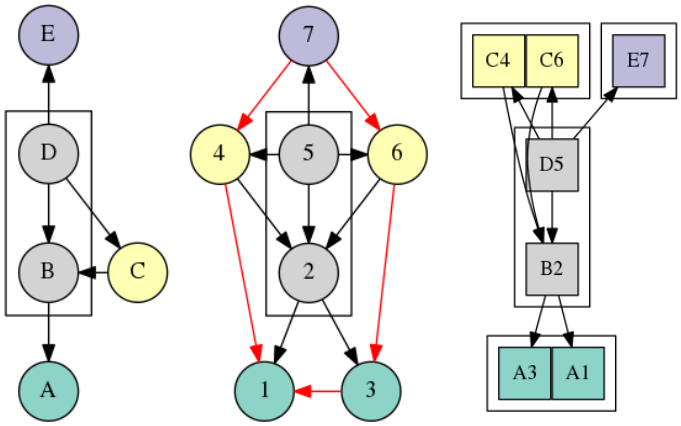

Figure 3.4: In order from left to right: template, world, and possible candidate structure. The boxed vertices comprise a node cover of the template and the image of the node cover in the world. Vertices of the same color in the world are node cover equivalent. The red edges are extraneous edges which once removed, expose equivalence.

preserved. If one of them is u_1 or u_2, say $x = u_1$, then it must be that y is in N as N is a node cover (u_1 is disconnected from any element outside the node cover). Since f is a subgraph isomorphism, (u_1, w_1) must be joinable to M and $(w_1, f(y)) \in \mathcal{E}_t$ and so $w_1 \in C(u_1)$. It must be that $w_2 \in C(u_1)$ and so (u_1, w_2) is also joinable to M. Hence $(w_2, f(y)) = (g(x), g(y)) \in \mathcal{E}_t$. The last case $x = u_1$ and $y = u_2$ is impossible since x and y are outside the node cover and therefore disconnected. □

Node cover equivalence is easy to check and captures a significant portion of the equivalence posed by other methods. This is often due to interchangeable vertices being composed of sibling leaves which are generally outside of a node cover.

We note that the methods discussed in the chapter (with the exception of basic structural equivalence for template and world as in Proposition 8) cannot easily incorporate both template and world equivalence. The combination of allowing template and world vertex interchanges and having dynamic world equivalence classes significantly complicates the counting process. One approach which can facilitate the use of both forms of equivalence

involves a tree search where entire template equivalence classes are assigned at once instead of individual template vertices. We do not consider this approach here.

3.4.1 Equivalence Hierarchy

There is a relation between node cover equivalence and full candidate equivalence, given in the following proposition:

Proposition 16. *Suppose that N is a node cover of $\mathcal{V}_t$, M is a partial match on N, candidate sets are reduced to joinable vertices, and $w_1, w_2 \in \mathcal{V}_t \setminus N$. Then $w_1 \sim_{N,M} w_2 \Leftrightarrow w_1 \sim_c w_2$.*

Proof. Fix a template vertex u. If $u \in N$, then these vertices are already assigned to world vertices, neither of which will be w_1 or w_2 and so $w_1, w_2 \notin C(u)$. Therefore $w_1 \sim_{c,u} w_2$. If $u \notin N$, and we have $((u, w_1), (v, x)) \in \mathcal{E}_C$, then $(u, v) \in \mathcal{E}_t$ and $(w_1, x) \in \mathcal{E}_w$. v must be inside the node cover as those can be the only connections to u and therefore v must already be assigned to x. As $w_1 \sim_{N,M} w_2$, we must have $w_2 \in C(u)$ as well, and so must be joinable to the matching. This implies that $(w_2, x) \in \mathcal{E}_w$. Hence we must have $((u, w_2), (v, x)) \in \mathcal{E}_C$. Since this edge was chosen arbitrarily, we must have $w_1 \sim_{c,u} w_2$. Since this holds for all u, we must have $w_1 \sim_c w_2$.

On the other hand, if $w_1 \sim_c w_2$, for any template vertex t, if (t, w_1) is joinable to M, then (t, w_2) is also joinable to M. Hence, the template vertices for which w_1 and w_2 are candidates coincide, so that $w_1 \sim_{N,M} w_2$. □

Thus, until we have assigned a node cover, we can use candidate equivalence to prevent redundant branching; once we have matched all vertices in the node cover, we can check for node cover equivalence—a simpler condition.

The agreement of node cover equivalence and fully candidate equivalence is apparent in the candidate structure presented on the right of Figure 3.4. From the candidate structure, the yellow vertices and the green vertices are fully candidate equivalent as they have the

same neighbors, and that they are node cover equivalent as they only appear as candidates for the corresponding yellow and green vertices in the template.

Based on these propositions, we can organize the various notions of equivalence into a hierarchy. Structural equivalence of world vertices has the strictest requirements and implies all other forms of equivalence. Proposition 16 asserts that under mild conditions, full candidate equivalence and node cover equivalence are one and the same. We can also include candidate equivalence with respect to a template vertex as a weaker condition implied by full candidate equivalence that does not guarantee interchangeability. The following proposition summarizes these findings:

Proposition 17. *Suppose the assumptions of Proposition 16 hold. Given template vertex t, world vertices w_1, w_2, we have $w_1 \sim_s w_2 \Rightarrow w_1 \sim_{N,M} w_2 \Leftrightarrow w_1 \sim_c w_2 \Rightarrow w_1 \sim_{c,t} w_2$. Under the first three equivalences, w_1 and w_2 are interchangeable.*

From this proposition, we observe that of the notions of symmetry considered, full candidate equivalence and node cover equivalence provide the most compact solution space as they require the weakest conditions while still guaranteeing interchangeability. However, the cost in determining full candidate equivalence may be prove excessive compared to simpler types of equivalence. We address these trade-offs on real and simulated data in Section 3.5.

3.5 Experiments

To demonstrate the utility of equivalence for the SMP, we adapt a state-of-the-art tree search subgraph isomorphism solver, Glasgow [MPT20], using the modifications described in Algorithm 3. We consider seven levels of equivalence: no equivalence (NE) (default), template structural equivalence (TE), world structural equivalence (WE), template and world structural equivalence (TEWE), candidate equivalence as in Proposition 11 (CE), full candidate equivalence as in Proposition 13 (FE), and node cover equivalence (NC). Each equivalence mode is integrated into the Glasgow solver separately.

Dataset	# Instances	# Vertices		# Edges		Density	
		Min	Max	Min	Max	Min	Max
SF	100	180	900	478	5978	0.006	0.165
LV	6105	10	6671	10	209000	0.001	1.000
SI	1170	40	777	41	12410	0.005	0.209
images-cv	6278	15	151	20	215	0.019	0.190
meshes-cv	3018	40	199	114	539	0.022	0.146
images-pr	24	4	170	4	241	0.017	0.667
biochemical	9180	9	386	8	886	0.012	0.423
phase	200	30	30	128	387	0.294	0.890
www	3850	5	15	5	45	0.071	0.750

Table 3.1: Template Graph Statistics from Benchmark Datasets used in Equivalence Experiments

For each test class involving template equivalence (TE, TEWE), we compute the template structural equivalence classes, and for each involving world equivalence (WE, TEWE, CE, FE, NC), we compute world structural equivalence classes at the start using Algorithm 2. Then we make the modifications for template and world equivalence as in Algorithm 3. For algorithms requiring recomputation of the equivalence classes at each node of the tree search, for speed purposes, we only recompute equivalence for vertices that appear as candidates for the current template vertex under consideration. We check equivalence between each pair of candidates using the definitions directly.

We consider graphs from the benchmark suite in [Sol19]. Basic graph statistics for the templates from these datasets are listed in Table 3.1 and the corresponding statistics for the worlds are listed in Table 3.2. *SF* is composed of 100 instances that are randomly generated using a power law and are designed to be scale-free networks. *LV* is a diverse collection of randomly generated graphs satisfying various properties (connected, biconnected, triconnected, bipartite, planar, etc.). *SI* is a collection of randomly generated instances falling into four categories: bounded valence, modified bounded valence, 4D meshes, and Erdős–Rényi

Dataset	# Instances	# Vertices		# Edges		Density	
		Min	Max	Min	Max	Min	Max
SF	100	200	1000	592	7148	0.006	0.159
LV	6105	10	6671	10	209000	0.001	1.000
SI	1170	200	1296	299	34210	0.004	0.191
images-cv	6278	1072	5972	1540	8891	4.89e-4	0.003
meshes-cv	3018	201	5873	252	15292	4.40e-4	0.022
images-pr	24	4838	4838	7067	7067	0.001	0.001
biochemical	9180	9	386	8	886	0.012	0.423
phase	200	150	150	4132	8740	0.370	0.782
www	3850	325729	325729	1497135	1497135	1.41e-5	1.41e-5

Table 3.2: World Graph Statistics from Benchmark Datasets used in Equivalence Experiments

graphs. The *images-cv*, *meshes-cv*, and *images-pr* data [DSH11, SDD15] sets are real instances representing segmented images and meshes of 3D objects drawn from the pattern recognition literature. The *biochemical* dataset [GFM14] contains matching problems taken from real systems of biochemical reactions. The *phase* dataset [MPS18] is comprised of randomly generated Erdős–Rényi graphs with parameters known to be very difficult for state-of-the-art solvers.

We also include a problem set where the template graph is a small Erdős–Rényi graph and the world graph is composed of the webpages on the Notre Dame university website with directed edges representing links between pages [AJB99]. In these instances, the template is randomly generated with n_t vertices and e_t edges where $5 \leq n_t \leq 15$ and $n_t \leq e_t \leq 3n_t$. The world graph is fairly sparse and has 325,729 vertices and 1,497,135 edges. We refer to this problem set as the *www* dataset. We collected 50 template graphs for each value of n_t for a total of 550 templates.

For each instance, we run the algorithm for each equivalence level with the solver configured to count all solutions. We record the number of representative solutions found, the

Dataset	NE	TE	WE	FE	TEWE	CE	NC
biochemical	0.73	0.78	0.83	0.86	0.81	0.84	0.85
LV	0.10	0.10	0.14	0.15	0.13	0.14	0.15
scalefree	1.00	1.00	1.00	1.00	1.00	1.00	1.00
images-cv	1.00	1.00	1.00	1.00	1.00	1.00	1.00
meshes-cv	0.00	0.00	0.00	0.00	0.00	0.00	0.00
si	0.83	0.92	0.83	0.94	0.90	0.85	0.93
images-pr	1.00	1.00	1.00	1.00	1.00	1.00	1.00
phase	0.00	0.00	0.00	0.00	0.00	0.00	0.00
www	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Table 3.3: Proportion of Satisfiable Instances from Benchmark Datasets in Table 4.1 Which Were Fully Enumerated by Various Equivalence Levels

total number of solutions that can be generated by interchanging, as well as the total run time for the instance. We terminate each run if the search is not completed after 600 seconds and record the statistics for the incomplete run. For each run, we measure the compression rate which is the number of representative solutions divided by the total number of solutions found. This quantity indicates the factor by which the form of equivalence chosen decreases the size of the solution space. All experiments were performed on an Intel Xeon Gold 6136 processor with 3 GHz, 25 MB of cache, and 125 GB of memory.

In Table 3.3, we record the proportion of satisfiable problems for which the solution space is fully enumerated by each algorithm within 600 seconds. For the *biochemical*, *LV*, and *si* datasets, there is an increase of 5-10% of all problems fully solved when using any form of equivalence. We further note that full equivalence always has the best performance followed by node cover equivalence. This is not surprising given that Proposition 16 states that full equivalence is the most expansive form of equivalence.

Figure 3.5 portrays how many instances are solved by a given time and demonstrates

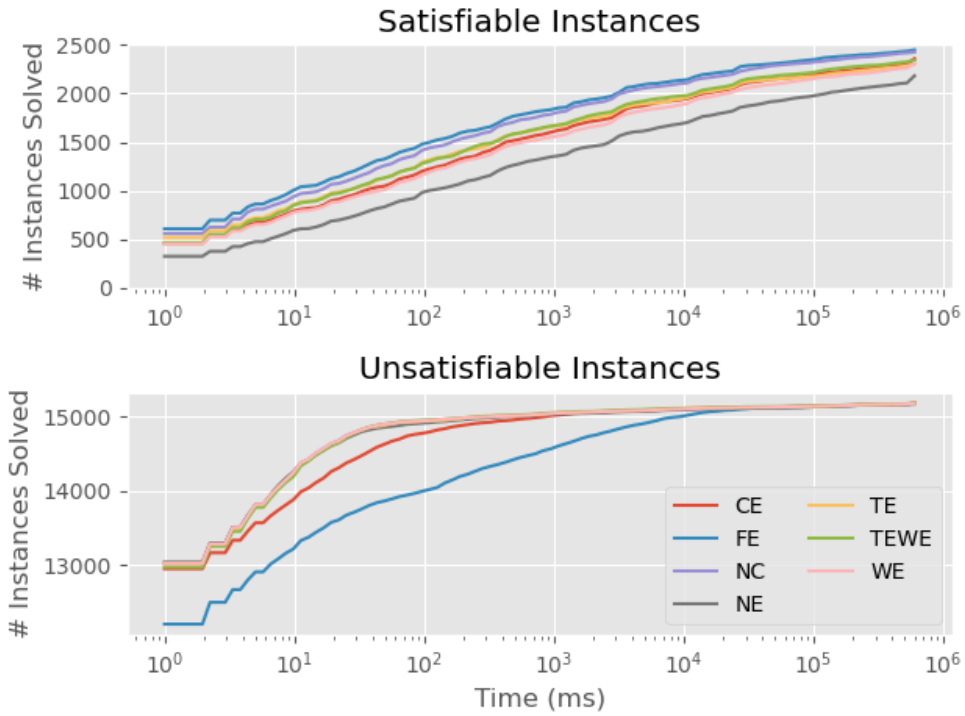

Figure 3.5: Number of satisfiable (top) and unsatisfiable (bottom) instances solved after a given amount of time. This is aggregated over all single channel benchmark data sets. For satisfiable instances, "solved" means having fully enumerated the solution space.

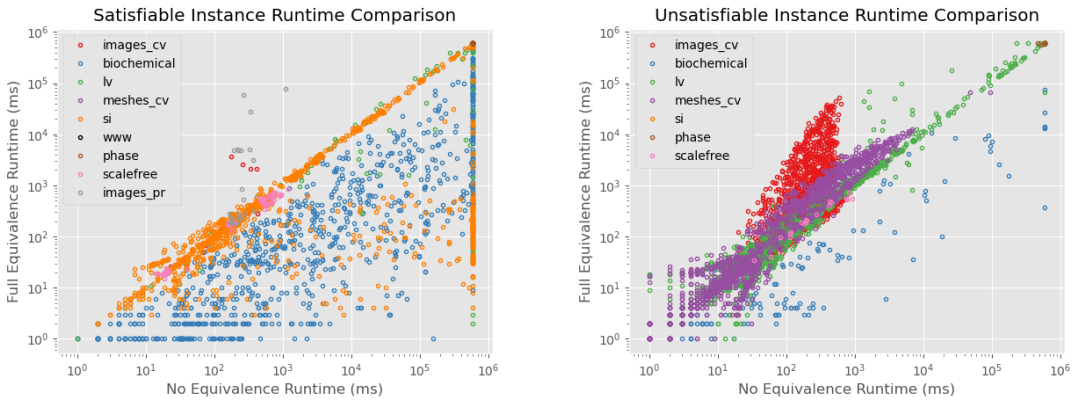

Figure 3.6: Comparison of individual run times for full enumeration between no equivalence and full equivalence runs for satisfiable problems (left) and unsatisfiable problems (right). Note the *phase*, *www*, and *meshes_cv* problems do not terminate for any instance and take the full 600 second runtime, so they can be difficult to discern as each occupies the same spot in the upper right corner of the graphs.

that full equivalence performs best in solution space enumeration for satisfiable problems followed by node cover and the other forms of equivalence. The plot for unsatisfiable problems demonstrates drawbacks to using full candidate equivalence; the additional computation time to check equivalence is not needed if there are no solutions. Checking equivalence in the TE, WE, and NC routines are very cheap operations so there is little difference in the amount of unsatisfiable problems solved compared to the NE routine.

Figure 3.6 demonstrates the variation among and within the data sets by comparing run times for the NE and FE routines. We observe that it is primarily among the *biochemical*, *SI*, and *LV* for which the full equivalence routine vastly outperforms the no equivalence routine often by several orders of magnitude. On the other hand, the *images-cv*, *meshes-cv*, and *images-pr* datasets are more challenging, especially for unsatisfiable problems. This may be due to wasted equivalence checks as there is no solution space to be compressed.

Figure 3.7 includes two plots to illustrate variation in solution counts when using the

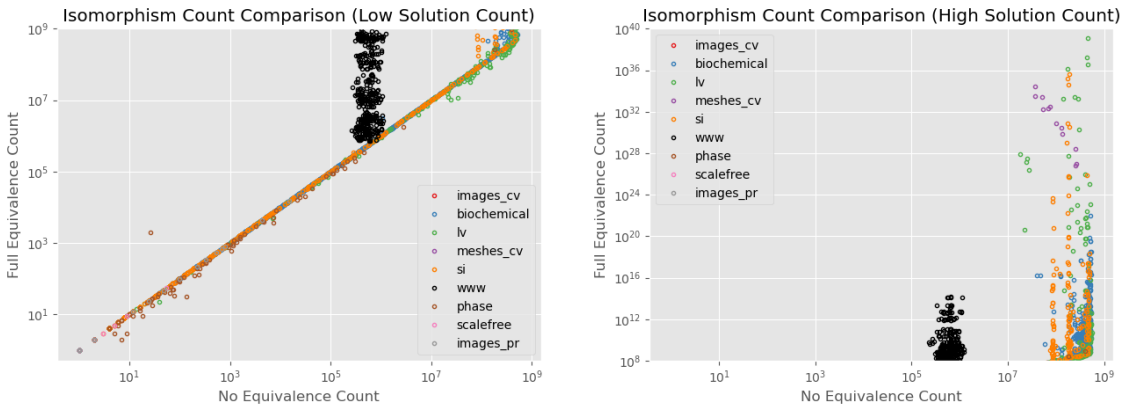

Figure 3.7: Comparison of isomorphism counts for full enumeration between no equivalence and full equivalence runs for problems with small ($< 10^9$) numbers of isomorphisms (left) and problems with large ($\geq 10^9$) numbers of isomorphism (right). Take note of the scales chosen for each graph. For 110 instances the solver with full equivalence found greater than 10^{40} isomorphisms (the largest had $\approx 10^{384}$ isomorphisms), and they are not shown on these graphs.

NE and FE routines. Each has different limits on the axes to emphasize different aspects. The left demonstrates that for problems with fewer than 10^9 isomorphisms, 10 minutes is often enough time to fully enumerate the solution space without using equivalence. The number 10^9 functions as an approximate upper bound for the number of solutions found in 10 minutes for any problem using the original Glasgow solver. We note that for the *www* dataset, the base Glasgow solver finds at most approximately 10^6 solutions, a significantly smaller number. This happens because that a significant portion of time is taken to load in the large world graph. The right plot shows problems for which the FE routine finds many orders of magnitude more solutions. In these highly symmetric problems, an equivalence-based approach is essential to fully understand the solution space. We observe that for the *biochemical*, *si*, *lv*, and *www*, we find many orders of magnitude more solutions when using full equivalence. The largest disparity found between FE and NE solution counts is not displayed - FE found about 10^{384} solutions and NE found roughly 10^9 solutions: a difference

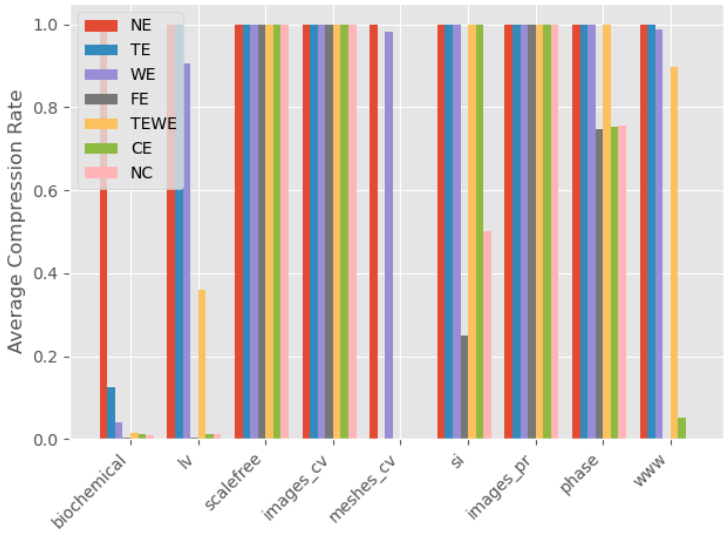

Figure 3.8: The average compression rate for each dataset and equivalence type.

of 375 orders of magnitude.

Figure 3.8 demonstrates the different average compression rates across each dataset and equivalence level. As expected, FE, NC, and CE perform the best in terms of compression followed by TEWE and then depending on the dataset, either TE or WE. For the *biochemical*, *lv*, *meshes_cv*, and *www* datasets, we observe on average, the solution space is compressed in size by an order of magnitude or more when using the CE, FE, or NC methods. On specific particularly symmetric problems from these datasets, we find the solution space can be compressed by tens or even hundreds of orders of magnitude when using FE or NC. For these cases, it is necessary to incorporate equivalence to come close to understanding the set of solutions to the subgraph isomorphism problem.

3.6 Compact Solution Representation

As noted in prior sections, the solution space for subgraph matching is often combinatorially complex. However, the notion of structural equivalence provides methods to diagram the

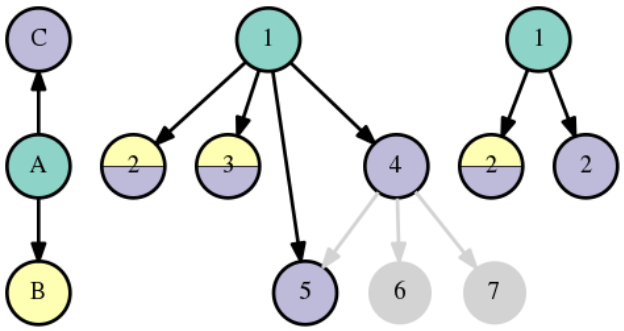

Figure 3.9: The template (left) and world (center) from Figure 3.2 recolored to represent solution $(B \to \{2,3\}, A \to 1, C \to \{2,3,4,5\})$. Each world vertex is colored with the same color as template vertices which can map to it or gray if no vertex maps to it. The right graph compresses the world graph by dropping nonparticipant vertices and combining vertices of the same color into a vertex with a label indicating the amount combined.

solution space in a compact visual way that a user can understand. We explain this first for the toy example from Figure 3.2. We begin with the base representation of one solution, $\{A \to 1, B \to 2, C \to 3\}$, which pairs template vertices with world vertices and extend it to incorporate equivalence.

If we use template structural equivalence, equivalent template vertices are interchangeable. We indicate this by pairing template equivalence classes with world vertices; producing a solution amounts to picking a unique representative from each class. In our example, as B and C are equivalent, we write $\{A \to 1, \{B,C\} \to 2, \{B,C\} \to 3\}$. World equivalence is represented similarly: the representation $\{A \to 1, B \to \{2,3\}, C \to 4\}$ indicates B can match with 2 or 3.

Table 3.4 illustrates the different numbers of representative solutions for each different equivalence level for the toy problem in Figure 3.2. The candidate and node cover equivalence numbers are computed assuming A is assigned first. As the full candidate and node cover equivalence levels have the broadest notion of equivalence, they have the most compression.

If we use candidate or node cover equivalence, equivalence classes are recomputed before

Eq. Level	# Rep. Sols.	Example Sol.
NE	18	$\{A \to 1, B \to 2, C \to 3\}$
TE	9	$\{A \to 1, \{B, C\} \to 2, \{B, C\} \to 3\}$
WE	10	$\{A \to 1, B \to \{2, 3\}, C \to 4\}$
TEWE	6	$\{A \to 1, \{B, C\} \to 5, \{B, C\} \to \{6, 7\}\}$
CE	5	$\{A \to 1, B \to 2, C \to \{3, 4, 5\}\}$
FE	2	$\{A \to 1, B \to \{2, 3, 4, 5\}, C \to \{2, 3, 4, 5\}\}$
NC	2	$\{A \to 1, B \to \{2, 3, 4, 5\}, C \to \{2, 3, 4, 5\}\}$

Table 3.4: Number of representative solutions for each equivalence level in the toy problem in Figure 3.2

each assignment. Hence, it may be the case that a template vertex is paired with a world equivalence class that has been previously assigned but has grown in size due to recomputing equivalence. For example, if we first assign template vertex B to the equivalence class $\{2, 3\}$, we are forced to assign A to 1. Finally, we recompute equivalence, and we find that $\{2, 3, 4, 5\}$ comprise an equivalence class, to which we assign our last template vertex C. We therefore have solution class $\{B \to \{2, 3\}, A \to 1, C \to \{2, 3, 4, 5\}\}$. We diagram this class in Figure 3.9 where we color each template vertex and its associated candidates the same color. The subgraph of all vertices and edges that participate in the representative solution is the **solution-induced world subgraph**. The final graph depicts the **compressed solution-induced world subgraph** where we drop all nonparticipant vertices and edges, and we combine like-colored vertices into "supervertices" with a label indicating the number of vertices joined. From this last graph, we can observe the original template graph structure among the participant world vertices.

We use these graphical representations depict various symmetric features of our datasets. As an example, we plot the template and world subgraph for an example from the *biochemical* dataset in Figure 3.10. From this depiction, we observe that there are multiple different

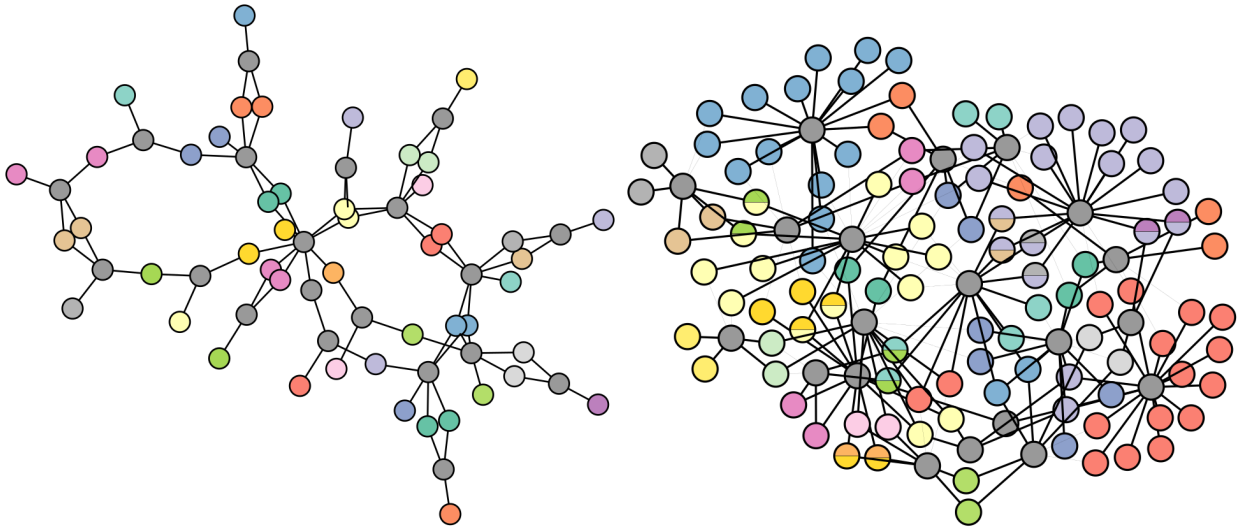

Figure 3.10: A biochemical reactions [GFM14] template graph (left) and the solution-induced world subgraph (right) for a solution class comprised of 9.18×10^{13} solutions. Dark gray vertices are vertices with a single candidate. Vertices with the same non-gray color in the world subgraph are fully candidate equivalent. Vertices with two or more colors were part of one class at an early stage of subgraph search which was later merged into another class. All solutions represented by the compressed solution can be generated by mapping templates vertices of one color to world vertices with the same color.

sources from which equivalence may arise. One aspect is the large number of pairs of structurally equivalent vertices that are colored the same in the template graph. A second source is leaf vertices on the template graph that can be mapped to large equivalence classes in the world graph. By using an equivalence-informed subgraph search, we can expose exactly where these complexities arise. The compressed solution-induced world graph is depicted in Figure 3.11 and clearly shows the role each world vertex plays with respect to the template graph in a solution.

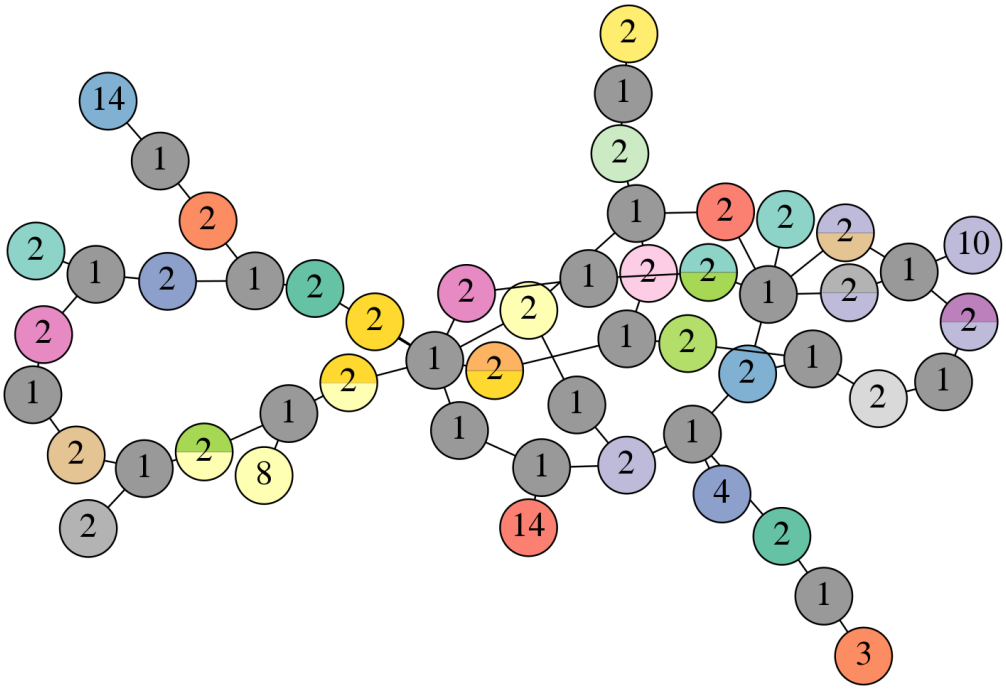

Figure 3.11: The world graph from Figure 3.10 with equivalent vertices joined into super-verticess with numbers indicating the size of the class.

3.7 Application to Multiplex Networks

3.7.1 Multiplex MultiGraph Matching

Often analysts wish to encode attributed information into the vertices and edges of a graph and allow for more than one interaction to occur between vertices. For example, a transportation network may have multiple modes of travel between hubs (e.g., trains and subways). Formally, if we have K distinct edge labels, then a **multiplex multigraph** is a $K+1$-tuple $(\mathcal{V}, \mathcal{E}^1, \mathcal{E}^2, \ldots, \mathcal{E}^K)$ where $\mathcal{V}$ is the set of vertices, and $\mathcal{E}^i : \mathcal{V} \times \mathcal{V} \to \mathbb{Z}_{\geq 0}$ is a function dictating how many edges there are of label i between two vertices. Intuitively, a multiplex multigraph is a collection of K multigraphs which share the same set of vertices. The index i of the edge function is the "channel" i, and we refer to the edges given by edge function E^i as the edges in channel i, and the graph $(\mathcal{V}, \mathcal{E}^i)$ as the graph in channel i.

Multiplex multigraphs are special cases of multilayer graphs [DSC13]. Multilayer graphs in their full generality allow even more complicated interactions in that each vertex can exist on several different layers and edges connect vertices within and between layers. See the surveys [KAB14, BBC14] for a precise formalization of multilayer graphs, related concepts, and key properties which have been studied over the years. The graph isomorphism problem in the complete generality of multilayer graphs has been studied as well in [KP18]. The multiplex graphs which we study are simpler forms of multilayer graphs which require each vertex to exist in each layer and edges can only exist between vertices within a layer (in this context, the layer an edge lies in is a channel).

A **multiplex subgraph isomorphism** $f : \mathcal{V}_t \to \mathcal{V}_w$ preserves the number of edges in each channel. Given template $(\mathcal{E}_t, \mathcal{E}_t^1, \ldots, \mathcal{E}_t^K)$ and world $(\mathcal{V}_w, \mathcal{E}_w^1, \ldots, \mathcal{E}_w^K)$, for any $u, v \in \mathcal{V}_t$, we require $\mathcal{E}_w^i(f(u), f(v)) \geq \mathcal{E}_w^i(u, v)$, i.e., there need to be enough edges between $f(u)$ and $f(v)$ to support the edges between u and v. Definitions for equivalence also extend naturally: we say $v \sim_s w$ if in each channel i for each $u \neq v, w$, $\mathcal{E}^i(v, u) = \mathcal{E}^i(w, u)$, $\mathcal{E}^i(u, v) = \mathcal{E}^i(u, w)$, and $\mathcal{E}^i(v, w) = \mathcal{E}^i(w, v)$. The other forms of equivalence and related theorems all generalize

similarly.

Recently, significant work has been done on developing algorithms for finding multiplex subgraph isomorphisms. [IIP16] develops an indexing approach based on neighborhood structure in multichannel graphs. [MBF20] extends the single channel package [BGP13] to handle the multichannel case and focuses on using intelligent vertex ordering for finding isomorphisms. [MCT18] utilizes a constraint programming approach for filtering out candidates which is extended in [MTC21]. A similar filtering approach is taken in [LDT19]. [SPP19] relaxes the problem to a continuous optimization problem which is then solved and projected back onto the original space.

3.7.2 Multiplex Experiments

We assess the performance of our equivalence enhancements on the Glasgow solver, adapted to handle multiplex subgraph isomorphism problems. The adaptations involve minimal changes to the base algorithm, to ensure that matches are only made if they preserve the edges in every channel. To eliminate more candidates, we also perform a prefilter using the statistics and topology filters from [MCT18] as well as maintain the subgraphs in each channel as the supplemental graphs used in the Glasgow algorithm.

We consider datasets including those from [MTC21] and which represent both real world examples and synthetically generated data. The real world examples include a transportation network in Great Britain [GB15], an airline network [CGZ13], a social network built on interactions on Twitter related to the Higgs Boson [DLM13], and COVID data [ZPM21]. For the transportation and twitter networks, the template is extracted from the world graph. The synthetically generated datasets are examples which represent emails, phone calls, financial transactions, among other interactions between individuals and are all generated as part of the DARPA-MAA program [KSG18, BJU18, CPM18]. The subgraph isomorphisms to be detected may be a group of actors involved in adversarial activities including human trafficking and money laundering. The statistics regarding these different subgraph isomor-

	Template		World		
Dataset	Vertices	Edges	Vertices	Edges	Chan.
Brit. Trans.	53	56	262377	475502	5
Higgs Twitter	115	2668	456626	5367315	4
Airlines	37	210	450	7177	37
PNNL RW	74	35	158	6407	3
PNNL v6-b0-s0	74	1620	22996	12318861	7
PNNL v6-b5-s0	64	1201	22994	12324975	7
PNNL v6-b1-s1	75	1335	22982	12324340	7
PNNL v6-b7-s1	81	1373	23011	12327168	7
GORDIAN v7-1	156	3045	190869	123267100	10
GORDIAN v7-2	92	715	190869	123264754	10
IvySys v7	92	195	2488	5470970	3
IvySys v11	103	387	1404	5719030	5
COVID	28	38	87580	1736985	9
Twitter - ER	5-15	4-31	456626	5367315	4

Table 3.5: Basic Graph Statistics for the Multichannel Graphs

phism problems are described in Table 3.5. For more details on these particular datasets, see [MTC21].

The multiplex datasets are much larger than the single-channel graphs in the previous section, with the largest world graphs having hundreds of thousands of vertices and hundreds of millions of edges. The synthetic datasets are divided into three groups based on which organization generated the dataset: PNNL [CPM18], GORDIAN [KSG18], and IvySys Technologies [BJU18].

For our experiments, we examine the same seven modes of equivalence used in the single channel case, but with a time limit of one hour to count as many solutions as possible. These experiments were run on the same computer using our adapted version of the Glasgow solver. The amount of time required to enumerate all the solutions is displayed in Table 3.6 and the number of solutions found with a given method is displayed in Table 3.7. A quick inspection of the times illustrates that in a few cases (Airlines, GORDIAN, and Higgs Twitter), using full equivalence can enumerate the full solution space an order of magnitude faster than any other approach. This speedup is reflected in the solution count table for which FE finds significantly many more solutions. The other methods only find a mere fraction of the total solutions. The NC method often appears to be the second best both in terms of solutions found and time taken to enumerate all. This makes sense given Proposition 17. TE appears to be the third best method which can be explained by the simplicity of implementation and having no need to recompute equivalence. WE and CE are not competitive with the other methods. The datasets bear different qualities that illustrate why certain levels of equivalence work better than others. We discuss a few datasets in detail.

3.7.2.1 PNNL

The PNNL template and world graphs [CPM18] are generated to model specific communication, travel, and transaction patterns from real data and the templates are then embedded into the world graph. For these instances, the counting problem is almost entirely solved after applying the initial filter and the solution space is understood by equivalence in the template. For example, observe the template displayed in Figure 3.12. The number of solutions generated equals the count of solutions generated by permutations of the template vertices for a single representative solution. We have a group of 9, a group of 4, and two groups of 3 interchangeable vertices, meaning any solution can generate 9!4!3!3! more solutions. All variants on the PNNL problems illustrate this behavior.

Algorithm Dataset	CE	FE	NC	NE	TE	TEWE	WE
Brit. Trans.	3600	3600	3600	3600	3600	3600	3600
Higgs Twitter	3600	**369**	456	3600	3600	3600	3600
Airlines	0.34	**0.24**	1985	3600	1329	3600	3600
PNNL RW	3600	3600	3600	3600	3600	3600	3600
PNNL v6-b0-s0	41.6	42.1	41.8	**41.3**	41.7	41.4	41.6
PNNL v6-b1-s1	241	**240**	241	**240**	242	**240**	**240**
PNNL v6-b5-s0	62.6	58.1	58.9	**58.0**	62.3	62.3	63.0
PNNL v6-b7-s1	1133	200	201	3600	**188**	211	1138
GORDIAN v7-1	3600	**327**	3600	3600	3600	3600	3600
GORDIAN v7-2	3600	**316**	3600	3600	3600	3600	3600
IvySys v7	3600	3600	3600	3600	3600	3600	3600
IvySys v11	3600	3600	3600	3600	3600	3600	3600
COVID	3600	3600	3600	3600	3600	3600	3600
Twitter - ER	514.7	505.4	**494.8**	536.0	536.7	544.2	541.3

Table 3.6: Time (s) to enumerate solution spaces of multichannel problems. Experiments were timed out at one hour. Bolded entries indicate the equivalence algorithm which fully enumerated all subgraph isomorphisms the quickest.

Algorithm Dataset	CE	FE	NC	NE	TE	TEWE	WE
Brit. Trans.	1.5e+11	**2.3e+15**	5.0e+08	1.3e+07	2.5e+12	2.0e+12	1.2e+07
Higgs Twitter	1.4e+14	**3.2e+14**	**3.2e+14**	5.7e+06	6.4e+06	5.7e+06	7.0e+06
Airlines	**3.7e+09**	**3.7e+09**	**3.7e+09**	3.6e+09	3.7e+09	8.1e+08	2.4e+09
PNNL RW	3.5e+09	2.8e+11	**4.7e+11**	8.6e+08	2.0e+10	5.0e+09	8.7e+08
PNNL v6-b0-s0	**1.2e+03**	**1.2e+03**	**1.2e+03**	**1.2e+03**	**1.2e+03**	**1.2e+03**	**1.2e+03**
PNNL v6-b1-s1	**1.2e+03**	**1.2e+03**	**1.2e+03**	**1.2e+03**	**1.2e+03**	**1.2e+03**	**1.2e+03**
PNNL v6-b5-s0	**1.2e+03**	**1.2e+03**	**1.2e+03**	**1.2e+03**	**1.2e+03**	**1.2e+03**	**1.2e+03**
PNNL v6-b7-s1	**3.1e+08**	**3.1e+08**	**3.1e+08**	8.6e+07	**3.1e+08**	**3.1e+08**	**3.1e+08**
GORDIAN v7-1	1.1e+12	**9.1e+12**	1.4e+10	1.6e+07	7.8e+09	5.3e+09	1.6e+07
GORDIAN v7-2	2.1e+11	**1.4e+16**	1.2e+15	1.7e+07	3.9e+08	3.2e+08	1.6e+07
IvySys v7	2.0e+14	**8.0e+96**	2.1e+90	1.8e+09	2.7e+47	5.8e+45	5.4e+07
IvySys v11	7.4e+10	**3.6e+89**	5.1e+66	1.8e+09	4.4e+72	6.6e+71	4.9e+07
COVID	5.3e+14	**7.5e+21**	3.1e+20	9.6e+06	9.5e+06	4.5e+07	1.3e+07
Twitter - ER	3.5e+08	8.8e+09	**1.4e+11**	7.2e+05	7.4e+05	6.1e+05	6.8e+05

Table 3.7: Number of solutions found for multichannel subgraph isomorphism problems listed in Table 3.5 within one hour. Bolded entries indicate the algorithms which found the most solutions in the allotted time.

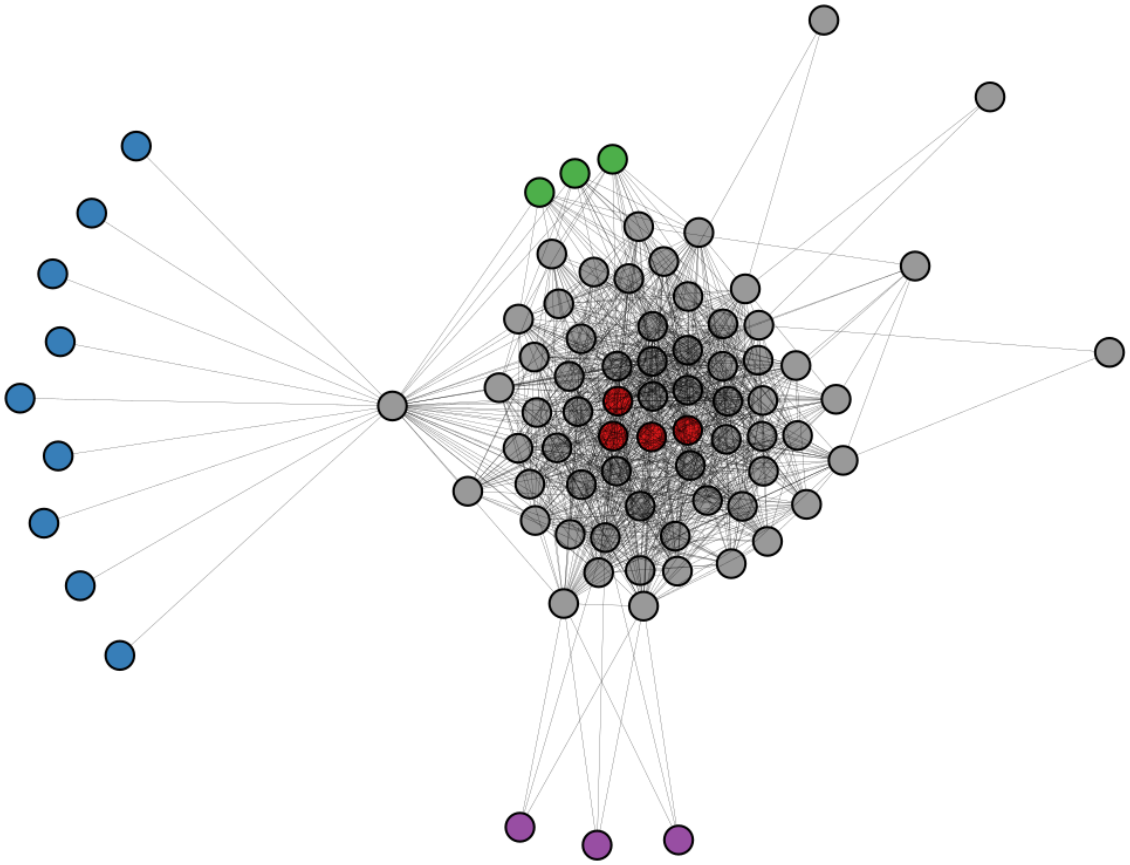

Figure 3.12: Template Graph for PNNL v6-b7-s1. Non-gray vertices of the same color are structurally equivalent.

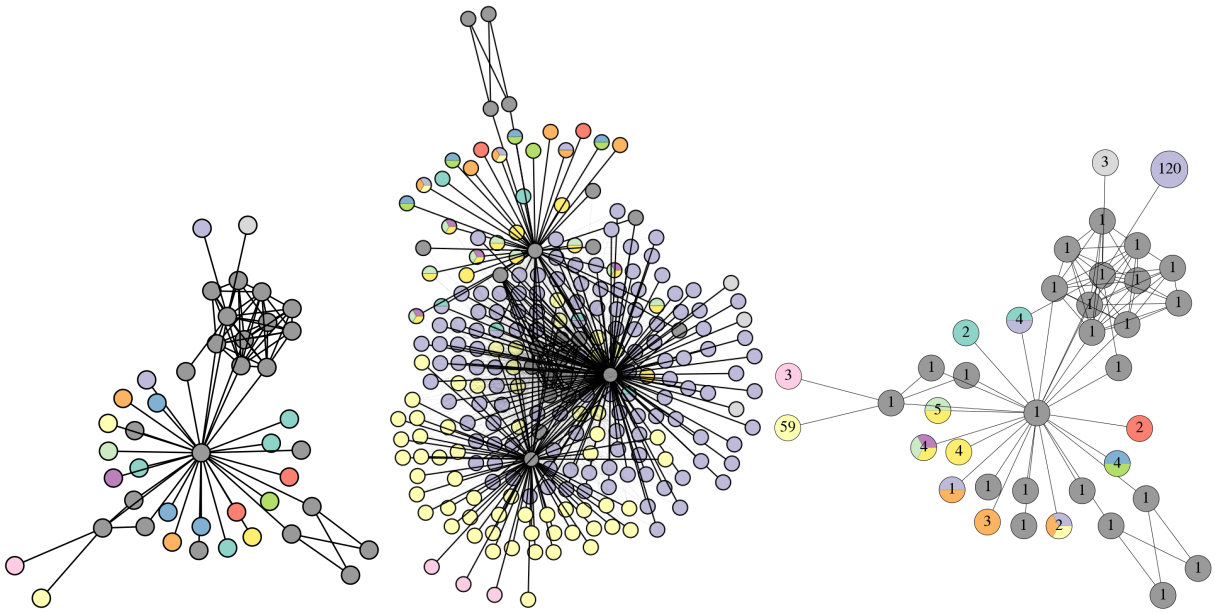

Figure 3.13: Template (left), solution-induced world subgraph (middle) and the compressed solution-induced world subgraph (right) for a solution class which can generate about 3×10^{12} solutions to GORDIAN v7-2 [KSG18]. World vertices of the same color are fully candidate equivalent and are candidates of the template vertex of the same color. All solutions represented by this compressed solution can be generated by mapping each colored vertex to one of groups of world vertices with the same color.

3.7.2.2 GORDIAN

The GORDIAN datasets [KSG18] have a much larger templates and worlds than PNNL and they are generated separately in an agent-based fashion to match the daily routines and travel patterns of a certain population of people. Only the FE method fully enumerates the solution space, but the NC and CE methods come close to a full enumeration. Figure 3.13 illustrates the symmetries for one solution class; the template graph possesses a large group of leaf vertices. After mapping the central vertex to a candidate, the leaves need only be mapped to neighbors of this candidate. These graphs demonstrate a trade-off between vertex specificity and symmetry: template vertices with fewer edges exhibit great amounts

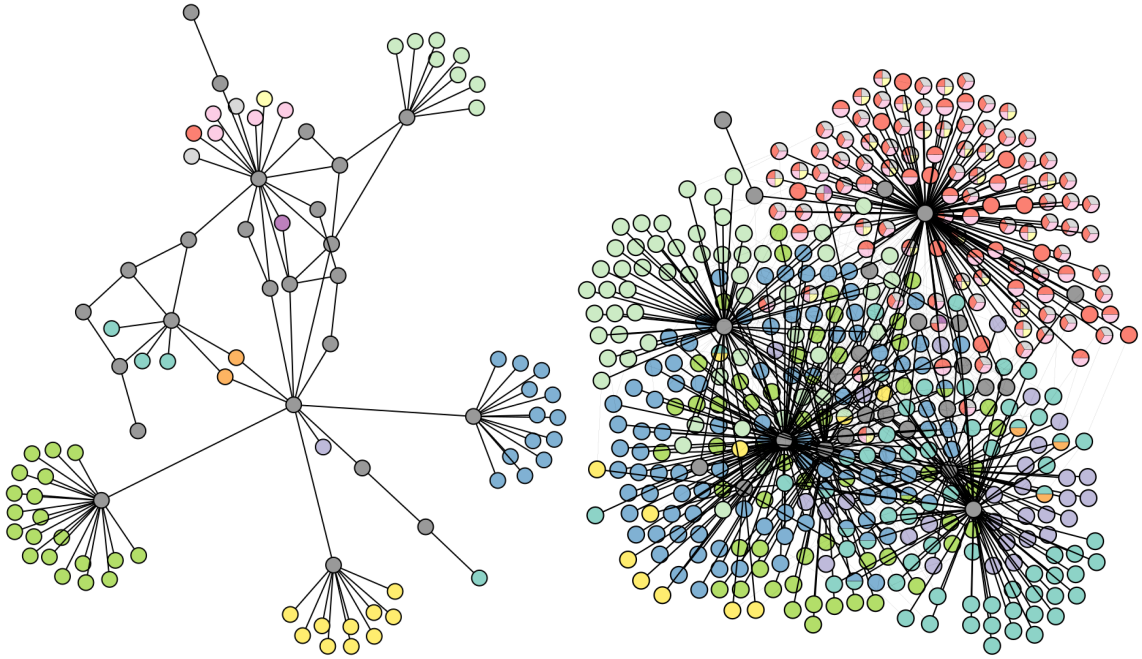

Figure 3.14: Template (left) and solution-induced world subgraph (right) for a solution class from which 7.82×10^{103} solutions to IvySys v7 [BJU18] can be generated. World vertices of the same color are fully candidate equivalent and are candidates of the template vertex of the same color. All solutions represented by this compressed solution can be generated by mapping each colored vertex to one of groups of world vertices with the same color.

of symmetry, whereas dense subgraphs are restricted in their candidates and have minimal symmetry. The right graph in Figure 3.13 depicts a compressed version of the world graph induced by this solution class from which 3×10^{12} solutions may be generated.

3.7.2.3 IvySys

The IvySys template and world graphs [BJU18] are separately generated to match the degree distribution and email behavior of the Enron email dataset and have the most complex solution space. None of the methods were successful at enumerating all solutions. The vastness of the solution space is in contrast to the size of the graphs which only have thousands

of vertices. The complexity emerges from the preponderance of template leaf vertices as shown in Figure 3.14, depicting one particularly large solution class. Figure 3.15 depicts the compressed representation of the world subgraph for this solution as well as a Venn diagram displaying candidates of certain template vertices.

The TE solver finds an astonishing 2.7×10^{47} solutions for IvySys v7. However, using the FE method still dramatically increases the solution count to 8×10^{96}, by mapping these large template equivalent classes into larger world equivalence classes. An equivalence-informed subgraph search is essential as the NE method finds only 1.75×10^{9} solutions, 87 orders of magnitude less than the FE search. Furthermore, a typical subgraph search would assign each group of leaf vertices sequentially meaning only the candidates of the last group would be explored. Incorporating symmetry gives a fuller vision of the solution space.

3.7.2.4 COVID

We lastly apply our algorithm to the problem of querying a knowledge graph representing known causal relations between a large variety of biochemical entities. This problem arises from a desire to extracting causal knowledge in an automated fashion from the research literature. In [ZPM21], a knowledge graph is assembled from multiple sources including the COVID-19 Open Research Dataset [WLC20], the Blender Knowledge Graph [WLW21], and the comparative toxigenomics database [DWW21]. The authors of [ZPM21] then create a query representing how SARS-CoV-2 might cause a pathway leading to a cytokine-storm in COVID-19 patients, but is generalized to detect other possible confounding factors in the pathway.

When rephrased as a multichannel subgraph isomorphism problem, template and world vertices represent biochemical entities. Some template vertices are specified, and others are labeled as a chemical, gene or protein. The 9 channels in this problem are various known types of interactions between entities, e.g., activation. A solution is an assignment of each vertex which has the desired chemical interactions.

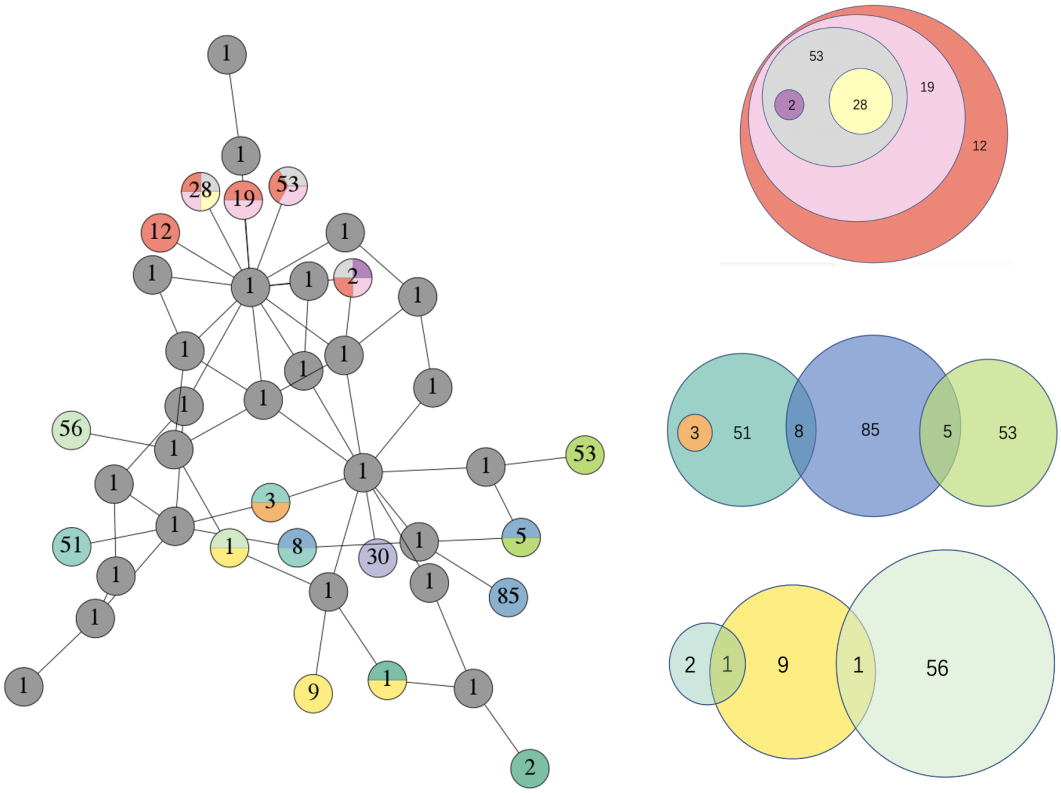

Figure 3.15: IvySys v7 [BJU18] Compressed solution-induced world graph (left) and the Venn diagram representation of intersecting candidate sets in world graph(right) for a solution class from which 7.82×10^{103} solutions to can be generated. The number in each section in the Venn diagram represents the size of a node cover equivalence class in the world graph. All solutions represented by this compressed solution can be generated by mapping each colored vertex in the template to the set in the Venn diagram with the same color.

As can be seen in Tables 3.6 and 3.7, there is an abundance of solutions to this problem, and incorporating equivalence greatly enhances our ability to understand the solution space. Figure 3.17 depicts the template and Venn diagrams of candidates sets for one solution class and exposes unspecified template vertices with a large amount of candidates. Such information is useful to an analyst for determining confounding factors in a pathway and suggesting label information or interactions to add to better specify the entire solution space.

3.7.2.5 Higgs Twitter Erdős–Rényi Experiments

Lastly, we perform a similar experiment as we did for single channel graphs using small Erdős–Rényi graphs as our templates and our largest graph, the Higgs Twitter dataset, as our world graph. We generate a multichannel template graph by overlaying 4 different graphs corresponding to each channel each generated as an Erdős–Rényi graph with $p = \frac{\log n_t}{8n_t}$ where n_t is the number of template vertices. This value p is chosen so that the graph will be connected with high probability. We generate 45 connected graphs in this way for each of $n_t = 5, 7, 9, 11, 13, 15$. We then compute the number of isomorphisms counted for each method within 10 minutes. For these problems, we precompute the world structural equivalence classes of the Higgs Twitter graph prior to running our algorithms. The average isomorphism count for each equivalence method and template size is depicted in Figure 3.16. The overall averages for total runtime and isomorphism count are included in Tables V and VI under the *Twitter-ER* dataset.

From these results, we observe that the NC, FE, and CE methods find significantly more solutions than the base routine whereas the other equivalence methods do not improve on the NE method. NC performs the best both in terms of the number of isomorphisms found and the total amount of time which we speculate is due to its lightweight computation and ability to capture most of the equivalence. FE and CE are a few orders of magnitude worse, and the remaining methods TE, WE, and TEWE fail to provide significant benefit over the base method and in fact does worse when involving world structural equivalence. That

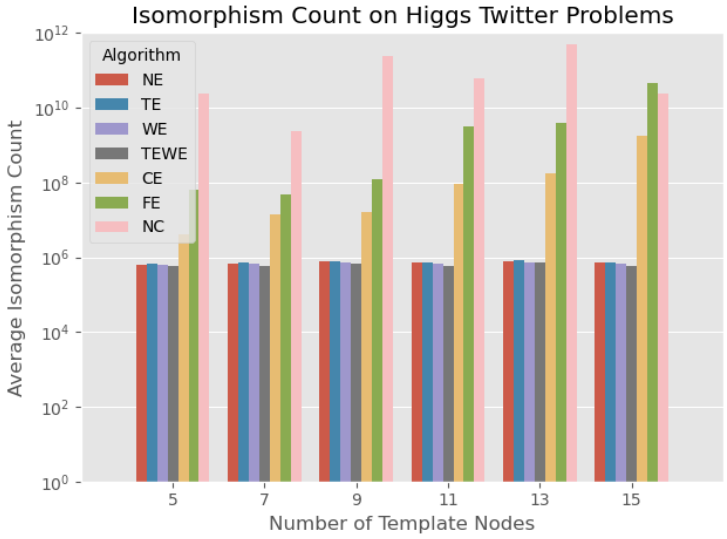

Figure 3.16: The average number of subgraph isomorphisms found for each equivalence level where the templates are small Erdős–Rényi graphs and the world is the *Higgs Twitter* graph.

these methods do not improve much we can explain by the fact that vertices in multichannel Erdős–Rényi graphs are fairly unlikely to be structurally equivalent. All in all, these experiments demonstrate even when using randomly generated template graphs, significant improvements can be had in incorporating equivalence into the algorithm. However, certain modes of equivalence may be more appropriate for certain classes of graphs and some care must be taken to ensure that the level of equivalence chosen actually helps with solving the problem.

3.8 Conclusion

In this chapter, we have developed a theory for static and dynamic notions of equivalence and presented conditions under which vertex assignments can be interchanged while preserving isomorphisms. With minimal changes to a subgraph isomorphism routine to incorporate equivalence during a tree search, we can dramatically reduce the amount of time to solve a problem and get a compact characterization of the solution space. For instances with

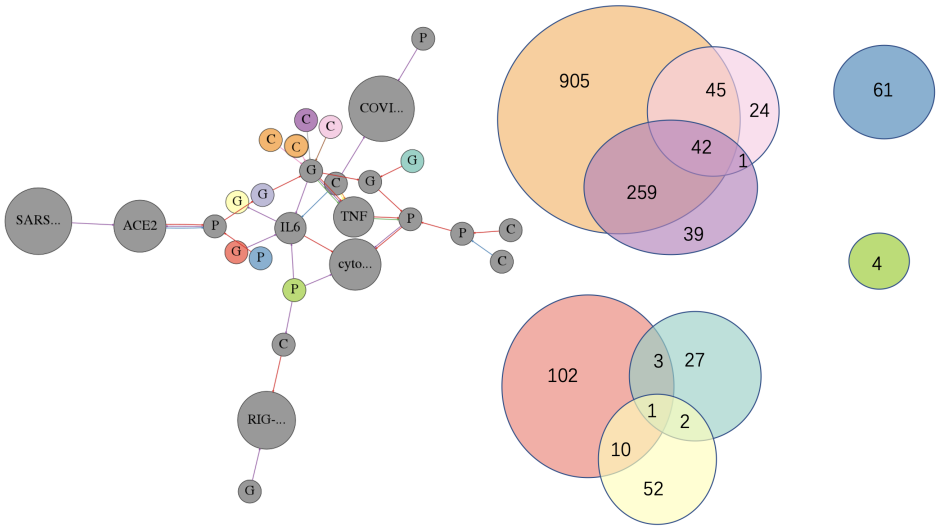

Figure 3.17: COVID-19 [ZPM21] template (left) and the Venn diagram of candidate sets in world graph (right) from which 2.6×10^{18} solutions can be generated in one solution class. Each section in the Venn Diagram represents a node cover equivalence class, and the number in the section is the size of the class. A few template vertices were specified at the start whereas others simply received a vertex label of C, P, or G indicating chemical, protein, and gene respectively. The solutions may be generated by mapping non-gray template vertices of one color to world vertices in the Venn diagram section of the same color.

minimal symmetry, little is to be gained, but for problems with large symmetric structures, it is essential to exploit equivalence in order to understand the large solution space. In particular, we demonstrated that the FE and NC methods both perform well in capturing equivalence present in the problem enabling the greatest compression of the solution space. We showed our results apply to standard subgraph solvers by integrating our methods into the state-of-the-art solver Glasgow and extended our methods to the more complex problem spaces of multiplex multigraphs.

Future directions for this research include adapting these notions of equivalence to inexact search as well as producing inexact forms of equivalence. We would also like to better understand how to incorporate automorphic equivalence with the different notions of equivalence discussed in this chapter.

CHAPTER 4

Iterative Active Learning Strategies for Subgraph Matching[1]

4.1 Introduction

Despite the simple statement of the subgraph matching problem, both real world and synthetic examples illustrate the complexity of the set of all subgraph isomorphisms. As can be seen from the benchmark subgraph isomorphism problems in the prior chapter, even problems with relatively small templates and worlds may still have trillions of possible subgraph isomorphisms. This abundance of solutions may arise from the existence of multiple good candidates for matches for our template graph of interest. Alternatively, it may be the case that our template graph is underspecified, and if we could add more information to our template in the form of more edges or labels, we may be able to eliminate the vast majority of these subgraph isomorphisms as irrelevant.

In a real use-case scenario, we often wish to identify exactly one specific subgraph isomorphism out of the many possible matchings. There are a number of reasons why this would be - for example if the world graph represents data related to an investigation involving an unknown actor, such as in a homicide investigation, it would be important to identify the actual person involved. The consequences of misidentifying someone could be grave - both for the person wrongly identified and for potential future victims of the actual person involved.

[1]**This chapter is adapted from [GYB23] which is currently under review.**

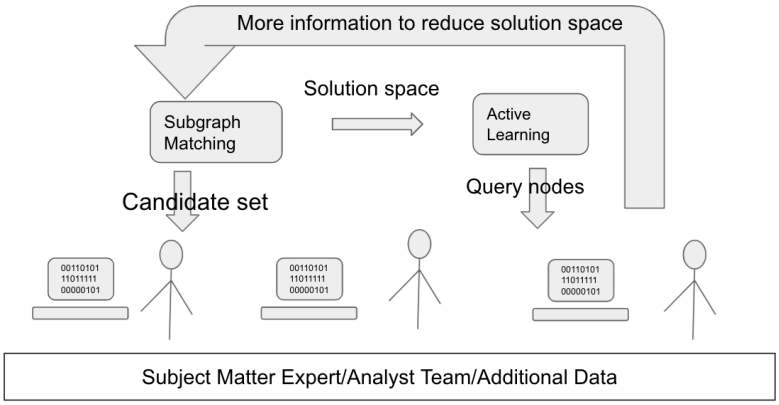

Figure 4.1: Active learning flowchart for subgraph matching [GB21]. First, a subgraph matching algorithm determines all potential candidates for template vertices. Then an active learning algorithm determines the optimal vertices for subject matter experts to obtain information about. These two steps are repeated until the number of subgraph isomorphisms is reduced to a desired amount

In some cases, there may be more than one subgraph isomorphism of relevance, for example in the case of identifying different but equally important pathways in a biochemical reaction network [GFM14] or the case of identifying groups involved in human trafficking or smuggling. Likewise, organizations or people interested in identifying those wrongly accused of crimes could look at a knowledge graph of information that might present alternate scenarios. In a real life setting, this could entail additional constraints added to the problem space such as attributes for the vertices (e.g. names, dates, times etc). Such information might come at a cost and therefore it would be of interest to understand strategies to reduce the complexity of the solution space with the minimal cost.

Active learning is an area of research in statistical machine learning that involves a subject matter expert (SME) as a human in the loop in algorithms for classification of points in a dataset. Supervised machine learning algorithms require an abundance of labeled data. In the real world however, unlabeled data is common and accurate labeling may require human

involvement that can not be crowd-sourced due to privacy or security reasons. The choice of labelled data often affects classifier performance, and so the development of methods to determine which data points to label can significant improve accuracy. Active learning involves the use of an algorithm or formula to choose individual data points for labeling by a SME. Then the newly labeled data are included in the semi-supervised learning problem. These active learning methods iterate between the following procedures: (1) Training a model given the current labeled data (2) Choosing one or a batch of query points in the unlabeled set based on an active learning criterion such as an acquisition function. Most active learning acquisition functions for statistical machine learning belong to one of a few categories: uncertainty [Set12, HHG11, GIG17], margin [TK01, BBL06, JG19], clustering [DH08, MM19], and look-ahead [ZLG03, CZZ13].

We can come up with an analogous routine for subgraph matching. A flowchart describing how this approach might be used in a real world setting in shown in Fig. 4.1. Initially, we obtain the candidate set for all template vertices using a filter based subgraph matching algorithm which loosely represent the set of possible solutions. Then, an active learning algorithm determines the optimal vertices for subject matter experts to obtain additional information about. After that, the additional information is fed back into the subgraph matching algorithm filter out more candidates. We repeat this procedure until we can reduce the number of candidates for any given template vertex to a manageable amount (just one if we want to identify a unique solution).

Researchers have introduced active learning into problems similar to subgraph isomorphism problems such as the network alignment problem. This problem tries to find an optimal mapping of graph vertices with maximum similarity between the vertices and edges, in which a cost function measures differences between the vertices or edges. The optimal solution with least cost is given by updating the probability distribution for each vertex. Prior research shows that better alignment can be achieved by introducing interaction with a human to obtain extra information on certain vertices. For example, in [SC15], researchers

compare three probability matrix based query strategies. In [CS20], the cost function for network alignment is updated after interaction with human. In [MGT17], the authors examine the case where experts only provide partial information about the mapping of certain vertices instead of the exact answer. In [PPL20], the authors study the problem of vertex nomination in the inexact subgraph matching problem.

The objective of this chapter is to explore the implementation of an active learning scenario in the context of subgraph matching. This chapter is a significant extension of previous work [GB21] which rigorously defines the problem, establishes key propositions about the active learning problem, and assesses various active learning strategies on a large, diverse dataset. The outline of this chapter is as follows: In section 4.2, we introduce the key concepts and terminologies of the subgraph matching problem and the formal definitions of the active learning framework. In section 4.3, we will study the complexity of the active learning problem we defined and prove that it is in fact NP-complete. In section 4.4, we will formulate different active learning strategies for querying template vertices. In section 4.5, we present the assess the performance of our strategies on the subgraph isomorphism benchmark datasets and discuss advantages and disadvantages to each querying strategy.

4.2 Active Learning Framework

In this section, we will present the general algorithmic framework we will be using to test various active learning strategies. For each experiment, we will assume we have a template graph $\mathcal{G}_t$, world graph $\mathcal{G}_w$, and a ground truth subgraph matching f which we are trying to determine. $f(t)$ will represent the ground truth world vertex which is associated with a given template vertex t.

As a toy example of how the active learning problem may proceed, we consider the example template and world presented in Figure 4.2. Initially, there are four possible subgraph isomorphisms in the world graph and the ground truth matching maps vertices 1 and 2 to

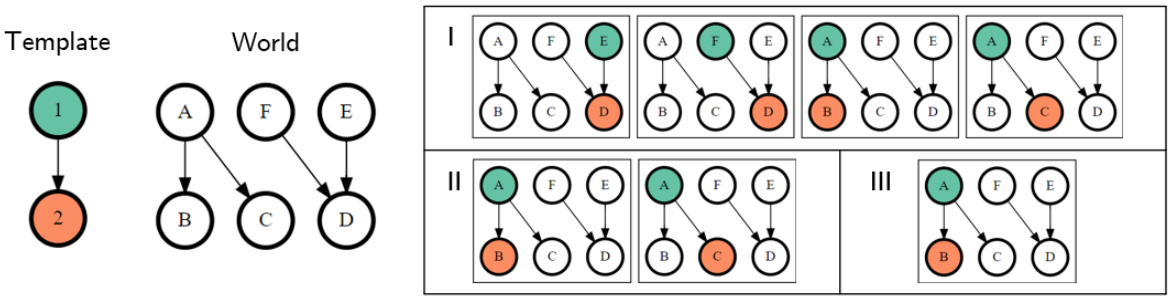

Figure 4.2: Solution spaces for example template and world graph following active learning queries. The ground truth subgraph matching is given by mapping 1 to A and 2 to B. In I, there are initially four possible SIs, in II, there are two SIs after querying template vertex 1 and finding it maps to A, and in III, finally we have reduced the solution space to one SI after querying vertex 2 and finding it maps to B.

A and B, respectively. At this stage, we query template vertex 1 and determine its true assignment, A. From this knowledge, we can rule out any solution which maps 2 to a world vertex not adjacent to A eliminating two solutions. Then, with two solutions left we query template vertex 2 finding it maps to B which fully determines the true subgraph matching after two queries. However, if we instead had queried template vertex 2 first, we would have found it mapped to B, which only one of the candidate solutions does, and we can determine the solution in 1 query. From this simple example, we observe that having a clever strategy for querying template vertices is important to minimizing the total work needed to find a solution.

We now present in Algorithm 4 the basic framework by which we perform active learning for the subgraph matching problem. The algorithm proceeds by alternately filtering the current candidate sets of each template vertex based on the filtering criteria described in [MTC21] and querying for a template vertex according to a given strategy.

The QueryStrategy function picks out a template vertex to query and is the central object of study in this chapter and we will discuss it in detail in Section 4.4. The Match function

Algorithm 4 Active Learning Template Query Loop

Input: Template $\mathcal{G}_t$, World $\mathcal{G}_w$, Matching f

$C \leftarrow \text{Filter}(\mathcal{G}_t, \mathcal{G}_w)$ ▷ Initialize domains

$count \leftarrow 0$

while Not all t in $\mathcal{V}_t$ have 1 candidate **do**

$t \leftarrow \text{QueryStrategy}(\mathcal{G}_t, \mathcal{G}_w, C)$

$\text{Match}(t, f(t))$

$C \leftarrow \text{Filter}(\mathcal{G}_t, \mathcal{G}_w, C)$

$count \leftarrow count + 1$

Return *count*

formally associates t and $f(t)$, and then the Filter function eliminates candidates based on the various filters. Once filtering has reduced the size of the candidate sets of each template vertex to one candidate, the matching has been determined, and we report the number of queries made.

We now formally state the problem we will address:

Definition 18 (Optimal Template Query Problem)**.** *Given $\mathcal{G}_t, \mathcal{G}_w$, candidate sets $C(t)$ for $t \in \mathcal{V}_t$, and a fixed choice of filter, which query strategy will require the fewest template queries as given by Algorithm 4?*

Our metric is the number of queries required to fully determine a subgraph matching. We envision this as the most expensive operation as it potentially requires expert involvement to perform the query. In our problem, each template vertex requires the same amount of work to query, but future work may study an extension of this problem which varies the work required on a per-vertex basis. We also note that this problem depends on the choice of filter. Stronger filters (filters which eliminate more candidates) will give us more information and it is possible that strategies which make better use of this information may perform better in that context. We will sidestep this problem instead opting to fix the choice of filter to

that of the LAD filter introduced in [Sol10].

4.2.1 Associated Theoretical Problems

Within this framework, we can consider two similar theoretical problems given a template $\mathcal{G}_t$ and a world $\mathcal{G}_w$. The first problem involves a candidate solution $f \in \mathcal{F}(\mathcal{G}_t, \mathcal{G}_w)$, and we wish to determine the minimal number of template vertices we need to query to verify that f is the true solution. This is formalized in the following definitions:

Definition 19 (Solution Verifying Set)**.** *Given $\mathcal{G}_t$, $\mathcal{G}_w$ and $f \in \mathcal{F}(\mathcal{G}_t, \mathcal{G}_w)$, a **solution verifying set** is a subset $A \subset \mathcal{V}_t$ such that if we have $g \in \mathcal{F}(\mathcal{G}_t, \mathcal{G}_w)$ with $g(t) = f(t)$ for all $t \in A$, then $g = f$.*

Definition 20 (Minimal Solution Verifying Set Problem)**.** *Given $\mathcal{G}_t$, $\mathcal{G}_w$ and $f \in \mathcal{F}(\mathcal{G}_t, \mathcal{G}_w)$, what is a solution verifying set for f of minimal size?*

In the example in Figure 4.2, we observe that for any of the solutions, we need only to query one template vertex to verify a given solution. For solutions 1 and 2, we query vertex 1 and for solutions 3 and 4, we query vertex 2.

For the second problem, we do not have a candidate solution. Rather, we wish to know the minimal size of a subset of $\mathcal{V}_t$ for which if we knew the images of these vertices, we could uniquely identify the images of the remaining vertices. We introduce the following definition and problem to this end:

Definition 21 (Determining Set)**.** *Given $\mathcal{G}_t$ and $\mathcal{G}_w$, a **determining set** is a subset $A \subset \mathcal{V}_t$ where for every $f, g \in \mathcal{F}(\mathcal{G}_t, \mathcal{G}_w)$ if $f(t) = g(t)$ for $t \in A$, then $f = g$.*

Definition 22 (Minimal Determining Set Problem)**.** *Given $\mathcal{G}_t$ and $\mathcal{G}_w$, what is a determining set of minimal size?*

For the problem in Figure 4.2, we need to query both template vertices 1 and 2 to determine the subgraph isomorphism in all cases. If we query template vertex 1 and 1 maps

to world vertex C, there are still two SIs which are possible. Similarly, if we query template vertex 2 and find that 2 maps to world vertex D, there are also two possible SIs. Hence, the minimal determining set is $\{1, 2\}$ and is of size 2. Note that this is a counterexample to the statement that the size of the minimal determining set is the maximal size of a minimal verifying set over all SIs $f \in \mathcal{F}(\mathcal{G}_t, \mathcal{G}_w)$.

The decision variants of both of these problems are NP-complete. We will prove that the solution verification set problem is NP-complete in Section 4.3 by reduction from the minimum set cover problem. As for the minimal determining set problem, we note that in the case that $\mathcal{G}_t = \mathcal{G}_w$, finding a determining set of a certain size is equivalent to finding a base for the automorphism group of $\mathcal{G}_t$, which is already known to be NP-complete [Bla92].

In practice, the optimal template query problem is intermediate to both of these theoretical problems as we will generally not have a solution f at our disposal (and almost certainly not the whole set of subgraph isomorphisms $\mathcal{F}(\mathcal{G}_t, \mathcal{G}_w)$). However, we may have some prior information, and in the process of querying template vertices, we will be gathering additional information which will aid us in determining the ground truth subgraph isomorphism.

4.3 NP-Completeness of the Minimal Solution Verification Set Problem

In this section, we will present results on the complexity of the optimal template query problem. We first introduce the set cover problem, which is well known for being NP-complete [Kar72]. Then we prove that the minimum set cover problem is reducible to the minimal solution verifying set problem. As the optimal template query problem is an extension of the minimal solution verifying set problem, it is also as hard as the set cover problem.

4.3.1 Reduction of Minimum Set Cover to Minimum Solution Verification Set

We define the minimum set cover problem as follows:

Definition 23 (Minimum Set Cover Problem)**.** *Suppose S is a set $S = \{1, 2, \ldots, m\}$, and we have k subsets $S_i \subseteq S$, $1 \leq i \leq k$. The* ***minimum set cover problem*** *is to find an index set $I \subseteq \{1, 2 \ldots, k\}$ with minimum cardinality, such that $S \subseteq \cup_{i \in I} S_i$.*

We assume the problem is non-trivial and has at least one set cover, i.e., $\cup_{i=1}^{n} S_i = S$. We let m denote the cardinality of S and without loss of generality, we write $S = \{1, \ldots, m\}$.

Given an instance of the minimum set cover problem $(S, \{S_1, \ldots, S_n\})$, we will produce an equivalent instance of the solution verification set problem. The basic idea behind this proof is to construct a template graph and world graph where each vertex in the template graph will correspond to one of the subsets S_i. Each element of S will correspond to a subgraph isomorphism which we want to rule out and querying a template vertex will correspond to ruling out all isomorphisms in that subset.

We choose our template graph to be a complete undirected graph on n vertices and we denote the vertices by $\mathcal{V}_t = \{1, \ldots, n\}$. We impose that each vertex has a label given by the labeling function $\mathcal{L}_V(i) = i$. We define our world graph $\mathcal{G}_w = (\mathcal{V}_w, \mathcal{E}_w)$ where the vertices are given by the following expression:

$$\mathcal{V}_w = \{(i, 0) : i = 1, \ldots, n\} \cup \{(i, j) : j \in S_i\}. \tag{4.1}$$

Given two different vertices $(i_1, j_1), (i_2, j_2) \in \mathcal{V}_w$, there is an edge between them if any of the three conditions hold: (1) $j_1 = j_2$, (2) $j_1 = 0$, $j_2 \notin S_{i_1}$, or (3) $j_2 = 0$ and $j_1 \notin S_{i_2}$. We then label each vertex according to the label function $\mathcal{L}_V((i, j)) = i$.

For an example depiction of these graphs, in Figure 4.3, we show the associated template and world graphs for the set cover problem where $S = \{1, 2, 3\}$ and we have the subsets $S_1 = \{1\}, S_2 = \{2, 3\}, S_3 = \{2\}$.

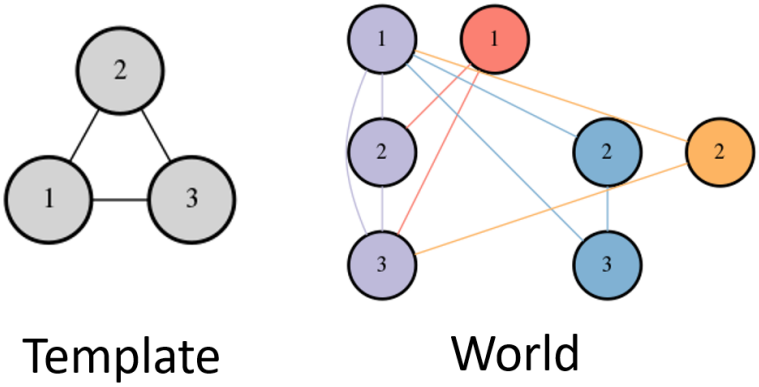

Figure 4.3: The associated template and world for the set cover problem where $S = \{1, 2, 3\}$ and the subsets are $S_1 = \{1\}, S_2 = \{2, 3\}, S_3 = \{2\}$. The world graph vertex (i, j) can be found in the ith row and jth column. The numbers represent the vertex labels. The colors indicate four different subgraph isomorphisms which can be found by picking all vertices of one color and the purple vertices adjacent to them.

To turn this into a template query problem, we set as our ground truth the mapping $g_0 : \mathcal{V}_t \to \mathcal{V}_w$ which is given by $g_0(i) = (i, 0)$ for all $i \in \mathcal{V}_t$. This is easily verified to be a subgraph isomorphism.

With our template and world defined in this manner, we note that there are exactly $m+1$ subgraph isomorphisms which are given by the following proposition:

Proposition 24. *$f : \mathcal{V}_t \to \mathcal{V}_w$ is a subgraph isomorphism from $\mathcal{G}_t$ to $\mathcal{G}_w$ if and only if $f = g$ or there exists a $j \in S$, $f = g_j$ where g_j is given by*

$$g_j(i) = \begin{cases} (i, 0) & j \notin S_i \\ (i, j) & j \in S_i \end{cases}$$

Proof. First we note that as $\mathcal{G}_t$ is complete, all template vertices are adjacent. By construction, each of the g_j are clearly injective and label-preserving. To see that it is edge-preserving, we consider multiple cases. Suppose we have two different template vertices i_1, i_2. If j is not

in S_{i_1} or S_{i_2}, the images will be $(i_1, 0)$ and $(i_2, 0)$ which are adjacent. If j is in exactly one of them, say S_{i_1}, then by definition, (i_1, j) is adjacent to $(i_2, 0)$. If j is in both, (i_1, j) is also adjacent to (i_2, j) by definition.

There are a few cases for mappings that are not g_0 or some g_j. If we have some template vertex i which maps to some (i', j) where $i =\neq i'$, this map will not be label preserving and therefore not a subgraph isomorphism. The remaining cases we assume each vertex i maps to (i, j) for some j. Suppose we have two vertices i_1, i_2 which map to (i_1, j_1) where (i_2, j_2), respectively, where neither of j_1 and j_2 are 0. This is not a subgraph isomorphism as (i_1, j_1) and (i_2, j_2) are not adjacent. The only other case to consider that is not g or some g_j is when there is some i_1 mapping to (i_1, j) with $j \neq 0$ and there is some i_2 with $j \in S_{i_2}$ where i_2 maps to $(i_2, 0)$. This is also not a subgraph isomorphism as (i_1, j) is not adjacent to $(i_2, 0)$ $\square$

Now we observe that if we query template vertex i and find that it maps to $(i, 0)$, we can rule out the subgraph isomorphisms g_j for $j \in S_i$ as these map i to (i, j). To rule out all subgraph isomorphisms except for g_0, we can query vertices $i_1, \ldots, i_k$ whose sets $S_{i_1}, \ldots, S_{i_k}$ constitute a set cover of S. Hence, finding a minimal verification set corresponds to finding a minimal set cover of S.

This construction proves the following theorem:

Theorem 25. *The minimum set cover problem is reducible to solving the minimum solution verification set problem.*

This demonstrates that the decision variant of the minimum solution verification problem (finding a verification set with fewer than N queries) is NP-hard. If we note that any such set is a certificate for the problem, it follows that the problem is in NP, and so we have the following theorem.

Theorem 26. *The decision variant of the solution verification problem is NP-complete.*

4.3.2 Solving the Minimal Solution Verification Set Problem

In spite of the fact that determining if there is a solution verification set of a given size is NP-complete, it may still be possible to solve this problem if the solution space $\mathcal{F} \coloneqq \mathcal{F}(\mathcal{G}_t, \mathcal{G}_w)$ can be computed and is of a manageable size.

Given $\mathcal{F}$ and a solution $f \in \mathcal{F}$, we can reduce finding a minimal verification set to the minimum set cover problem in the following manner. For each subset of template vertices $A \subset \mathcal{V}_t$, we define $\mathcal{F}_A \coloneqq \{g \in \mathcal{F} : \exists t \in A, g(t) \neq f(t)\}$, the set of SIs that we can rule out if we know $f(t)$ for all $t \in A$. A solution verification set is any set $A \subset \mathcal{V}_t$ for which $\mathcal{F}_A = \mathcal{F} \setminus \{f\}$. We then define $|\mathcal{V}_t|$ sets, $S_1 = \mathcal{F}_{t_1}, \ldots, S_{|\mathcal{V}_t|} = \mathcal{F}_{t_{|\mathcal{V}_t|}}$. Our goal is then to determine an index set $I \subset \{1, 2, \ldots, |\mathcal{V}_t|\}$ of minimal cardinality such that $\bigcup_{i \in I} S_i = \mathcal{F} \setminus \{f\}$. This is precisely the minimum set cover problem as defined in Definition 23.

We can solve the minimum set cover using a binary program where we introduce binary variables $z_1, \ldots, z_k \in \{0, 1\}$ where z_i represents whether or not we are using subset S_i in our set cover. We define a matrix $A \in \{0, 1\}^{k \times m}$ with entries a_{ij} for $i = 1, \ldots, k$ and $j = 1, \ldots, m$ where $a_{ij} = 1$ if element j is in subset S_i and 0 otherwise. We can then write the optimization problem we wish to solve with the following formulation:

$$\min_{z} \sum_{i=1}^{k} z_i \tag{4.2}$$

$$\text{s.t.} \sum_{i=1}^{k} a_{ij} z_i \geq 1, \qquad j = 1, \ldots, m \tag{4.3}$$

$$z_i \in \{0, 1\}, \qquad i = 1, \ldots, k \tag{4.4}$$

where the objective that we are minimizing in (4.2) is exactly the count of how many subsets we use. The constraints in (4.3) verify that each element in S is in at least one chosen subset. In practice, the matrix A often has a significant number of duplicate columns which we can safely drop without changing the solution set of the problem. We can then solve this integer program using any standard integer program solver; in this work, we use the state-of-the-art solver Gurobi [Gur18]. Once we have solved this problem, the size of the given solution

verifying set can then be used as a lower bound on the number of queries under any given template query strategy.

4.4 Querying Strategies for Template Vertices

In this section, we will introduce various query strategies for the active learning problem in subgraph matching.

4.4.1 Local Strategies

The first strategies we consider are those which are based entirely on information from the immediate neighborhood of a given vertex. These strategies target vertices which are the least constrained which typically correspond to vertices with low degree or have many candidates. In [GB21], these strategies are shown to be effective to reduce the solution space with a limited number of queries. These techniques are as follows:

- Minimum degree: choose the template vertex with the lowest degree.

$$t = \arg\min_{t \in \mathcal{V}_t} \deg(t) \tag{4.5}$$

- Max candidate: Choose the vertex with the largest number of candidates.

$$t = \arg\max_{t \in \mathcal{V}_t} |C(t)| \tag{4.6}$$

- Max sum of candidates: choose the vertex with the largest sum of the number of candidates for neighboring template vertices and itself.

$$t = \arg\max_{t \in \mathcal{V}_t} \left(|C(t)| + \sum_{t' \in N(t)} |C(t')| \right) \tag{4.7}$$

- Edge entropy: This is the novel technique introduced in [GB21]. Let $t \in \mathcal{V}_t$ and $t' \in N(t)$. Then let $N_{t'}^{tw}$ be the number of candidates for t' if t is matched to $w \in C(t)$.

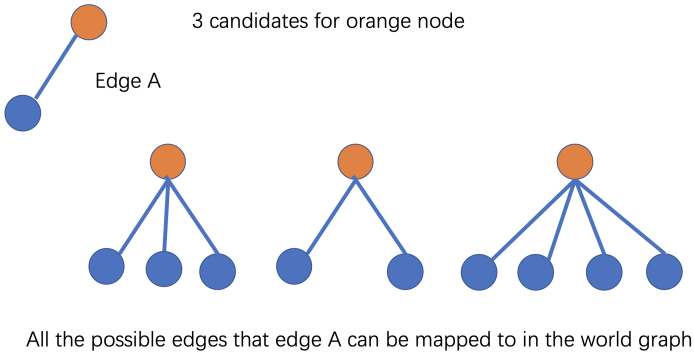

Figure 4.4: Edge entropy example: Edge A may be mapped to nine possible candidate edges in the world. There are three cases that the orange vertex can map to. In the first case, there are three edges connected to the selected world vertex. So the probability in this case is 1/3. Similarly the probability for the remaining edges are 2/9 and 4/9. Hence, the edge entropy of this edge is $-(\frac{1}{3}\log(\frac{1}{3}) + \frac{2}{9}\log(\frac{2}{9}) + \frac{4}{9}\log(\frac{4}{9}))$.

Let $N^t_{t'} = \sum_{w \in C(t)} N^{tw}_{t'}$. Then we can interpret $P^{EE}_{t'}(t = w) := N^{tw}_{t'}/N^t_{t'}$ as an estimate for the probability that t is mapped to w based on the possible mappings of the edge with endpoints t' and t. The edge entropy formula is then given as follows:

$$EE(v) = \sum_{t' \in N(t)} \sum_{w \in C(t)} -P^{EE}_{t'}(t = w) \log P^{EE}_{t'}(t = w) \tag{4.8}$$

Figure 4.4 shows a computation of the edge entropy.

4.4.2 Probabilistic Query Strategies

In this section, we introduce strategies for querying which attempt to estimate the probability with which a template vertex is assigned to a given world vertex. There are a variety of ways of establishing a probability space on the set of subgraph isomorphisms but the simplest involves granting all subgraph isomorphisms for a given template graph and world graph equal probability.

With this established, the probability that a template vertex $t \in \mathcal{V}_t$ and world vertex $w \in \mathcal{V}_w$ are paired together is simply the proportion of subgraph isomorphisms where they are matched:

$$P_{SI}(t = w) = \frac{|\{f \in \mathcal{F}(\mathcal{G}_t, \mathcal{G}_w) : f(t) = w\}|}{|\mathcal{F}(\mathcal{G}_t, \mathcal{G}_w)|}. \tag{4.9}$$

If we know certain template vertices are already assigned to world vertices, we can adjust the above definition to restrict $\mathcal{F}(\mathcal{G}_t, \mathcal{G}_w)$ to those with the known assignments.

Directly computing all subgraph isomorphisms is in many cases computationally intractable owing to the NP-complete nature of simply finding one. Instead of fully enumerating all subgraph isomorphisms, we can instead try to approximate the above quantity based on local structures.

One such structure is the immediate neighborhood. We can attain an approximation for the number of solutions by asking how many mappings there are from the neighbors of a template vertex t to the neighbors of a world vertex w. Put concretely, we can define the set of **local subgraph isomorphisms** (LSI) for any template vertex, world vertex pair (t, w) in the following manner:

$$\begin{aligned} LSI(\mathcal{G}_t, \mathcal{G}_w, t, w, C) = \{f : N(t) \to N(w) : f \text{ injective}, \\ f(t') \in C(t'), \forall t' \in N(t)\}. \end{aligned} \tag{4.10}$$

This is precisely the set of subgraph isomorphisms from the neighborhood of t to the neighborhood of w with the additional requirement that each template vertex has candidates inherited from the original subgraph isomorphism problem.

We can then define an associated probability:

$$P_{LSI}(t = w) = \frac{|LSI(\mathcal{G}_t, \mathcal{G}_w, t, w, C)}{\sum_{w' \in C(t)|} |LSI(\mathcal{G}_t, \mathcal{G}_w, t, w', C)|}. \tag{4.11}$$

Computing the number of LSIs corresponds to counting mappings between $N(t)$ and $N(w)$ and ensuring that each template vertex is assigned a different world vertex. In the constraint programming framework, this is referred to as an *alldifferent* problem. We can reduce this

problem to that of finding the permanent of a 0-1 matrix which lies in the P#-complete complexity class [Val79] suggesting that even this problem may be difficult to solve efficiently. In practice, the problem sizes are often small enough and we can exploit symmetry in the candidate sets to solve these relatively quickly.

In the case that this problem is still too costly to solve, we can compute a crude approximation to the number of LSIs by removing the injectivity requirement in which case we are counting the number of **local edge preserving mappings** (LEPM). This quantity is very easy to count as it is just given by the product of the sizes of each of the candidate sets for the all neighbors:

$$|LEPM(\mathcal{G}_t, \mathcal{G}_w, t, w, C)| = \prod_{t' \in N(t)} |C(t') \cap N(w)|. \tag{4.12}$$

Then our probability is defined in the same manner as before:

$$P_{LEPM}(t = w) = \frac{|LEPM(\mathcal{G}_t, \mathcal{G}_w, t, w, C)}{\sum_{w' \in C(t)|} |LEPM(\mathcal{G}_t, \mathcal{G}_w, t, w, C)}|. \tag{4.13}$$

We can also obtain a rough approximation of the number of SIs by instead considering EPMs for a spanning tree of the template. We denote the number of EPMs between a spanning tree T of template $\mathcal{G}_t$ world $\mathcal{G}_w$ where t is mapped to w by $STEPM(\mathcal{G}_t, \mathcal{G}_w, t, w, T)$. We then have an associated probability computed in the same manner:

$$P_{STEPM}(t = w) = \frac{|STEPM(\mathcal{G}_t, \mathcal{G}_w, t, w, T)|}{\sum_{w' \in C(t)} |STEPM(\mathcal{G}_t, \mathcal{G}_w, t, w', T)|}. \tag{4.14}$$

Once we have computed the associated probabilities, we can devise a variety of strategies for selecting query vertices. There are well-established strategies in active learning (surveyed here [Set12]) and three popular choices for determining queries are minimum confidence sampling, margin sampling, and maximum entropy sampling. We define these as follows:

- *Minimum Confidence Sampling*: Select the template vertex which is least confident about its most likely assignment.

$$t = \arg\min_{t \in V_T} \max_{w \in C(t)} P(t = w). \tag{4.15}$$

- *Margin Sampling*: Select the template vertex whose two most probable assignments are closest together in probability. If $w^* = \arg\max_{w \in C(t)} P(t = w)$, then this is given by

$$t = \arg\min_{t \in V_T} \left(P(t = w^*) - \max_{w \in C(t), w \neq w^*} P(t = w) \right). \tag{4.16}$$

- *Maximum Entropy Sampling*: Select the template vertex which maximizes entropy.

$$t = \arg\max_{t \in V_T} - \sum_{w \in C(t)} P(t = w) \log P(t = w). \tag{4.17}$$

Results from preliminary experiments from the `biochemical_reactions` dataset (as introduced in Table 4.1) are depicted in Figure 4.5. In these experiments, we compare methods by the average amount of queries needed to uniquely determine a solution for a random selection of subgraph matching problems from this dataset. We observe that the maximum entropy and minimum confidence methods have similar performances with the minimum margin method doing worse for each method for computing probability. As these methods have negligible differences on average, we will primarily focus on the maximum entropy approach here.

4.4.3 Symmetry in Active Learning

Another approach for developing a query strategy involves analyzing the symmetry apparent in the subgraph matching problem. Symmetry is well-known for confounding general combinatorial problems by dramatically expanding the solution space. Algorithms which exploit symmetry [BCL16, RW15, HLL13, NYG19] can significantly reduce search time (sometimes by an exponential factor for highly symmetric problems). These algorithms generally identify vertices which are effectively interchangeable in that exchanging them in a subgraph isomorphism will produce another valid isomorphism.

As discussed in Chapter 3, there are various notions of symmetry which can be discussed in the context of the subgraph matching problem. The simplest is **structural equivalence** and two vertices are deemed structurally equivalent when they have the exact same set of

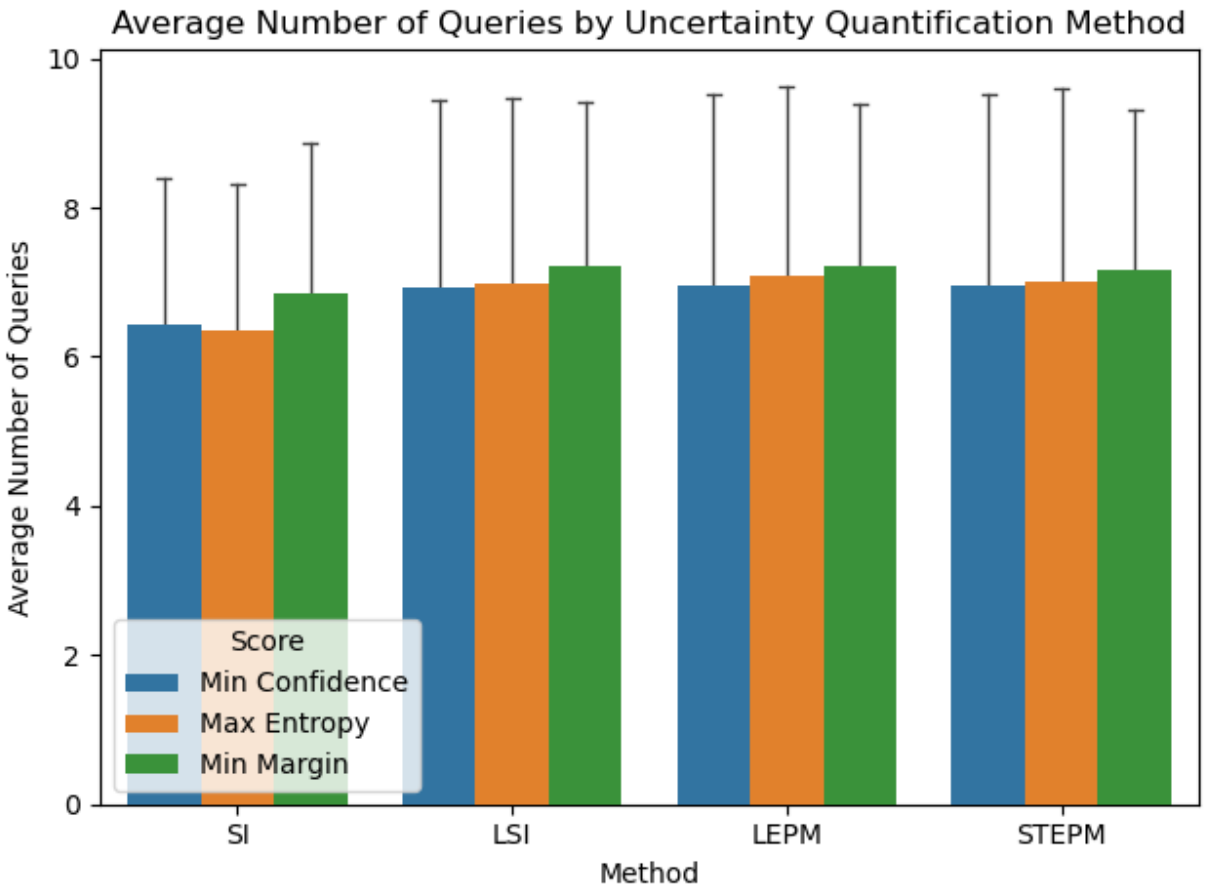

Figure 4.5: Average Number of queries needed to determine a solution to the subgraph matching problem on the `biochemical_reactions` dataset based on the approximation used for the number of SIs and the uncertainty quantification method. Error bars depict the standard deviation in the number of queries.

neighbors (excluding each other). These vertices can be interchanged in any isomorphism and the new mapping will also be an isomorphism. A more complicated notion which includes structural equivalence is **automorphic equivalence**. Two vertices are automorphically equivalent if there is an automorphism $\phi : \mathcal{V} \to \mathcal{V}$ on the graph mapping one vertex to the other. Then given this automorphism ϕ and any subgraph isomorphism f, we can construct a new subgraph isomorphism $f \circ \phi$.

In the context of the active learning problem, symmetry can be an important factor to consider when deciding which vertices to query. If we have a group of M structurally equivalent template vertices, then for any subgraph isomorphism f, we can construct $M! - 1$ additional isomorphisms by simply permuting the images of these vertices. All of these isomorphisms are valid based on the information apparent in the problem. To discern which permutation is the true solution necessarily requires querying at least $M - 1$ of these vertices (the last may be determined by process of elimination).

This discussion naturally leads to the following proposition:

Proposition 27. *Suppose we have a satisfiable subgraph isomorphism problem with template* $\mathcal{G}_t = (\mathcal{V}_t, \mathcal{E}_t)$ *and world* $\mathcal{G}_w = (\mathcal{V}_w, \mathcal{E}_w)$. *Let* $\mathcal{V}_t = \bigcup_{i=1}^{N} S_i$ *partition the template vertices into structural equivalence classes. Then at least* $\sum_{i=1}^{N}(|S_i| - 1)$ *template queries are needed to identify a unique solution.*

Discerning between isomorphisms which are generated by applying an automorphism to the template graph is more challenging. Relevant to this task is the notion of a base of the automorphism group of a template graph. A **base** of an automorphism group is a sequence of vertices $(v_1, v_2, \ldots, v_n)$ such that for any pair of automorphisms $\phi_1 \neq \phi_2$, the two sequences $(\phi_1(v_1), \phi_1(v_2), \ldots, \phi_1(v_n))$ and $(\phi_2(v_1), \phi_2(v_2), \ldots, \phi_2(v_n))$ are distinct. This definition implies that the values of ϕ on $v_1, \ldots, v_n$ uniquely identify ϕ. Hence, to distinguish isomorphisms generated by the automorphism group would involve querying all vertices of a base of the automorphism group.

The problem of generating a base for the automorphism group is difficult but the library `nauty` [MP14] in the process of finding generators for the automorphism group of a general graph will also find a base. Querying vertices which constitute a base for the automorphism group then may be a useful strategy which exploits symmetry.

4.5 Experiments

In this section, we describe a series of experiments performed on a collection of real and synthetic single channel subgraph isomorphism problems. We consider ten different strategies for querying vertices which are listed as follows:

1. MC (Most Candidates): Vertex with the most candidates as in (4.6).
2. MD (Minimum Degree): Vertex with the lowest degree as in (4.5).
3. MNCS (Maximum Neighbor Candidate Sum): Vertex whose neighbors have the most candidates as given by (4.7).
4. ME (Maximum Entropy): Vertex given by (4.17) with P_{SI} defined as in (4.9).
5. MLE (Maximum Local Entropy): Vertex given by (4.17) with P_{LSI} defined as in (4.11).
6. MLE~ (Maximum Local Entropy Approximation): Vertex given by (4.17) with P_{LSH} defined as in (4.13).
7. R (Random): Random vertex.
8. EE (Edge Entropy): Vertex with the highest edge entropy as defined in (4.8).
9. STE (Spanning Tree Entropy): Vertex given by (4.17) with P_{STEPM} as defined in (4.14).
10. O (Optimal): The optimal vertex to query as given by solving (4.2).

Dataset	# Instances	Template					
		# Vertices		# Edges		Density	
		Min	Max	Min	Max	Min	Max
LV-easy	26	10	435	10	2520	0.027	1.000
SI-easy	446	40	777	43	2047	0.006	0.201
biochemical-easy	739	9	68	8	90	0.036	0.417
images	10	5	11	6	13	0.236	0.600
LV-hard	246	10	280	10	1848	0.002	1.000
SI-hard	249	40	518	41	1669	0.006	0.197
biochemical-hard	554	9	184	8	355	0.021	0.423
phase	47	30	30	128	387	0.294	0.890

Table 4.1: Template Graph Statistics from Benchmark Datasets Used in Active Learning Experiments

In the event that multiple vertices tie on one of these criteria, the vertex with minimal index which has not been queried yet will be selected.

For our study, we will again consider the benchmark dataset for subgraph matching problems compiled by Solnon [Sol19]. For our experiments, we only consider problems for which there are at least ten subgraph isomorphisms. From these, we divide our datasets into "easy" and "hard" instances. Easy instances are those for which we can fully enumerate the solution space and therefore can compute the optimal amount of queries as in Section 4.3. Hard instances are the subgraph matching problems where we do not have the full solution space and as such we cannot compute the number of queries using the O or ME methods. We also include in this set, problems for which the MLE does not find terminate within ten minutes. A compilation of basic graph statistics for these datasets is presented in Tables 4.1 and 4.2.

For these problems, we select ten random solutions from the solution space for a given subgraph matching problem. For the easy instances, this is uniformly chosen over all so-

Dataset	World					
	# Vertices		# Edges		Density	
	Min	Max	Min	Max	Min	Max
LV-easy	10	2000	45	2592	0.001	1.000
SI-easy	200	1296	299	4377	0.004	0.098
biochemical-easy	9	386	12	886	0.012	0.423
images	4838	4838	7067	7067	0.001	0.001
LV-hard	42	6671	114	209000	0.001	1.000
SI-hard	200	1296	299	7788	0.004	0.191
biochemical-hard	25	386	44	886	0.012	0.160
phase	150	150	4312	8740	0.386	0.782

Table 4.2: World Graph Statistics from Benchmark Datasets Used in Active Learning Experiments

lutions, and for hard instances, it is randomly chosen from a selection of solutions by the Glasgow solver [MPT20] within one minute. Then for each solution, we proceed using the framework described in Algorithms 4. We count the number of queries made and then record the average number of queries required for a given subgraph isomorphism problem for each querying strategy.

The results for the easy data sets are presented in Figure 4.6. We also consider the average percent gap between a given method's number of queries and the best method for a specific problem. A bar chart of this data is presented in Figure 4.7. We observe that the ME method is nearly optimal for the *biochemical_reactions* and *SI* datasets and requires 20% more guesses on the other datasets. The next best methods are MLE, MLE~, and STE which all approximate the ME method and require between 15 and 40% more guesses across the datasets. Empirically, there does not seem to be much difference in using MLE or the approximation MLE~ with MLE~ even outperforming MLE on certain datasets. The EE and MNCS methods on average perform worse than picking vertices at random. From the

disparity between the optimal and the other methods, there remains significant work to be done to develop computationally tractable methods which can close this gap.

The average number of queries for the hard instances is presented in Figure 4.8 and the average percent additional queries needed over the best method is presented in Figure 4.7. As we do not have the full solution set computed, we cannot consider the O or ME methods, and we also exclude the MLE method due to its long computation time. These problems require significantly more queries, and we observe less variation in the number of queries across methods. On average, we observe that the MLE~ and STE methods perform the best with MC not too far behind. The random method on average requires 10-15% more queries over the best of these methods typically.

As discussed in Section 4.4.3, graph symmetry plays a significant role in subgraph matching and neglecting it may confound some of the methods presented in this section. The biochemical reactions data set is a highly symmetric data set, and as a result of Proposition 27 necessarily require a high number of queries to distinguish a solution. For some query strategies, they systematically avoid the structurally equivalent vertices leading to high query counts. We observe this behavior in Figure 4.10 where the template graph from the biochemical reactions dataset has multiple sets of structurally equivalent vertices. The ME strategy which picks out the structurally equivalent vertices to query first finds the ground truth solution after only seven queries whereas the MLE strategy requires 20 queries. The reason for the significantly higher count comes from how the MLE approach leaves these structurally equivalent vertices for the end as other vertices have higher local entropy values at the stage in which they are queried.

In Figure 4.11, we compare two equivalence-based improvements on the MC strategy to the base MC approach. As discussed in Section 4.4.3, the structural equivalence approach queries all but one member of each structural equivalence class before considering other template vertices. The automorphic equivalence approach first queries template vertices which constitute a base of the automorphism group. As can be seen in the figure, these

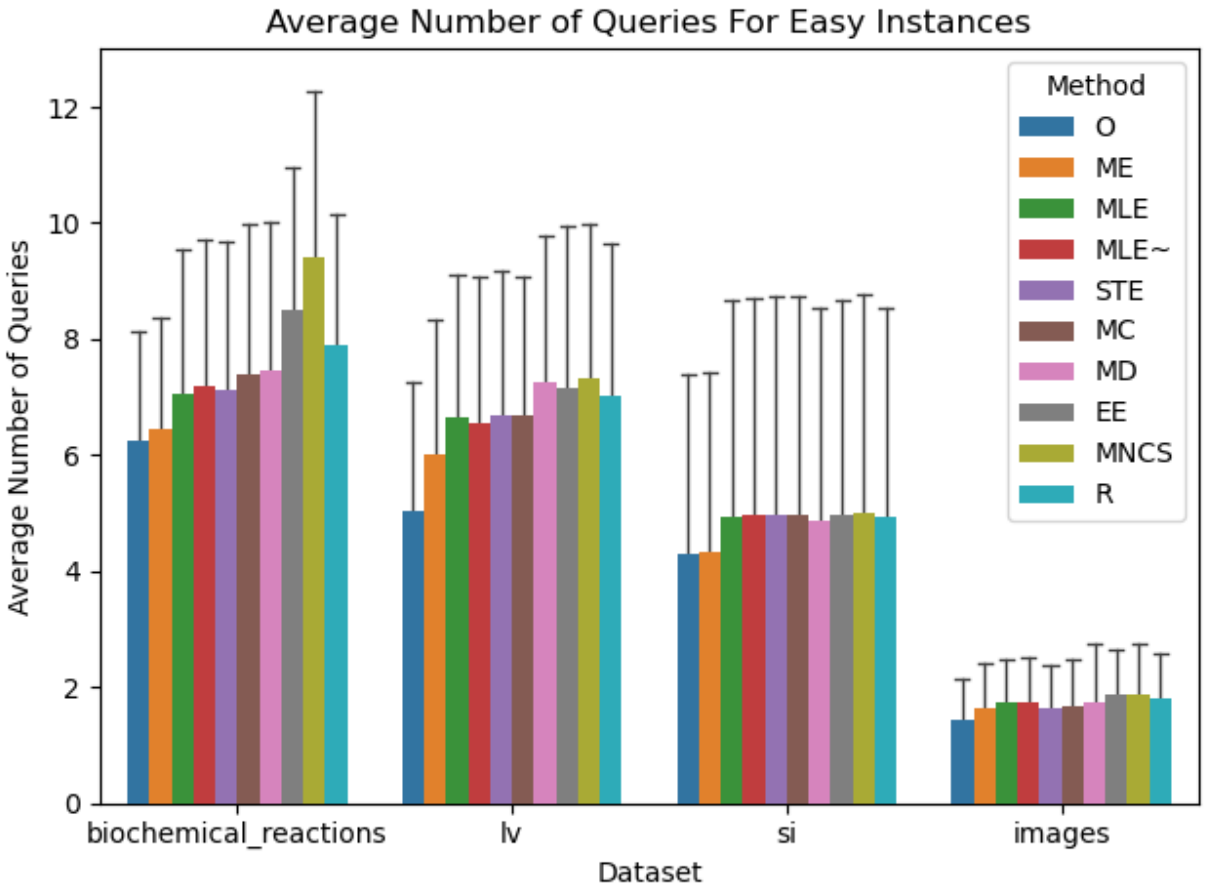

Figure 4.6: Bar chart depicting the average number of queries for the easy problems in Table 4.1. Error bars depict the standard deviation. The methods listed are described in Section 4.5.

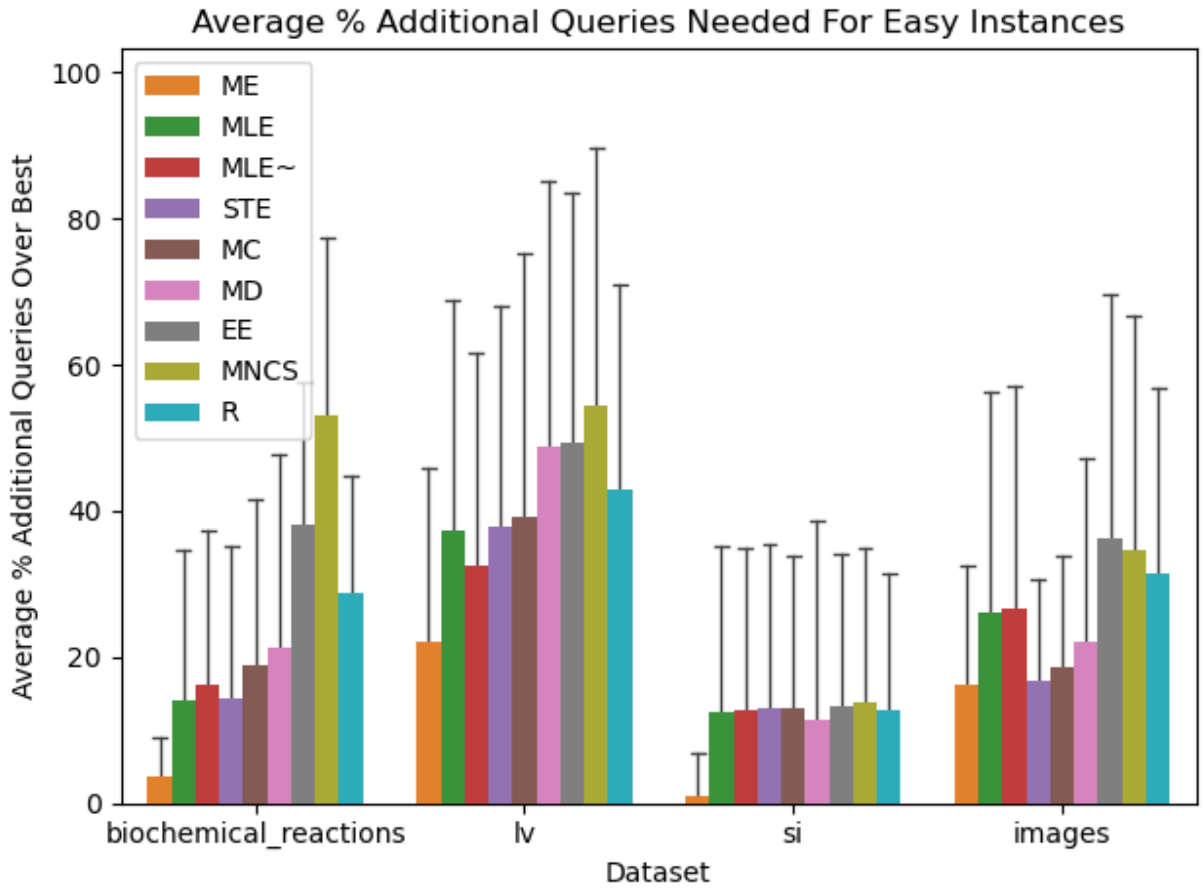

Figure 4.7: Bar chart depicting the average percent additional queries over the optimal amount for the easy problems in Table 4.1. Error bars depict the standard deviation. The methods listed are described in Section 4.5.

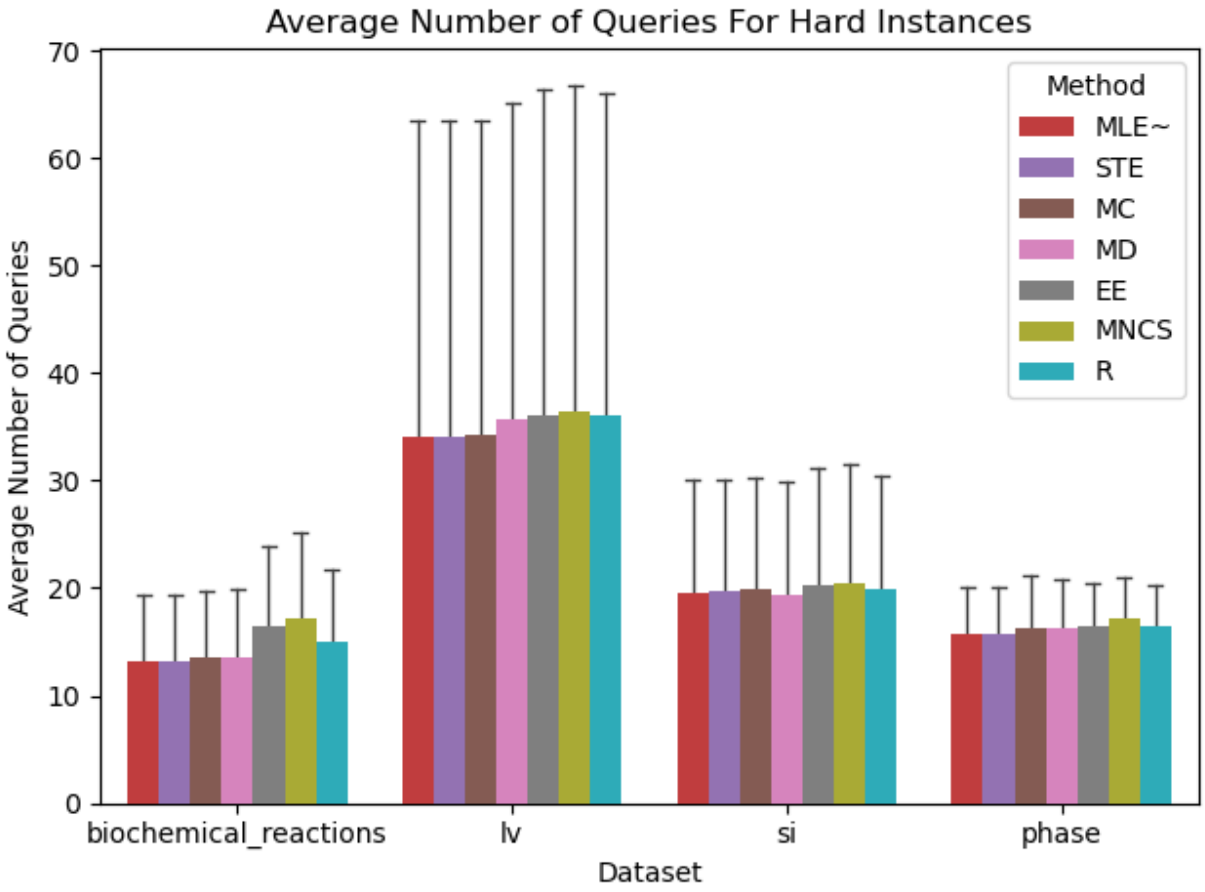

Figure 4.8: Bar chart depicting the average number of queries for the hard problems in Table 4.1. Error bars depict the standard deviation. The methods listed are described in Section 4.5.

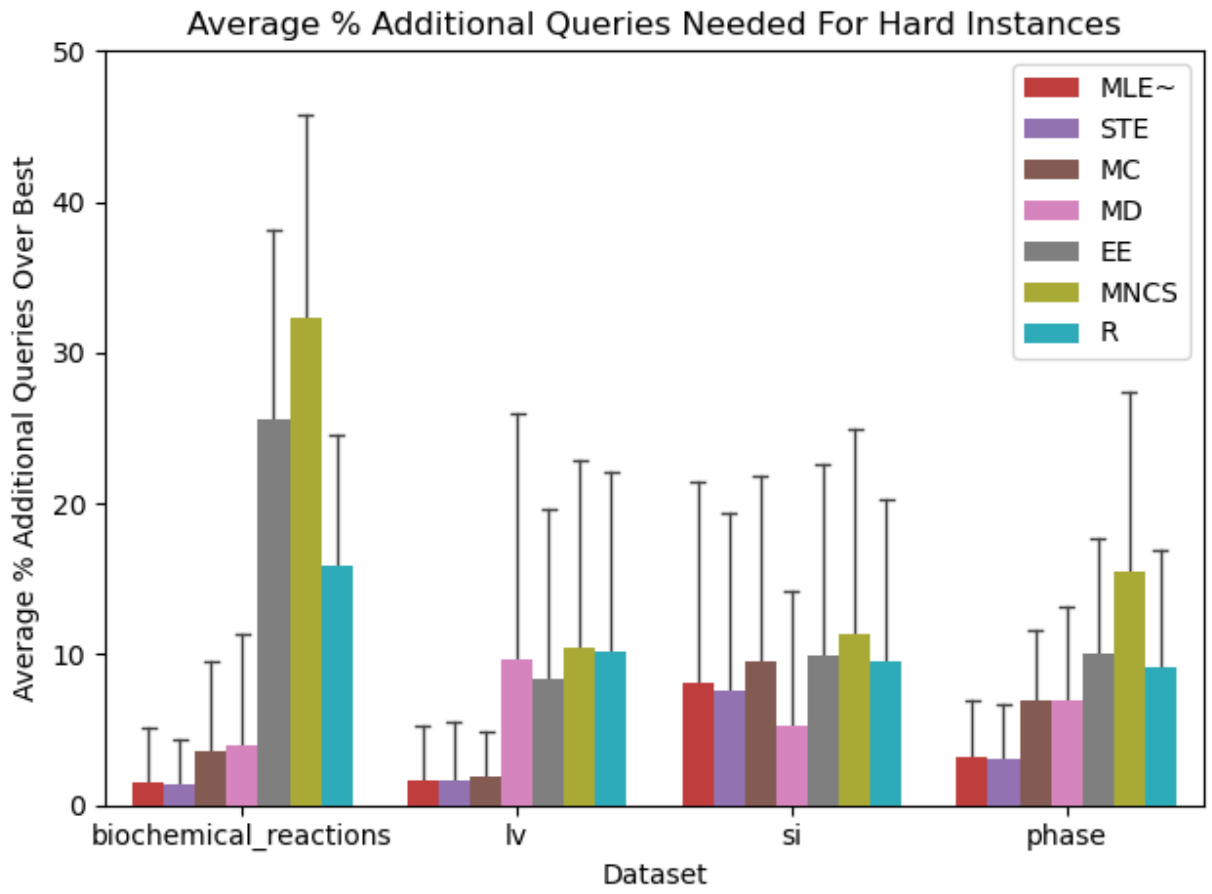

Figure 4.9: Bar chart depicting the average percent additional queries over the best query strategy for the hard problems in Table 4.1. Error bars depict the standard deviation. The methods listed are described in Section 4.5.

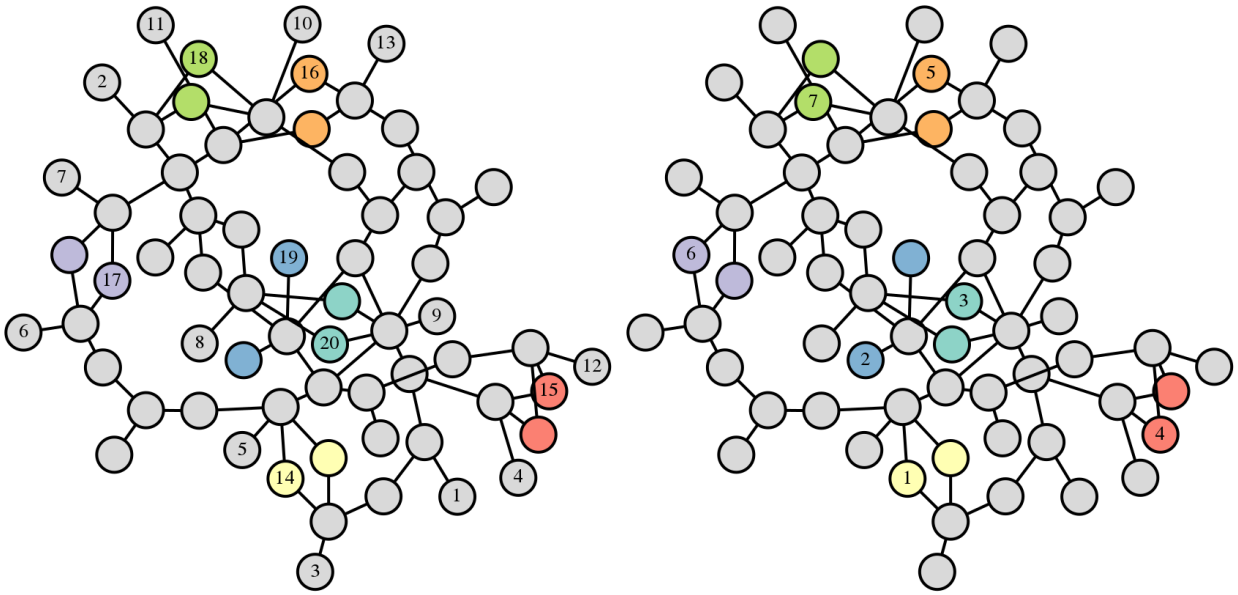

Figure 4.10: Two separate query strategies applied to an example template graph from the biochemical reactions dataset. The numbered vertices indicate the order in which template vertices are queried. On the left, the MLE query strategy is used, and on the right the ME query strategy is used. Non-gray vertices of the same color are structurally equivalent.

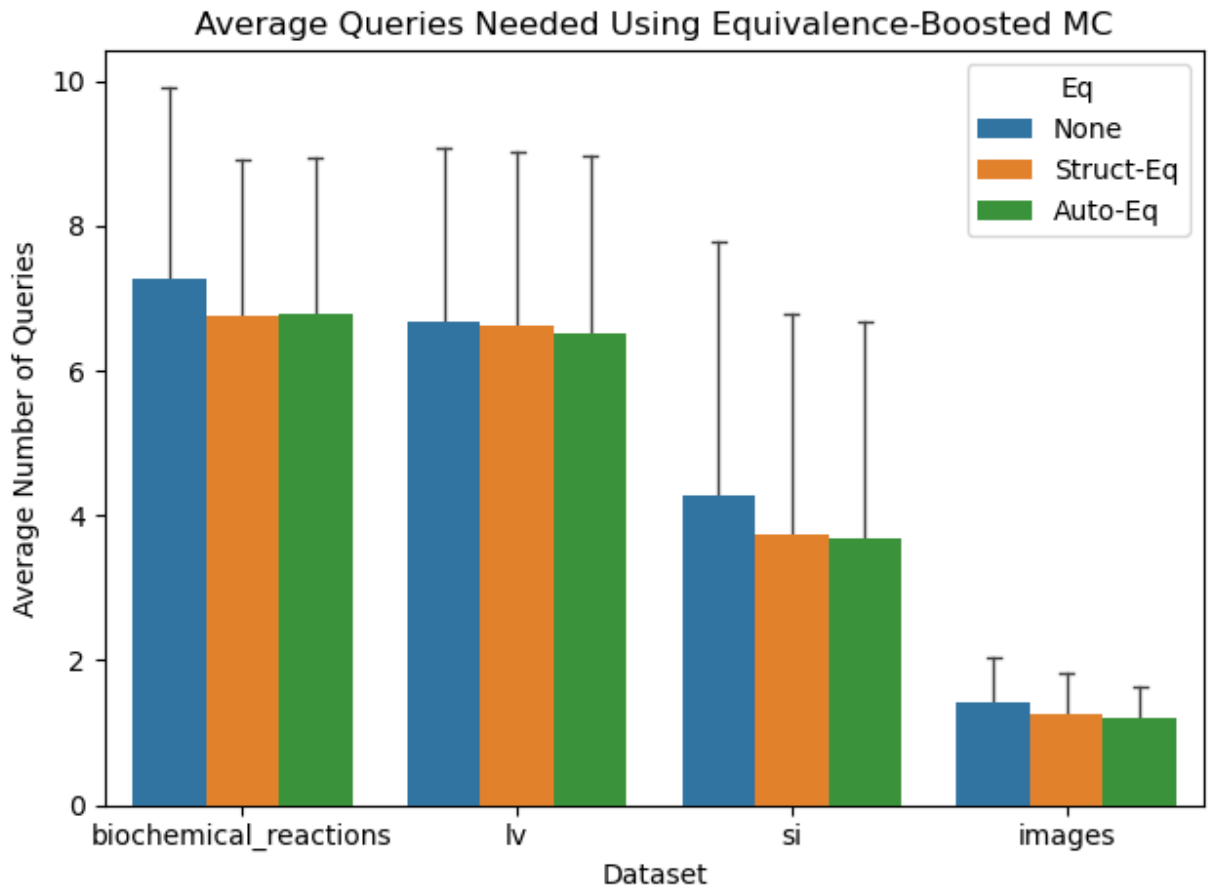

Figure 4.11: Average number of queries made before a solution is found on various single channel datasets when using the MC method with equivalence-informed queries.

equivalence-informed improvements on the base methods can make significant improvements on the average number of queries required to determine a solution. On average, we can save nearly half a query for particular symmetric datasets like the *biochemical_reactions* and *SI* graphs, and on particular examples as in Figure 4.10, using symmetry is essential.

4.6 Conclusion

In this chapter, we present a rigorous mathematical treatment of the active learning query problem for subgraph matching where only one ground truth solution exists. We prove that finding the optimal template vertices to query to be NP-complete even if the solution space and the ground truth are known. We present several strategies for determining template queries, some based on simple graph structure and others which attempt to estimate the probability of matching template and world vertices. We design an inexpensive and fast method to estimate the solution space based on the spanning tree. We assess our results on benchmark data and compare the performance of different strategies on single channel and multichannel network datasets with varying graph size and number of solutions. Our experiments show that for single channel networks, the maximum entropy strategy which requires the calculation of the solution space is nearly optimal and other methods which approximate this approach are close but with much room for improvement. For large multichannel graphs, several methods have similar behavior, with the spanning tree entropy method performing best on average.

There are many open problems which offer great potential directions for future research. One obvious one is the consideration of diverse active learning strategies. For example, the information of query could be related to world graph vertices rather than template graph. Likewise we do not consider combinations of different strategies in sequence but rather just consider iterations of a single strategy. Extensions on the subgraph isomorphism problem is another possible direction for future research. We can introduce noise to the problem and study the active learning for inexact subgraph matching problem [TMY20, KX19, SPP19]. We also only considered sequential queries. Future work could examine batch active learning scenarios - especially in the case of large complex graphs or queries done on the world graph rather than the template.

CHAPTER 5

Modeling design and control problems involving neural network surrogates[1]

5.1 Introduction

In this chapter, we will introduce and analyze the second topic of the thesis: formulating optimization problems involving neural network surrogates. We are interested in solving general optimization problems that include deep neural networks (DNNs) that are used as surrogate models of complex functions (e.g., physical processes [AB19], classification schemes [KSH12, SZ14, HZR16]). In particular, we consider optimization problems of the form

$$\underset{x}{\text{minimize}}\ f(\text{DNN}(x), x) \quad \text{subject to}\, c(\text{DNN}(x), x) \leq 0,\ x \in X, \tag{5.1}$$

where f and c are smooth functions representing the objective function and constraints, respectively; $\text{DNN}(x) \in Y \subset \mathbb{R}^m$ is the output of a DNN at x (which we assume to have been previously trained on suitable data); and x are the optimization variables. $X \subset \mathbb{R}^n$ and $Y \subset \mathbb{R}^m$ are compact sets that may include integer restrictions and represent the input and output spaces, respectively. We consider feedforward neural networks in this paper that are composed of a sequence of multiple layers of neurons. The values of neurons in layer ℓ, x^ℓ are a nonlinear function (the *activation function*) applied to a linear transformation of the values in the prior layer:

$$x^\ell = \sigma(W^{\ell^T} x^{\ell-1} + b^\ell),\ \ell = 1, \ldots, L. \tag{5.2}$$

[1]**This chapter is adapted from [YBL22] and is reproduced with permission from Springer Nature**

We have $x^0 = x$, $x^L = \text{DNN}(x)$, L is the number of layers, W and b are weights determined by a training procedure, and σ is the activation function applied componentwise. Unless otherwise specified, we will take σ to be the ReLU function: $\text{ReLU(x)} = \max\{x, 0\}$.

An example of an optimization problem we wish to solve is to minimize the output of a neural network regressor that predicts the quantity of emissions from automobile engine specifications. In this case X would be the set of existing automobile specifications, the constraints c would be constraints to ensure the engine is realistic, and the objective would be a function of the emissions or engine performance. We will discuss this problem in detail in Section 5.2.1.

To arrive at a tractable form of (5.1), we make the following assumptions.

Assumption 1. *We assume that the following conditions are satisfied for problem* (5.1)*:*

1. *The objective function,* $f : \mathbb{R}^{n+m} \to \mathbb{R}$*, and the constraint functions,* $c : \mathbb{R}^{n+m} \to \mathbb{R}^p$*, are twice continuously differentiable* convex *functions (in both arguments).*

2. *The sets* $X \subset \mathbb{R}^n, Y \subset \mathbb{R}^m$ *are convex and compact.*

The most restrictive assumption is the convexity assumption. We can relax this assumption by leveraging standard global optimization techniques (see, e.g., [Sah96, TS02, Bel20]), at the expense of making the reformulated problem harder to solve. We note, that the convexity assumption is only used in Section 5.4.1 to show that the reformulation with binary variables results in a convex MINLP. The smoothness assumption on f and c can be relaxed to Lipschitz continuity by using subgradients, and the compactness assumption is typically satisfied as long as x are constrained by bounds or the constraint function has compact level sets. Below, we consider deep neural networks with ReLU activation functions that satisfy this assumption.

5.1.1 Outline and Contributions

We start by discussing three applications that employ DNN surrogates within an optimization problem such as (5.1): (a) the minimization of emissions in an engine design problem (as in [AB19]), (b) the generation of optimal adversarial examples to "fool" a given classifier (as in [FJ18]), and (c) the optimization of the pump configuration for oil (as in [GA19]). We develop a warmstart heuristic for the engine design problem that generates good initial guesses from the training data and helps us overcome the challenges of the nonconvex formulation. We also show how to add constraints that restrict the optimization problem to search only near where existing training points can be added.

Next, we demonstrate empirically that simply including a ReLU DNN within an optimization problem can lead to poor convergence results. In particular, we have developed a new nonlinear constraint for JuMP [DHL17] that allows us to directly include DNNs within an optimization model specified in JuMP. We then give a compact characterization of stationarity of the embedded formulation of our optimization problem.

Next, we consider alternative formulations and show that DNNs that use purely ReLU activation functions can be formulated as mixed-integer sets, building on [FJ18, AHT19]. We can then formulate (5.1) *equivalently* as a convex mixed-integer nonlinear program (MINLP), which we refer to as the mixed-integer program (MIP) formulation. We also introduce a new formulation that results in a nonconvex nonlinear program (NLP) with complementarity constraints. We refer to this as the mathematical program with complementary constraints (MPCC) formulation. We prove this formulation has stationarity conditions equivalent to the embedded formulation. Our reformulations involve a lifting into a higher-dimensional space in which the MINLP problem is convex.

We show empirically for each of our applications that for moderately sized machine learning (ML) models the resulting programs can be solved by using state-of-the-art MINLP and NLP solvers, making optimization problems with ML models tractable in practice. We

demonstrate that using the MIP formulation, we can find provably optimal solutions for small problems. Using the embedded and MPCC formulation, we show that we can address significantly larger networks at the cost of guaranteeing only locally optimal solutions, and we showcase scenarios where the MPCC formulation outperforms the embedded formulation in terms of both optimal value found and consistency in convergence to a solution.

Throughout this chapter we assume that the deep neural network has been trained and is fixed for the optimization, and we do not consider the question of updating the neural network weights during the optimization loop. One limitation of our approach is that we use standard MINLP solvers to tackle the reformulated MINLP, which limits the size of the neural network, $\text{DNN}(x)$, that can be used in the optimization.

5.1.2 Related Work

Prior approaches to optimization over neural networks using MIP formulations include [FJ18, CNR17, DJS18, KGD18, SR20, TXT17]. These approaches primarily model the piecewise ReLU constraint using standard big-M modeling tricks. They generally use the same basic formulation, but each augments the solve by adding methods to tighten the big-M constraints [FJ18, TXT17, GA19], decomposing the problem into smaller problems [KGD18], or adding local search routines [DJS18]. Anderson et al. [AHT19] provide an in-depth overview of how to strengthen these models to an ideal formulation with exponentially many constraints, as well as a method to separate in linear time. Lombardi et al. [LMB17] present a MIP formulation as well as constraint programming and local search approaches to embedding neural networks in optimization problems. Bergman et al. [BHB21] have produced a modeling framework which incorporates embedded neural networks and logistic regression models as MIPs in optimization problems.

Some other approaches to these problems using methods from MINLP have been tried. Cheon [Che21] solves an inverse problem over ReLU constraints using a method inspired outer-approximation approaches, but without global optimality guarantees. Katz et al.

[KBD17] use an approach from satisfiability modulo theory to address an optimization problem over neural networks. Scheweidtmann and Mitsos [SM19] use a MINLP approach involving McCormick relaxations.

One major focus of these optimization problems is on testing the resilience of neural networks against adversarial attack [CW17]. This involves either maximizing a notion of resilience [CNR17] or finding minimal perturbations needed to misclassify an image [FJ18]. Some work has also been done in using optimization to visualize features corresponding to neurons [FJ18] and for surrogate optimization in the context of optimizing the production of a set of oil wells [GA19]. Other applications involve the use of neural networks as surrogates in the context of policy design for reinforcement learning [RCA19, DAT20]. A recent paper by Papalexopoulos et al. [PTA21] discusses the use of ReLU neural network surrogates for the purposes of black-box optimization.

5.2 Modeling Optimization Applications Involving Neural Network Surrogates

In this section we describe three optimization models that make use of neural-network surrogates, and we discuss some of the challenges that arise.

5.2.1 Optimal Design of Combustion Engine

Automobile engine operation is typically modeled by using highly intensive physics-based simulation code, which even on powerful computers can take hours just to model even a short drive. Hence, modeling the evaluation of this simulation code by using a neural network surrogate model can produce significant time savings, at the cost of producing slightly less accurate results. A study examining this approach is documented in [AB19].

Given a trained neural network, we could formulate several optimization problems an-

swering questions related to the operation of an engine on a given commute. The problem we consider is that of minimizing emissions over the course of a given drive. The resulting solution will then be both the engine type and the drive style (e.g., RPM at all times in the drive) that produces the most environmentally efficient commute. The following section demonstrates how such a problem can be formulated.

5.2.1.1 Simple Engine Design and Control Problem

Suppose we have a trained DNN that predicts engine behavior based on engine specifications and driving parameters, given in Table 5.1. The DNN is trained by using 64 trips each split into 1,500 observations (25 minutes observed at second intervals). This training data was provided by the authors of [AB19] and was generated in a comprehensive physics simulation. 250,000 separate cycles were produced and a representative set of 64 cycles ranging over the entire parameter space were sampled using Latin hypercube sampling. The authors of [AB19] report that with a configuration of 6 hidden layers of 16 nodes each, the resulting mean absolute percent test error was about 1%.

Input Parameter	Output Parameter
fuel injection (g) /s	nitrogen oxide, NO /s
engine RPM /s	carbon monoxide, CO /s
compression ratio	torque /s

Table 5.1: Input/output parameters of engine DNN model. Time-dependent parameters are shown with time per second (/s).

The problem we propose to solve is the optimal design and control of an engine for a given 25-minute trip. We will use a prescribed torque profile as a surrogate for the trip characteristics and minimize a weighted sum of NO and CO output. We have the following design variables that are input to the DNN: fuel injection, f; engine RPM, r; and compression ratio, c. Note that f and r are characteristics that change over time whereas c is an engine

parameter that is fixed for the full drive. We also have variables NO and CO that represent nitrogen oxide and carbon monoxide emissions produced by the engine, as well as a variable torque that indicates engine torque. Each of these quantities is predicted by the neural network for each time interval of the drive.

Formally, we state the following optimal design and control problem over the time horizon $t = 1, \ldots, T$ (where T is the number of time intervals):

$$\begin{aligned}
\underset{f,r,c,\mathrm{CO},\mathrm{NO},\mathrm{Torque}}{\text{minimize}} \quad & \sum_{t=1}^{T} (\mathrm{NO}_t + \lambda \mathrm{CO}_t) \Delta t && \text{min. emissions} \\
\text{subject to} \quad & \mathrm{Torque}_t \geq \mathrm{PrescribedTorque}_t, \ \forall t = 1, \ldots, T && \text{trip profile} \\
& (\mathrm{NO}_t, \mathrm{CO}_t, \mathrm{Torque}_t) = \mathrm{DNN}\,(f_t, r_t, c) && \text{DNN constraints} \\
& f_t \in [f_{\min}, f_{\max}], \ r_t \in [r_{\min}, r_{\max}], \ c \in [c_{\min}, c_{\max}] && \text{bounds on controls.}
\end{aligned} \tag{5.3}$$

We are using 1,500 time intervals (corresponding to $T = 1500$ seconds in a 25-minute drive). As written, we will have $2T + 1$ continuous control variables in addition to the $3T$ output variables predicted by the DNN for a total of $5T + 1$ continuous variables in this formulation.

We envision this problem to require only a single solve before a prospective drive. Modifications could be made to this formulation to adapt it to an online setting where control parameters are updated during a drive, but the methods developed in this paper address the former case.

Our model has a separate DNN evaluation for each time interval t, meaning each evaluation engenders different neural network activations. Because of the presence of the design variable c, which is independent of t, this problem does not decompose into individual time steps. One may construct a bilevel optimization problem wherein on the upper level we decide c and other engine-level variables of interest and on the lower level we determine the drive-specific variables that change over time by solving T separate optimization problems of much smaller size. This approach will not be addressed in this paper.

5.2.1.2 Convex Hull Constraints

In addition to the constraints that encode the evaluation of a neural network, constraints must be added to ensure that the solver does not extrapolate significantly from the training data. Without these constraints, the optimization routine may find that the optimal solution resides in an area for which the neural network has not learned the behavior of the modeled function, leading to a solution that, while optimal, does not reflect the true function behavior and may be nonsensical. In fact, in an earlier implementation without the convex hull constraint, we observed that the optimal design was obtained for an engine that produced negative emissions. We rectify this error by introducing constraints that constrain the input data to our model to be within the convex hull of the input training data.

In our experiments we examined how the optimization models operated with simple box constraints that bounded the input by the extremal values of the training data, as well as with the convex hull constraints. Figure 5.1 shows the location of fuel mass and RPM for solutions computed for a sample neural network trained on our data set compared with the actual training data. The corresponding solutions are displayed in Figure 5.2.

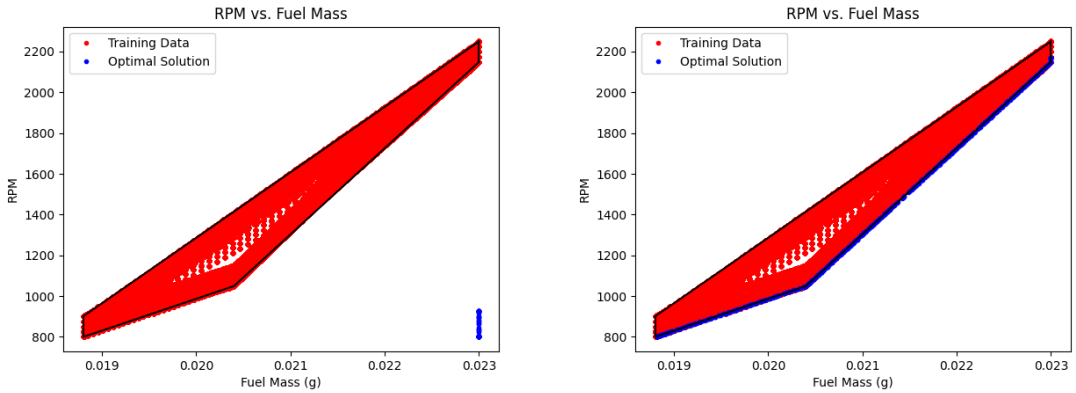

Figure 5.1: Location of training data and optimal solution for an engine design problem on a neural network with 1 hidden layer of 16 nodes. On the left the input is constrained by box constraints and on the right by convex hull constraints.

In our setting, this approach works well, because the convex hull of our data admits a small number of faces, owing to the shape of the training data. In general, the number of faces of the convex hull of n points can grow very quickly as n grows. In the worst case, where the convex hull is a cyclic polytope [Gal63], the number of facets can grow as fast as $n^{\lfloor \frac{d}{2} \rfloor}$. In these scenarios, alternative approaches would be required perhaps involving the collection of a representative sample.

Without the convex hull constraints, the optimal solution of neural network evaluation appears to wander off to an area for which no training data exists. This situation immediately creates problems in the computed solutions, which are obviously nonsensical because they involve negative emissions. This suggests that our neural network does not generalize well to data that has not yet been seen by the network. On the other hand, constraining the input data to be within the convex hull has the effect of producing solutions that look like the training data.

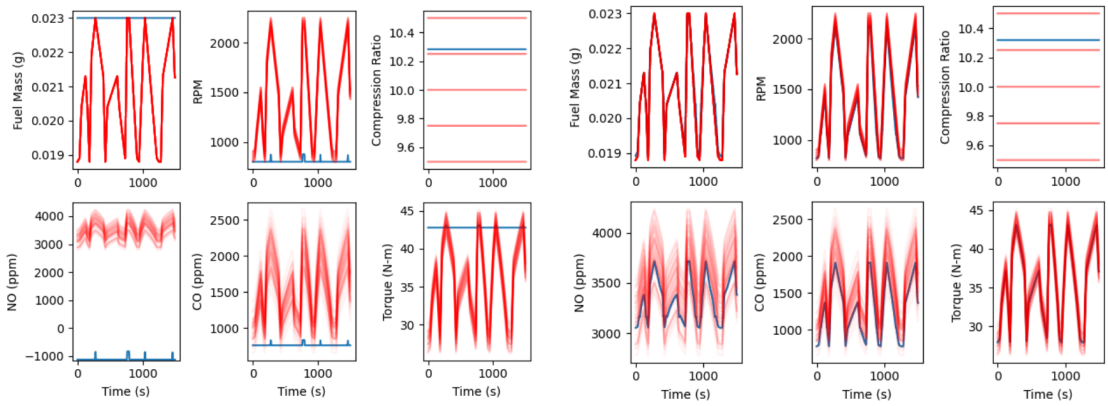

Figure 5.2: Optimal configuration for each time step when using box constraints (left) and using convex hull constraints (right). The red lines indicate the drive cycles used to train the neural network, and the blue lines indicate the optimal solution.

Unfortunately, these constraints have the effect of restricting possible solutions to those that are close to the training data, when the function being modeled may have a minimum that appears far from the training data. A happy medium between restricting the data to

the bounds of the training data and unrestricted optimization might be adjusting the given convex hull bounds by a small quantity ϵ. In this way, we could allow for a small amount of exploration in a region for which we do not have training data, but do not extrapolate too much from our known data. We do not consider this modification in this paper.

To find a global optimum of the original function would necessarily involve an alternating approach wherein the surrogate model was optimized and then this solution was queried against the modeled function for new data to be added to the surrogate. This type of approach is addressed in [QHS05] and a similar approach is implemented in [PTA21].

If we fix the compression ratio, we can plot the contours of the objective function as a function of fuel mass and RPM. Figure 5.3 shows the contours of the objective function for a 5-layer DNN. We observe that this objective function is highly nonconvex in the reduced space of the original variables but it is convex in the lifted MINLP space according to Proposition 32. We also observe that the convex hull constraint fulfills a second function by acting as a mild convexifier of the problem by restricting the variables to a small sliver of the feasible set.

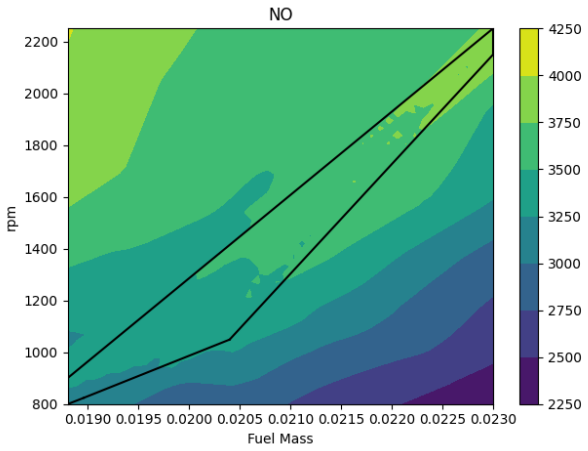

Figure 5.3: Contour plot for the NO output of a 5-layer DNN.

5.2.1.3 Warmstarting the Problem

Seeding a MILP or NLP solver with a feasible solution of high quality can significantly reduce the solve time. For MILP solvers, a feasible solution with sufficiently low value can be used to prune subproblems with higher objective value in a branch-and-bound tree search. For NLP solvers, beginning with a feasible solution avoids the need to search for a feasible solution, and the choice of a good solution may lead to quicker convergence to better locally optimal solutions. R3.2Being in the neighborhood of good solutions can lead to fewer iterations needed to reach the good solutions and we observe this behavior in the experiments in Section 5.5.1.

For the optimal engine design problem, we can use our collection of training data to find parameterizations of the engine that exceed the desired torque value but also have a low amount of emissions produced. We do not directly work with the output data from the training set but rather with the output from the neural network applied to the input data since this is what is constrained.

To produce a high-quality solution is then a matter of fixing the parameters that must remain constant for the entire drive (i.e., the compression ratio) and then, from the training data with these fixed parameters, choosing the remaining controls so as to minimize emissions while still exceeding the desired torque for each time step. The remaining variable values correspond to activation of each neuron in the neural network when applied to the input data and hence must be set to exactly those values. This procedure is summarized in Algorithm 5.

5.2.2 Adversarial Attack Generation

Our next problem involves the resilience of neural networks to incorrect classifications. Deep neural networks have the immensely useful property of being able to uniformly approximate any function in a certain general class, but at the cost of being fairly opaque in terms of how the underlying machinery works. This opacity may mask unpredictable behavior that

Algorithm 5 Algorithm for finding a warmstart solution to (5.3)

Input DNN, compression ratio c_0, training data $\mathcal{X}$, torque profile TorqueProfile_t, T

Select subset of training data $\tilde{\mathcal{X}}$ with compression ratio c_0

Let $\tilde{\mathcal{Y}} = \text{DNN}(\tilde{\mathcal{X}})$

for $t = 1, \ldots, T$ **do**

Let $\text{NO}, \text{CO}, \text{Torque} = \tilde{\mathcal{Y}}$

Let $\mathcal{T}$ be the set of times t where $\text{Torque}_t \geq \text{TorqueProfile}_t$

Let $\tau = \text{argmin}_{t \in \mathcal{T}}(NO_t + \lambda CO_t)$

Let $x_t = \tilde{\mathcal{X}}_\tau$

Set remaining variable values from activation of DNN when applied to x_t for all t.

makes these neural networks susceptible to attacks that disrupt the correct classifications observed on training and testing data. Szegedy et al. [SZS14] first demonstrated that almost imperceptible perturbations of image data may lead to misclassifications, in essence demonstrating key instabilities in neural networks. This work has led to a number of papers [CW17, CNR17, FJ18] developing algorithms that produce estimates of the resilience of neural networks, estimating how close an incorrectly classified image can be to a correctly classified one.

The central problem of interest as introduced by [SZS14] is described as follows. Given a classifier DNN, an image $x \in [0,1]^m$, and a desired classification label l, we have the following problem:

$$\begin{aligned} \underset{z}{\text{minimize}} \quad & \|x - z\| \\ \text{subject to} \quad & \text{DNN}(z) = l \\ & z \in [0,1]^m, \end{aligned} \tag{5.4}$$

where $\|\cdot\|$ is a suitable norm. Essentially, we ask for an image of minimal distance to image x that is classified with a different label. We remark that this particular problem is distinct from the other problems discussed in this paper as the actual object of interest is the neural network itself, and susceptibility of this surrogate to malicious data manipulation.

We include this as yet another example of an optimization problem involving an embedded neural network as well as a study into how we may analyze the robustness of a neural network surrogate using our three different methods.

We add additional constraints (see, e.g., [FJ18]) to ensure that this is a very confident classification. Specifically, we replace the constraint $\mathrm{DNN}(z) = l$ with a constraint asking that the activation for label l is some factor larger than the activation for each of the other labels; in other words, if y_L is the final layer, we have $y_{L,l} \geq \alpha y_{L,i}$ for $i \neq l$. Often the final layer is given by applying a softmax function, namely,

$$y_{L,i} = \sigma(z_1, \ldots, z_n)_i = \exp(z_i) / \sum_{j=1}^{n} \exp(z_j), \tag{5.5}$$

where the produced values $y_{L,i}$ effectively represent probabilities that the image is a given label i. In this case we can represent this constraint in terms of z_i as

$$\sigma(z)_l / \sigma(z)_i \geq \alpha \Leftrightarrow \exp(z_l - z_i) \geq \alpha \Leftrightarrow z_l \geq z_i + \log(\alpha),$$

which is a linear constraint. Then if we write $y_L = \mathrm{DNN}(z)$ as the final layer prior to the softmax layer, we will have the following problem:

$$\begin{aligned} \underset{z}{\text{minimize}} \quad & \|x - z\| \\ \text{subject to} \quad & y_L = \mathrm{DNN}(z) \\ & y_{L,l} \geq y_{L,i} + \log(\alpha), \qquad i \neq l \\ & z \in [0,1]^m. \end{aligned} \tag{5.6}$$

We observe that the formulation in (5.6) is easily extendable and that constraints on the allowable perturbations can augment the formulation. For example, we may restrict the magnitude of the perturbation to any given pixel by restricting $|x_i - z_i| \leq \epsilon$ for all i. Alternatively, we may be interested in continuous perturbations and therefore restrict $|(x_i - z_i) - (x_j - z_j)| < \epsilon$ for all pixels j adjacent to pixel i. Constraints of this sort have been used to extend adversarial attack optimization problems in [FJ18].

Set	Description
N	Set of nodes in the network.
N^w	Subset of well nodes in network.
N^m	Subset of manifold nodes in the network.
N^s	Subset of separator nodes in the network.
E	Set of edges in the network.
E^d	Subset of discrete edges that can be turned on or off.
E^r	Subset of riser edges.
E_i^{in}	Subset of edges entering node i.
E_i^{out}	Subset of edges leaving node i.
C	Oil, gas, and water.

Table 5.2: Sets used in oil well optimization problem

5.2.3 Surrogate Modeling of Oil Well Networks

The next example of an optimization problem with an embedded neural network involves the operation of an offshore oil platform and is taken from [GA19]. The full optimization problem is reproduced in Figure (5.4), and the associated sets are given in Table 5.2.

In this problem we have a network comprising three sets of nodes: the wells N^w that are the sources of oil, water, and gas; the manifolds N^m that are connected to wells and mix incoming flows of oil, water, and gas; and the separators N^s that are the sinks of all the flow. Each well is connected to each manifold by a pipeline in E^d that may be turned on and off. Each manifold is then connected to a unique separator by a riser in E^r. An example network with 8 wells, 2 manifolds, and 2 separators taken from [GA19] is depicted in Figure 5.5.

Our goal in this problem is then to optimize the flow rate of oil to each of the sinks. Various physical and logical constraints are used to encode the operation of this flow network. (5.7b) ensures that the flow into each node matches the flow out of each node. (5.7e) and

$$\underset{y,q,p,\Delta p}{\text{maximize}} \sum_{e \in E^r} q_{e,\text{oil}} \tag{5.7a}$$

$$\text{subject to} \sum_{e \in E_i^{\text{in}}} q_{e,c} = \sum_{e \in E_i^{\text{out}}} q_{e,c}, \ \forall c \in C, i \in N^m \tag{5.7b}$$

$$p_j = \text{DNN}_e(q_{e,\text{oil}}, q_{e,\text{gas}}, q_{e,\text{wat}}, p_i), \qquad \forall e = (i,j) \in E^r \tag{5.7c}$$

$$\Delta p_e = p_i - p_j, \qquad \forall e = (i,j) \in E^r \tag{5.7d}$$

$$-M_e(1 - y_e) \le p_i - p_j - \Delta p_e, \qquad \forall e = (i,j) \in E^d \tag{5.7e}$$

$$p_i - p_j - \Delta p_e \le M_e(1 - y_e), \qquad \forall e = (i,j) \in E^d \tag{5.7f}$$

$$\sum_{e \in E_i^{out}} y_e \le 1, \qquad \forall i \in N^w \tag{5.7g}$$

$$y_e q_{e,c}^L \le q_{e,c} \le y_e q_{e,c}^U \qquad \forall c \in C, e \in E^d \tag{5.7h}$$

$$p_i^L \le p_i \le p_i^U, \qquad \forall i \in N \tag{5.7i}$$

$$\sum_{e \in E_i^{\text{out}}} q_{e,\text{oil}} = \text{DNN}_i(p_i), \qquad \forall i \in N^w \tag{5.7j}$$

$$\sum_{e \in E_i^{\text{out}}} q_{e,\text{gas}} = c_{e,\text{gor}} \sum_{e \in E_i^{\text{out}}} q_{e,\text{oil}}, \qquad \forall i \in N^w \tag{5.7k}$$

$$\sum_{e \in E_i^{\text{out}}} q_{e,\text{wat}} = c_{e,\text{wor}} \sum_{e \in E_i^{\text{out}}} q_{e,\text{oil}}, \qquad \forall i \in N^w \tag{5.7l}$$

$$p_i = p_i^s = \text{const.}, \qquad \forall i \in N^w \tag{5.7m}$$

$$y_e \in \{0,1\}, \forall e \in E^d \tag{5.7n}$$

Figure 5.4: Oil well optimization problem with deep neural network surrogate functions.

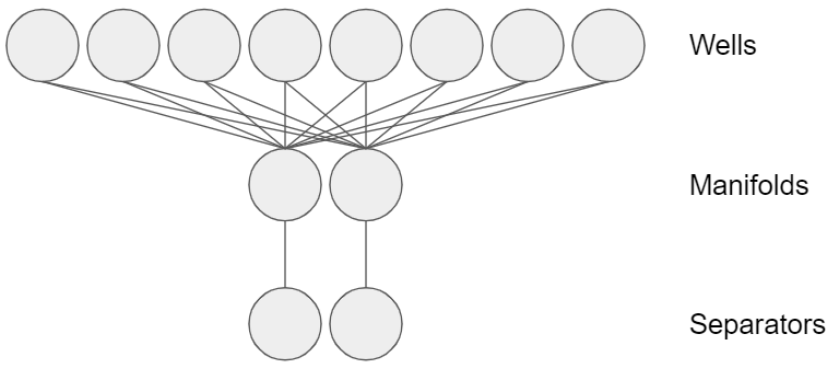

Figure 5.5: Example oil well network with 8 wells, 2 manifolds, and 2 separators, from [GA19].

(5.7f) ensure that if a pipeline is open, then the difference in pressure between the two nodes is actually represented by Δp_e. (5.7g) ensures that each well routes its flow only to a single manifold. (5.7h) bounds the flow rate of each material when pipelines are active and forces the rate to zero when inactive, whereas (5.7i) bounds the pressure at each node. (5.7k) and (5.7l) establish the expected ratio of flow rates of each material.

The neural networks DNN_e and DNN_i appear in the problem in (5.7c) and (5.7j) and represent nonlinear functions that predict the separator pressure and outgoing oil flow rate based on incoming flow rate and pressure, respectively. There is one neural network DNN_e for each riser edge e and one neural network DNN_i for each well i. Each is trained separately; and for our particular configuration of the network, we have 2 risers and 8 wells for a total of 10 separate neural networks that are encoded into our problem.

Among these three problems, with the exception of the DNN constraints, the engine design problem has a linear objective with linear constraints on real variables, the adversarial attack problem has a quadratic objective with linear constraints on real variables, and the oil well problem has a linear objective with linear constraints on a combination of real and integer variables. The introduction of neural network constraints to the problem introduces the complexity of nonlinear (and nondifferentiable) constraints. Our approach to handle

them is either using a directly embedded nonlinear function, as we consider in Section 5.3, or reformulate using integer variables or complementarity constraints which we consider in Section 5.4.

5.3 Embedded Neural Network Formulation

Given that many deep learning libraries (e.g.,, TensorFlow [ABC16] and PyTorch [PGC17]) have well-developed built-in automatic differentiation capabilities, we naturally want to see whether we can directly embed the evaluation of the neural network into an NLP:

$$\underset{x}{\text{minimize}}\ f(\text{DNN}(x), x) \quad \text{subject to}\, c(\text{DNN}(x), x) = 0,\ x \in X. \tag{5.8}$$

Unlike the formulations in Section 5.4, this formulation has the advantage of not requiring auxiliary variables for each of the internal nodes of the neural network. That is, this formulation should scale significantly better as the number of nodes in the neural network increases.

The modeling package JuMP [DHL17] in the programming language Julia is a library that establishes a general framework for representing generic optimization problems and interfacing with solvers. Of particular use in this problem is its ability to handle nonlinear functions with user-provided gradients.

The current release of JuMP supports only univariate user-defined nonlinear functions, but we can represent vector-valued functions by listing each output component separately. A single nonlinear constraint representing $y_i = \text{DNN}(x)_i$ can be written as

```
register(model, DNN_i, n, DNN_i, DNN_i_prime)))
@NLconstraint(model, y[i] == DNN_i(x...)))),
```

where `model` is the JuMP optimization model, `n` is the dimension of the input, and `DNN_i` and `DNN_i_prime` represent function evaluations of the neural network's ith output neuron and its gradient that can be provided by TensorFlow.

For networks with many outputs, explicitly listing these commands can quickly become unwieldy, but we can enumerate these constraints using macros. The following code demonstrates how one can encode $y = M(x)$, where $x \in \mathbb{R}^n$ and $y \in \mathbb{R}^m$, and `f` and `f_p` are evaluation of the neural network and the Jacobian, respectively.

```
macro DNNConstraints_grad(model, x, y, n, m, f, f_p)
  ex = Expr(:block)
  for i = 1:m
  dnn = gensym("DNN")
  push!(ex.args, :($dnn_s(z...) = $f(z...)[$i]))
    p = gensym("DNN_prime")
    push!(ex.args,:($prime(g, z...) = begin g .= $f_p(z...)[$i,:] end))
    push!(ex.args,:(register($model, $(QuoteNode(dnn_s)), $n, $dnn, $p)))
    push!(ex.args,:(@NLconstraint($model, $y[$i] == $dnn($x...))))
  end
  ex
end
```

This macro can then be called to add constraints on a neural network that takes in values from $\mathbb{R}^{20}$ and outputs values in $\mathbb{R}^{10}$ using the following command,

```
@DNNConstraints_grad(model, x, y, 20, 10, DNN, DNN_prime),
```

where `DNN` and `DNN_prime` are now vector- and matrix-valued functions that return the output vector and Jacobian for neural network evaluation, respectively.

With this macro, we now have the ability to directly embed evaluation of a neural network and its derivatives within a mathematical program. This allows us to treat neural network evaluation as simply another function that appears in functions and constraints so that we can use any NLP solver for our optimization problems.

5.3.1 Convergence Behavior

We have observed in our experiments that state-of-the-art solvers express real difficulties with convergence when applied to this formulation. We believe that the reason has to do with the choice of activation function, ReLU(a) = $\max(a, 0)$. This activation function results in nonconvex constraints and objective in (5.1); moreover, the function is nonsmooth whenever $a = 0$.

The nonsmoothness of the ReLU function has generally been considered to be of little concern since, in practice, neural networks involving ReLU neurons have found great success especially in problems of classification [HZR16, SZ14, KSH12] so that they have become an industry standard in deep learning. Part of their success is attributed to their ease in implementation as well as their tendency to produce sparse activation patterns and avoid the vanishing gradient problem in training [GBB11]. Furthermore, some theoretical results have affirmed some convergence guarantees under mild conditions [LY17, DZP18] and [GVS14] have demonstrated qualitatively that the training problem often experiences few of the issues common in nonconvex and nonsmooth optimization.

These results pertain primarily to the optimization problems solved in the training process. We have observed relatively little work in the literature regarding constrained optimization problems involving trained networks. Our preliminary experiments suggest that, in this context, the expectation of good behavior may be unfounded. Initial solves of Problem (5.3) using the state-of-the-art NLP solver Ipopt [WB06] experienced serious issues with convergence. Figure 5.6 illustrates the objective value and primal and dual infeasibility for one example solve of the problem. Qualitatively, we observe that instead of terminating, the solver bounces about among objective values of about the same magnitude. The reason for the lack of convergence can be explained by the fact that one measure of convergence, the dual infeasibility, remains high throughout the duration of the solve. This type of behavior makes this formulation difficult for general use because there are not clear conditions

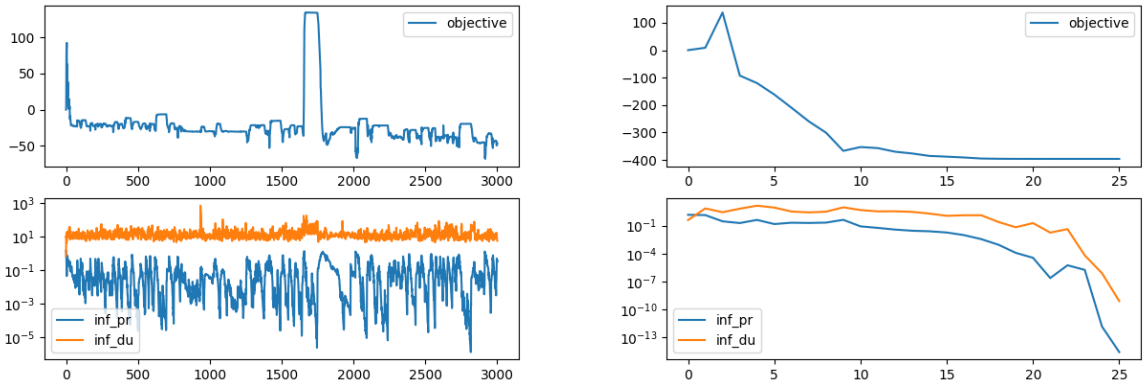

Figure 5.6: Objective value, primal infeasibility, and dual infeasibility plotted against iteration number for a sample ReLU network (left) and a sample swish network (right) over an Ipopt solve of the Engine Design Problem in (5.3).

in general for checking whether the current solution at hand is "good." We suspect that these convergence issues can arise in part due to the nondifferentiability of the ReLU neurons. Classical optimization algorithms such as quasi-Newton algorithms [Pow69] often have basked in the assumption of differentiability, and convergence results typically depend on them.

We observe that in the 480 solves of Problem (5.3) that we perform in Section 5.5.1, only 17.5% were reported by Ipopt to have converged to a locally optimal solution, 57.1% failed to converge as a result of exceeding the iteration limit of 3000, and the remaining 25.4% failed for other reasons, mainly an inability to call a restoration phase at a feasible point. Of this latter group, the impact of nondifferentiability on the solver is most apparent. In Figure 5.7, we plot the minimal distance to a discontinuity and the distance to the particular hyperplane discontinuity that the solve converges to for one failed solve of Problem (5.3). In this instance, even though the iterates do converge to a solution, Ipopt does not declare a solution found due to a failure to satisfy first order conditions. Furthermore, the solve exhibits the zigzagging behavior of repeatedly jumping between the two regimes on each

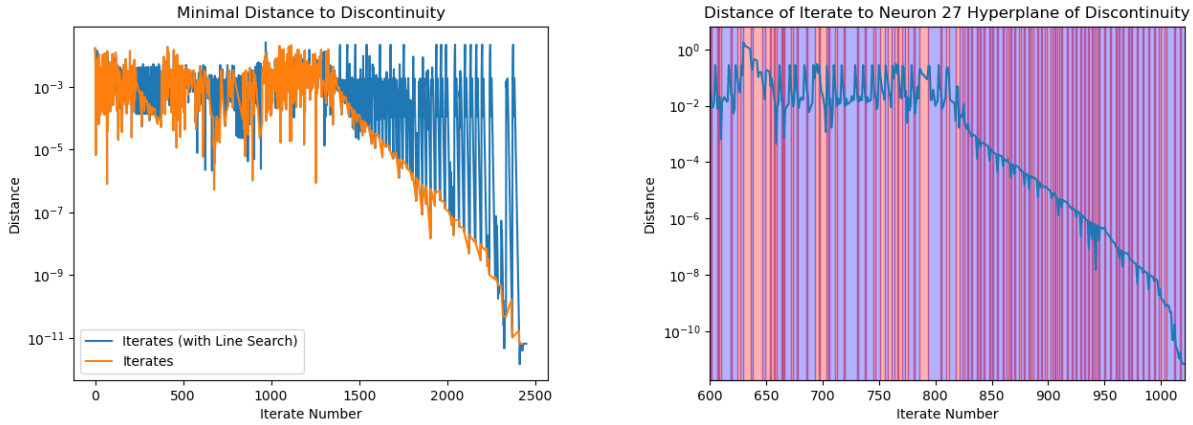

Figure 5.7: The minimal distance to a discontinuity (left) and distance to one neuron's hyperplane (right) for iterates from an Ipopt solve of one instance of Problem (5.3). On the right image, the highlighted sections indicate when the iterate is one side of the hyperplane and when it is on the other.

side of the hyperplane before eventually failing.

We note that there have been contexts where gradient-based optimization methods have been applied with success in the context of constrained optimization involving trained networks, most notably for adversarial attack generation. However, these methods typically address a modified optimization problem instead of directly treating the neural network as a surrogate function in a nonlinear program. This is the case for the original method presented in [SZS14] which instead minimizes a weighted combination of perturbation norm and neural network loss and [CW17] which addresses a variety of alternatives to the constraint that the neural network misclassifies the perturbed image. Furthermore, often these methods do not address issues of convergence to a local optimum instead settling for performing a fixed number of iterations or finding a sufficiently good solution. This is the case for "one-step methods" such as the "Fast Gradient Sign Method" [GSS15] and its generalization to multiple steps, the "Basic Iterative Method" [KGB18], each of which take a fixed number of steps based on a user-specified parameter in the direction of the sign of gradient of the loss.

An alternative type of activation function that has seen some success in classifier networks [RZL17] is the swish function, which is defined as follows:

$$\text{swish}(a) = \frac{a}{1 + e^{-\beta a}}. \tag{5.9}$$

β is a hyperparameter that may be learned but is typically set to 1. Note that for $\beta = 0$ this is simply a linear function and that for $\beta \to \infty$ this approaches the ReLU function. With the activation function in (5.9), all functions in (5.1) are twice continuously differentiable under Assumption 1.

To ensure that failure to converge when using the ReLU network is not due to choice of NLP solver, we perform the same solve of Problem (5.3) using instead swish neurons. We plot the objective and infeasibilities in Figure 5.6. Instead of failing to terminate after 3,000 iterations, Ipopt converges successfully after only 25 iterations; and both primal infeasibility and dual infeasibility converge rapidly to zero. As a differentiable network, the swish formulation does not express the convergence issues that the nondifferentiable ReLU networks appear to exhibit.

These initial results confirm the observation in [BP20, Fig. 1] that showed that ReLU networks can be *nondifferentiable almost everywhere* for certain types of networks. This observation makes the use of ReLU networks questionable as embedded constraints within optimization solvers that rely on differentiable problem functions. Hence, in Section 5.4 we develop alternative formulations that have better numerical properties.

5.3.2 Stationarity in the Embedded Formulation

In this section we will give a condition for the stationarity of the embedded formulation as well as prove a specific form for the Jacobian of a ReLU neural network that holds under mild genericity conditions. To handle the points of nondifferentiability in the embedded formulation, we consider generalized Jacobians first introduced in [CLJ96], which can be

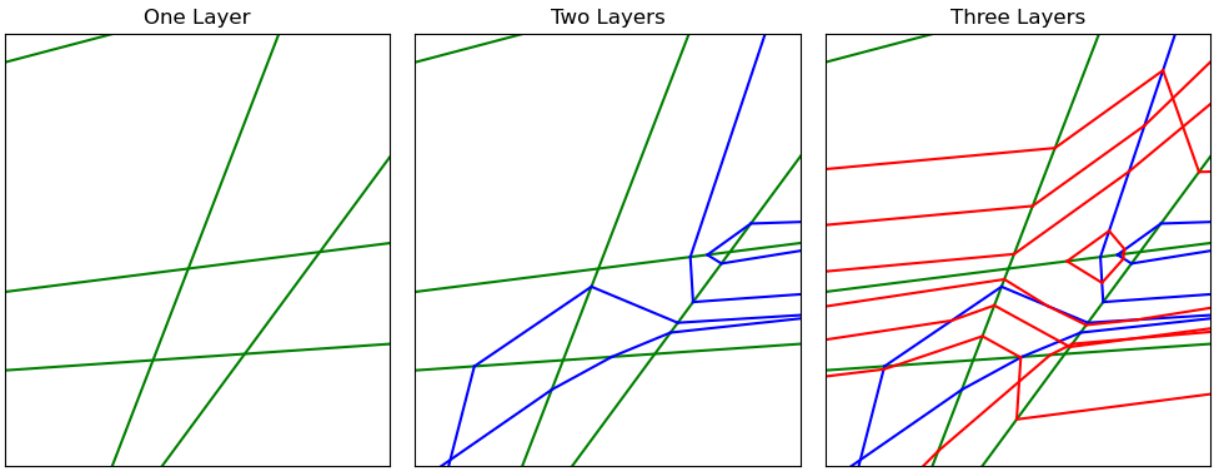

Figure 5.8: Regions of a randomly instantiated neural network with two inputs followed by 1, 2, and 3 layers of 5 ReLU neurons each.

defined for all x in the domain of a function g as follows:

$$\partial g(x) = \text{conv}\{\lim_k \nabla g(x_k) : x_k \to x, x_k \notin S\}, \tag{5.10}$$

where S is the set of points of nondifferentiability of g and $\text{conv}(A)$ is the convex hull of the set A. Since ReLU networks are piecewise linear functions by construction, the generalized Jacobian will be given by a convex hull of the Jacobians of linear functions on finitely many regions in the neighborhood of a given point. Stationarity of g at a point x is then equivalent to having $0 \in \partial g(x)$.

We elaborate on what we mean by region in the context of feedforward ReLU networks. As these neural networks are created by composing sequences of ReLU activation functions with affine maps, we can specify the region of the function by deciding which ReLU neurons will be active; each selection corresponds to a different affine function. Given a set of active neurons N (we will refer to these as *activation patterns*), we will define the associated region $R(N)$ to be the set of points x such that the value at a neuron in N prior to activation is positive and this value for any other neuron is negative. The value prior to activation being zero corresponds to being on the boundary of another region. $R(N)$ will be empty if

no point x engenders the activation pattern N. The regions will tile the input space, and adjacent regions are separated by the 0-level sets of a particular neuron.

Depictions of the regions of neural networks with 1, 2, and 3 layers can be seen in Figure 5.8. We observe that the regions of a single-layer network arise from arrangements of hyperplanes but that as more layers are added, the regions become more complex as they are subdivided. The total amount of nonempty regions of both shallow and deep neural networks has been studied in detail in [MPC14] and [PMB14]. However, their definition of region as maximally connected linear regions of the piecewise function slightly differs from ours because we distinguish adjacent regions that have the different activation patterns but ultimately get mapped to the same linear map.

Before we can discuss stationarity of the embedded formulation, we will need to introduce some notation. We will identify any neuron by its layer index ℓ and the index i within the layer. Given a subset of neurons in the neural network $N = \{(\ell_1, i_1), \ldots, (\ell_n, i_n)\}$, we define F_N to be the affine form given when the set of neurons in N are active and all other neurons are inactive. Put precisely, given the affine transform in layer ℓ, $A^\ell x = W^{\ell^T} x + b^\ell$, if neuron (ℓ, i) is inactive, we replace column w_i and component b_i with zeros. We define our new affine form as

$$\hat{A}^\ell_N x = \hat{W}^{\ell^T} x + \hat{b}^\ell,\ \hat{W}^\ell = W\mathrm{diag}(\kappa^\ell),\ \hat{b} = \mathrm{diag}(\kappa^\ell)b.$$

where κ^ℓ is a vector with κ^ℓ_i equal to 1 if neuron $(\ell, i) \in N$, and zero otherwise. Then $F_N(x) = \hat{A}^L_N \hat{A}^{L-1}_N \cdots \hat{A}^1_N x$.

We can partition the neurons of layer ℓ into three sets given x: strictly inactive neurons $I^{\ell,-}(x)$, nonstrictly inactive neurons $I^{\ell,0}(x)$, and active neurons $I^{\ell,+}(x)$. We define these to be the indices i of the neurons in layer ℓ for which $A^\ell_i \hat{A}^{\ell-1} \cdots \hat{A}^1 x$ is negative, zero, and positive, respectively. The sets $I^+(x), I^0(x), I^-(x)$ are then defined to be all layer-neuron pairs (ℓ, i), where $i \in I^{\ell,+}(x), I^{\ell,0}(x)$, or $I^{\ell,-}(x)$, respectively. With these definitions we observe that for any x, $\mathrm{DNN}(x) = F_{I^+(x)}(x)$.

Note that F_N is now smooth as a composition of affine functions, and we can compute the Jacobian J_{F_N} simply by the chain rule:

$$J_{F_N} = \hat{W}^1 \hat{W}^2 \cdots \hat{W}^L. \tag{5.11}$$

In light of (5.10), the generalized gradient at x can then be written as the convex hull of these Jacobians associated with the activation pattern for each region in the neighborhood of x. Any such activation pattern must necessarily include $I^+(x)$ and exclude $I^-(x)$ but may only include a subset of $I^0(x)$. If there is a region corresponding to each set of the form $I^+(x) \cup N$, where $N \subset I^0(x)$, we can write the generalized gradient in a nice form given by the following proposition.

Proposition 28. *Suppose at a point x that the region $R(I^+(x) \cup M)$ is nonempty for each choice of $M \subset I^0(x)$. Let S be the set of all matrices of the form*

$$W^1 diag(\kappa^1) \cdots W^L diag(\kappa^L),$$

where

$$\begin{cases} \kappa_i^\ell = 1 & (\ell, i) \in I^+(x), \\ \kappa_i^\ell \in [0,1] & (\ell, i) \in I^0(x), \\ \kappa_i^\ell = 0 & (\ell, i) \in I^-(x). \end{cases}$$

Then the generalized gradient of $DNN(x)$ at x, $\partial DNN(x) = S$.

Proof. First, we remark that any Jacobian J_{F_N}, where $N = I^+(x) \cup M$, is clearly in S since we can take $\kappa_i^\ell = 1$ for all $(\ell, i) \in M$ and take it equal to 0 for $(\ell, i) \in I^0(x) \setminus M$, and the two expressions agree. Furthermore any convex combination of the J_{F_N} should be contained in S since S is a convex set. Hence $\partial \mathrm{DNN}(x) \subset S$.

Now given a matrix $V = W^1 \mathrm{diag}(\kappa^1) \cdots W^L \mathrm{diag}(\kappa^L)$ in S, we show that it is in $\partial \mathrm{DNN}(x)$. We enumerate $I^0(x) = \{(\ell_1, i_1), \ldots, (\ell_n, i_n)\}$. Consider first any alteration V_1 of V by fixing $\kappa_{i_1}^{\ell_1}$ and replacing $\kappa_{i_j}^{\ell_j}$ by $\hat{\kappa}^{\ell_j} \in \{0, 1\}$ for $j > 1$. Note that V_1 is a convex combination of the

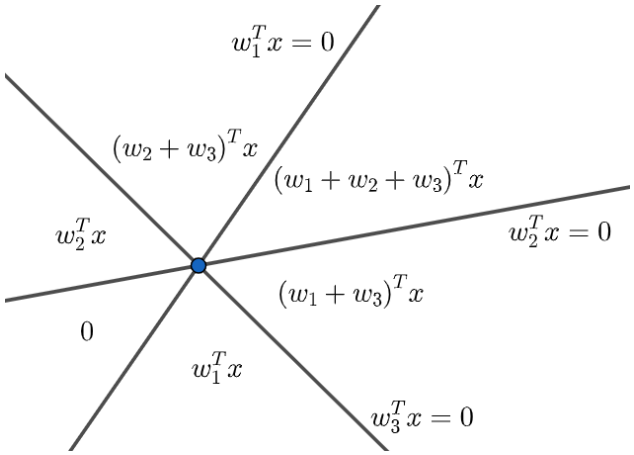

Figure 5.9: Regions of a zero-bias, single-layer network with two input variables and three output variables.

two matrices given by replacing $\kappa_{i_1}^{\ell_1}$ with 1 and by replacing it with 0. Both these matrices are in $\partial\text{DNN}(x)$ since this set includes all Jacobians associated with any activation pattern on $I^0(x)$. Hence we must have that $V_1 \in \partial\text{DNN}(x)$. If we then consider fixing the first two components and replacing the remaining such components to produce V_2, in the same fashion we can find two matrices of the form V_1 of which V_2 is a convex combination and hence $V_2 \in \partial\text{DNN}(x)$. Repeating until we have fixed all components, we will have that $V \in \partial\text{DNN}(x)$. □

Unless the region is nonempty for each of these activation patterns, the generalized Jacobian will not necessarily take this form. As a counterexample, consider a zero-bias two-layer network with one input variable and $W^1 = (1, 1)$, $W^2 = (1, -1)^T$. This corresponds to having $\text{DNN}(x) = \text{ReLU}(x) - \text{ReLU}(x) = 0$, and so the generalized gradient is $\{0\}$ everywhere, whereas the formulation given above would introduce many more possible gradients.

For a slightly more interesting example, consider a one-layer network with two input variables, no bias, and three output variables, namely, $\text{DNN}(x) = \text{ReLU}(W^T x) = \text{ReLU}(w_1^T x) + \text{ReLU}(w_2^T x) + \text{ReLU}(w_3^T x)$. The input domain $\mathbb{R}^2$ is then divided into regions by the lines $w_i^T x = 0$ for $i = 1, 2, 3$ each intersecting at the origin, as can be seen in Figure 5.9. Observe

that in the neighborhood of the origin, there are only six neighboring regions and two regions are missing, one where the output would be $(w_1+w_2)^T x$ and another where the output is w_3^T, corresponding to activation patterns where only the first two neurons are active and only the third neuron is active, respectively. In this scenario the generalized gradient is similarly not as expansive as we desire.

Requiring each region $R(I^+(x) \cup N)$ to be nonempty seems to be an onerous requirement, especially given there are $2^{|I^0(x)|}$ such regions. However, it turns out that under some mild genericity conditions, we can expect this to be true. These conditions arise from the theory of hyperplane arrangements. Suppose we have a collection of hyperplanes $H_1, \dots, H_n \subset \mathbb{R}^d$, where H_i is given by $\{x \in \mathbb{R}^d : a_i^T x = b_i\}$ for some vectors $a_i \in \mathbb{R}^d$ and constants $b_i \in \mathbb{R}$. We say the collection is in *general position* if the intersection of m hyperplanes has dimension $d - m$ if $m \leq d$ and is empty if $m > d$. If the intersection of all hyperplanes is nonempty, it is called a *central arrangement*. In particular, if $d = 2$, the hyperplanes are lines in $\mathbb{R}^2$, and they are in general position if no three lines intersect at a point and no two lines are parallel. A well-known result from Zaslavsky [Zas75] is that if we have m hyperplanes in general position, the number of regions they divide $\mathbb{R}^d$ into is given by $\sum_{i=0}^{d} \binom{m}{i}$. Note that if $m \leq d$, this quantity is 2^m, and we can identify each region by choosing whether $a_i^T x < b$ or $a_i^T x > b$ for all i.

To ensure that the necessary regions are not empty, we will require a specific set of hyperplanes to be in general position. Before we describe the hyperplanes, we will prove a useful lemma.

Lemma 29. *Suppose a central arrangement of hyperplanes $H_1, \dots, H_n$, where $H_i = \{x \in \mathbb{R}^n : a_i^T x = b_i\}$. Then the set $H_1, \dots, H_{n-1}, H_n^A$, where $H_n^A = \{x : (\sum_{i \in A} w_i a_i + a_n)^T x = \sum_{i \in A} w_i b_i + b_n\}$ for any choice of constants $w_i \in \mathbb{R}$, is also in general position for any $A \subset \{1, \dots, n-1\}$.*

Proof. Because the hyperplanes intersect, we can change coordinates so that each hyperplane intersects the origin; thus, without loss of generality, we can write $b_i = 0$ for all i. In this set-

ting, hyperplanes being in general position is equivalent to the normal vectors being linearly independent. Obviously if $a_1, \ldots, a_n$ are linearly independent, $a_1, \ldots, a_{n-1}, a_n + \sum_{i \in A} w_i a_i$ are linearly independent. □

Using this lemma, in the following proposition we can now show exactly which hyperplanes need to be in general position.

Proposition 30. *Suppose that the set of hyperplanes* $\{H_{\ell,i} = \{x : A_i^\ell \hat{A}_{I^+(x)}^{\ell-1} \cdots \hat{A}_{I^+(x)}^1 x = 0\} : \ell = 1, \ldots, L, i \in I^{0,\ell}(x)\}$ *is in general position. Then for any choice of* $N \subset I^0(x)$*, the set of hyperplanes* $\{H_{\ell,i} = \{x : A_i^\ell \hat{A}_M^{\ell-1} \cdots \hat{A}_M^1 x = 0\} : \ell = 1, \ldots, L, i \in I^{0,\ell}(x)\}$ *is also in general position with* $M = I^+(x) \cup N$*.*

Proof. Observe for any hyperplane $A_i^\ell \hat{A}_{I^+(x)}^{\ell-1} \cdots \hat{A}_{I^+(x)}^1 x = 0$ and any j in $1, \ldots, \ell-1$, we consider the associated affine form $A_i^\ell \hat{A}_{I^+(x)}^{\ell-1} \cdots \hat{A}_{I^+(x)}^{j+1} (\hat{A}_{I^+(x)}^j + \hat{A}_N^j) \hat{A}_{I^+(x)}^{j-1} \cdots \hat{A}_{I^+(x)}^1 x = 0$. Let $b^T x + c$ denote the affine form associated with $A_i^\ell \hat{A}_{I^+(x)}^{\ell-1} \cdots \hat{A}_{I^+(x)}^{j+1}$, and for each neuron i in layer j let $d_i^T x + e_i$ denote the affine form associated with $\hat{A}_i^j \hat{A}_{I^+(x)}^{j-1} \cdots \hat{A}_{I^+(x)}^1 x$. Then we can write

$$A_i^\ell \hat{A}_{I^+(x)}^{\ell-1} \cdots \hat{A}_{I^+(x)}^{j+1} (\hat{A}_{I^+(x)}^j + \hat{A}_N^j) \hat{A}_{I^+(x)}^{j-1} \cdots \hat{A}_{I^+(x)}^1 x = \sum_{i \in I^{+,j}(x) \cup N_j} b_i (d_i^T x + e_i) + c_i,$$

and by assumption the hyperplane defined by $H = \{x : \sum_{i \in I^{+,j}(x)} b_i (d_i^T x + e_i) + c_i = 0\}$ is in general position with the hyperplanes defined by $H_i = \{x : d_i^T x + e_i = 0\}$ for $i \in N_j$. The new hyperplane is created by adding to the normal vector of H a linear combination of the normal vectors of each H_i, and so it remains in general position with the H_i. Hence, we have replaced one $I^+(x)$ with M. Repeating in this fashion, we can replace each $I^+(x)$ in each affine form with M and still remain in general position. □

Now that for any choice of $N \subset I^0(x)$ the associated hyperplanes will be in general position, we can be assured that the associated region is nonempty.

Proposition 31. *Suppose the hypothesis of Proposition 30 holds at a given* x*. Then for each subset* $N \subset I^0(x)$*, the region* $R(M)$ *with* $M = I^+(x) \cup N$ *is nonempty.*

Proof. From Proposition 30, the hyperplanes $\{H_{\ell,i} = \{x : A_i^\ell \hat{A}_M^{\ell-1} \cdots \hat{A}_M^1 x = 0\} : \ell = 1, \ldots, L, i \in I^{0,\ell}(x)\}$ are in general position; and since these form a central arrangement (centered at x), there must be a nonempty region with $A_i^\ell \hat{A}_M^{\ell-1} \cdots \hat{A}_M^1 x > 0$ if $(\ell, i) \in M$ and $A_i^\ell \hat{A}_M^{\ell-1} \cdots \hat{A}_M^1 x < 0$ otherwise. □

The condition that the hyperplanes enumerated in Proposition 30 be in general position is fairly mild. Certain types of neural networks will violate this assumption: for example, zero-bias neural networks will violate this assumption since each hyperplane will go through the origin. Additionally, we cannot have $|I^0(x)|$ be larger than the dimension of the input space since in this case, we will also have too many intersecting hyperplanes. That being said, we generally expect a collection of randomly generated hyperplanes to be in general position with probability 1 under most natural probability distributions.

With these tools in hand, we can now state in full the stationarity conditions for the embedded problem:

Theorem 1. *Suppose at a given x^*, the conditions of Proposition 30 hold. Then x^* is a stationary point of* (5.8) *if there exist nonnegative multipliers μ^* and matrices $\hat{W}^1, \ldots, \hat{W}^L$ with $\hat{W}^1 \cdots \hat{W}^L \in \partial DNN(x)$ such that*

1. $c(DNN(x^*), x^*) \leq 0$
2. ${\mu^*}^T c(DNN(x^*), x^*) = 0$
3. $\nabla_x f^* + \nabla_x c^* \mu^* + \hat{W}^1 \cdots \hat{W}^L (\nabla_y f^* + \nabla_y c^* \mu^*) = 0,$

where f^*, c^* indicate evaluation of f, c at x^*.

Proof. The first two conditions are standard primal feasibility and complementary slackness conditions. To arrive at the third, we note that the gradient of the Lagrangian $L(x, \mu) = f(\mathrm{DNN}(x), x) + \mu c(\mathrm{DNN}(x), x)$ on each region (with activation pattern B) nearby is given by $\nabla_x f + \nabla_x c\mu + J_{F_B}(\nabla_y f + \nabla_y c\mu)$ evaluated at x. As $x \to x^*$ in a given region, since f

and c are smooth, this approaches $\nabla_x f^* + \nabla_x c^* \mu + J_{F_B}(\nabla_y f^* + \nabla_y c^* \mu)$. As the generalized gradient at x^* is given by the convex hull of all such gradients and by Proposition 31 there is one for region $I^+(x) \cup A$ with $A \subset I^0(x)$, it follows in a similar fashion as in the proof of Proposition 28 that any element of the generalized gradient can be written $\nabla_x f^* + \nabla_x c^* \mu^* + \hat{W}^1 \cdots \hat{W}^L (\nabla_y f^* + \nabla_y c^* \mu^*)$. Thus condition 3 holds. □

5.4 Formulating DNNs as Optimization Models

We provide two alternative formulations of ReLU DNNs in terms of optimization models that avoid the pitfalls of the embedded formulation in Section 5.3.1. The first formulation uses binary variables to model the max-functions, resulting in a mixed-integer program, building on [GA19, FJ18, AHT19]. Our formulation differs from [Che21], which considered only inverse problems and did not exploit the convex structure of the DNNs that arises when we lift the DNN constraint, resulting in a nonconvex mode. The second formulation uses complementarity constraints that can be solved as systems of nonlinear inequalities. We derive theoretical properties of both formulations.

5.4.1 Formulating DNNs with Mixed-Integer Sets

In this section we show how general optimization problems involving DNNs, such as (5.1), can be equivalently formulated as convex MIPs, extending [FJ18].

We assume that the DNN is a deep neural network with ReLU activation functions, and we rewrite the nonconvex problem (5.1) as a constrained problem:

$$\underset{x, y^L}{\text{minimize}} \; f(x, y^L) \quad \text{subject to } y^L = \text{DNN}(x),\; c(x, y^L) \leq 0,\; x \in X, \tag{5.12}$$

where L is the number of layers of the DNN. Fischetti and Jo [FJ18] have shown that the nonconvex constraint, $y^L = \text{DNN}(x)$, can be formulated as a mixed-integer linear set in the case of DNNs with ReLU activation functions. We let w_i^ℓ denote the weights of neuron

$i = 1, \ldots, N_\ell$ at level $\ell = 1, \ldots, L$ and b_i^ℓ its corresponding bias. Then, the levels $\ell = 1, \ldots, L$ are computed as

$$y_i^\ell = \mathrm{ReLU}\left(w_i^{\ell^T} y^{\ell-1} + b_i^\ell\right) = \max\left(0, w_i^{\ell^T} y^{\ell-1} + b_i^\ell\right). \tag{5.13}$$

We lift this constraint by introducing slack variables, s_i^ℓ, and binary variables, $z_i^\ell \in \{0,1\}$, and observe that (5.13) is equivalent to the mixed-integer linear constraints

$$y_i^\ell - s_i^\ell = w_i^{\ell^T} y^{\ell-1} + b_i^\ell,\ 0 \le y_i^\ell \le M_i^{\ell,y}(1 - z_i^\ell),\ 0 \le s_i^\ell \le M_i^{\ell,s} z_i^\ell,\ z_i^\ell \in \{0,1\}, \tag{5.14}$$

where $M_i^{\ell,y}, M_i^{\ell,s} > 0$ are sufficiently large constants (if $z_i^\ell = 1$, we are on the 0-branch of ReLU, and if $z_i^\ell = 0$, we are on the positive branch). By substituting (5.14) into (5.12) we obtain a convex MINLP that is equivalent to the problem (5.1):

$$\begin{aligned}
\underset{x,y,s,z}{\text{minimize}} \quad & f(x, y_L) \\
\text{subject to} \quad & c(x, y_L) \le 0 \\
& y_i^\ell - s_k^\ell = w_i^{\ell^T} y^{\ell-1} + b_i^\ell, && \ell = 1, \ldots, L \\
& 0 \le y_i^\ell \le M_i^{\ell,y}(1 - z_i^\ell),\ 0 \le s_i^\ell \le M_i^{\ell,s} z_i^\ell, && i = 1, \ldots, N_\ell, \ell = 1, \ldots, L \\
& y_0 = x,\ z_i^\ell \in \{0,1\}, x \in X.
\end{aligned} \tag{5.15}$$

We observe that we have as many binary variables in this problem as we have ReLU nodes. Next we show that the resulting lifted formulation results in a tractable convex MINLP.

Proposition 32. *Let Assumption 1 hold. Then it follows that* (5.15) *is a convex MINLP in the sense that the continuous relaxation of* (5.15) *is a convex NLP.*

Proof. The result follows from the convexity of f, c, and X and the fact that the remaining constraints are affine (with the exception of the integrality restriction on z). □

The choice of the big-M constants M_i^ℓ in (5.14) can have a great impact on the solution time of our MIP. If M_i^ℓ is too small, the problem excludes solutions that should be feasible; but if it is too large, the space that must be searched by the solver may become so large as to be too computationally intractable for most computers.

These upper bounds are common in mixed-integer programs, and we follow the approach in [FJ18] where we consider each neuron in our neural network individually, removing all constraints on other neurons in the same or subsequent layers. We do not relax the integrality constraints on the neurons remaining. We also retain the convex hull constraints as presented in Section 5.2.1.2 as well as other constraints on the input, but discard the constraints on the output of the neural network, namely the torque bound. Then we set as our objective function to maximize y_i^ℓ in one iteration and s_i^ℓ in another iteration. We can then use these computed optimal values as upper bounds $M_i^{\ell,y}$ and $M_i^{\ell,s}$, respectively, for computing the big-M values of neurons deeper in the neural network. After we have processed all layers, we will have computed big-M bounds for all our variables. This process of obtaining bounds is related to the optimality-based bound-tightening technique used in global optimization; see, for example, [GBM17].

Since these constants depend solely on the weights for the neural network as well as inputs to the neural network, we need to compute these bounds only once and may reuse them in any optimization problem involving this neural network. Alternative methods for handling the big-M constraints are considered in [GA19, TXT17, TKT21].

With this formulation established, we may then easily pass the problem as is to any standard MINLP solver such as Gurobi [Gur12], CPLEX [IBM09], Bonmin [BL07], MINOTAUR [MLL11], or Baron [Sah96] to compute the solution. Because MINLPs are NP-complete, however, these problems do not scale well, and only problems involving modestly sized networks (on the order of 100 hidden nodes) may be tractably solved. Neural networks used in commercial settings generally involve at least thousands of hidden nodes, resulting in thousands of binary variables, a number that typically is well beyond the reach of commercial solvers.

5.4.2 Formulating DNNs with Complementarity Constraints

In this section we discuss an alternative formulation of (5.1) as a nonconvex nonlinear program using complementarity constraints. This approach has the advantage of scaling significantly better for problems with larger neural networks, but with the caveat that solutions produced can be guaranteed only to be locally optimal. The

Our approaches are based on the following observations. We can rewrite the ReLU activation function in (5.13) equivalently as a complementarity constraint (using vector notation):

$$y^\ell = \max(W^{\ell^T} y^{\ell-1} + b^\ell, 0) \;\Leftrightarrow\; 0 \le y^\ell \;\perp\; y^\ell \ge W^{\ell^T} y^{\ell-1} + b^\ell, \tag{5.16}$$

where $\perp$ means that for each component, i, both inequalities $y_i^\ell \ge 0$ and $y_i^\ell \ge [W^{\ell^T} y^{\ell-1} + b^\ell]_i$ are satisfied and at least one is satisfied at equality. By replacing the ReLU function with these complementarity constraints, we obtain a mathematical program with complementarity constraints, which we can solve using standard NLP solvers; see, for example, [Ley03, LLN06, FLR06, RB05]. The reformulation of the ReLU activation function using complementarity constraints is based on a classical reformulation of the max function, see, e.g. [BRB08]. A similar approach is taken in [ZB17], where the authors consider a reformulation of the ReLU function as a constrained projection problem, and add a quadratic penalty or regularization to the loss function. However, unlike our approach, this reformulation is not exact, and provides a relaxation only, because the ReLU constraints are generally violated for any penalty parameter less than infinity. The mathematical program modeling language `AMPL` [FGK93] allows the modeling of complementarity constraints: for example, the above constraint can be written as follows:

```
ReLUCompl{l in Level, i in Neuron[l]}: 0 <= y[l,i] complements
y[l,i] >= sum{j in Neuron[l-1]} W[l,i,j] y[l-1,j] + b[l,i] .
```

The most successful NLP solvers handle MPCCs by reformulating the complementarity constraints in (5.16) by first introducing the same slack variables as for the MINLP and

lifting the formulation

$$s^\ell = y^\ell - w_i^{\ell^T} y^{\ell-1} + b^\ell,$$

and then rewriting (5.16) equivalently as

$$s^\ell = y^\ell - w_i^{\ell^T} y^{\ell-1} + b^\ell \text{ and } 0 \le y^\ell \perp s^\ell \ge 0.$$

A nonlinear optimization formulation is then given as

$$s^\ell = y^\ell - w_i^{\ell^T} y^{\ell-1} + b^\ell, \quad y^\ell \ge 0, \quad s^\ell \ge 0, \text{ and } y^{\ell^T} s^\ell \le 0,$$

where the lower bound, $y^{\ell^T} s^\ell \ge 0$, is implied by the nonnegative bounds on $y^\ell, s^\ell \ge 0$, and again omitted for numerical reasons; see [FLR06].

Because the last constraint involves the nonconvex term $y^{\ell^T} s^\ell$, the problem as a whole is nonconvex, and standard NLP solvers will produce only locally optimal solutions instead of globally optimal ones. Altogether, we then have the following NLP:

$$\begin{array}{lll} \underset{x,y,s,z}{\text{minimize}} & f(x, y_L) & \\ \text{subject to} & c(x, y_L) \le 0 & \\ & y_i^\ell - s_i^\ell = w_i^{\ell^T} y^{\ell-1} + b^\ell i, & \ell = 1, \ldots, L \\ & 0 \le y_i^\ell,\ 0 \le s_i^\ell, & i = 1, \ldots, N_\ell, \ell = 1, \ldots, L \\ & \sum_{\ell=1}^{L} \sum_{i=1}^{N_\ell} y_i^\ell s_i^\ell \le 0, & \\ & y_0 = x,\ x \in X. & \end{array} \tag{5.17}$$

Note that we could have used the constraint $y^{\ell^T} s^\ell \le 0$ for each ℓ separately. We prefer the formulation in (5.17) because it has better convergence behavior in practice; see [FL04]. Again we have a continuous variable for every node in our neural network as well as a slack variable for each ReLU node. The absence of integer variables, however, produces a much more scalable problem.

We now demonstrate that the stationarity conditions for the MPCC formulation coincide with those of the embedded formulation. In MPCC form, the optimization problem is given

by the following:

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & f\left(x^0, x^L\right)\right) \\ \text{subject to} \quad & c\left(x^0, x^L\right)\right) \leq 0 \\ & 0 \leq x^\ell \;\perp\; x^\ell \geq W^{\ell^T} x^{\ell-1} + b^\ell,\ \ell = 1, \ldots, L. \end{aligned} \tag{5.18}$$

First, we state the definition of strong stationarity for the MPCCs (5.18); see, for example, [SS00] for its general form. We then show that the two conditions are equivalent in our case.

Definition 33 (Scheel and Scholtes, [SS00])**.** *We say that* $(x^{0,*}, x^{1,*}, \ldots, x^{L,*})$ *is a strongly stationary point of* (5.18) *if there exist multipliers* $\mu^* \geq 0$ *and* ν_1^*, ν_2^* *such that the following conditions are satisfied:*

$$c^* \leq 0 \text{ and } 0 \leq x^{\ell,*} \;\perp\; x_{\ell,*} \geq W^{\ell^T} x^{\ell-1,*} + b^\ell,\ \ell = 1, \ldots, L \tag{5.19a}$$

$$\mu^{*,T} c^* = 0 \tag{5.19b}$$

$$0 = \nabla_{x^0} f^* + \nabla_{x^0} c^* \mu^* + W^1 \nu_2^{1,*} \tag{5.19c}$$

$$-\nu_1^{\ell,*} - \nu_2^{\ell,*} + W^{\ell+1} \nu_2^{\ell+1,*} = 0,\ \ell = 1, \ldots, L-1 \tag{5.19d}$$

$$\nabla_{x^L} f^* + \nabla_{x^L} c^* \mu^* - \nu_1^{L,*} - \nu_2^{L,*} = 0 \tag{5.19e}$$

$$x_j^{\ell,*} > 0 \Rightarrow \nu_{1j}^{\ell,*} = 0 \text{ and } x_j^{\ell,*} > w_j^{\ell^T} x^{\ell-1,*} + b_j^\ell \Rightarrow \nu_{2j}^{\ell,*} = 0,\ \ell = 1, \ldots, L \tag{5.19f}$$

$$0 = x_j^{\ell,*} = w_j^{\ell,T} x^{\ell-1,*} + b_j \Rightarrow \nu_{1j}^{\ell,*} \geq 0, \nu_{2j}^* \geq 0,\ \ell = 1, \ldots, L, \tag{5.19g}$$

where all functions and gradients are evaluated at (x^*, y^*)*, that is,* $f^* := f(x^*, y^*)$.

Using these stationarity conditions, we can arrive at the following conditions on the gradients of our optimization problem.

Proposition 34. *Given a strongly stationary point satisfying* (5.19)*, there exist* $\hat{W}^1, \ldots, \hat{W}^L$*, column scalings of* $W^1, \ldots, W^L$ *such that* $\nabla_{x^0} f^* + \nabla_{x^0} c^* \mu^* + \hat{W}^1 \cdots \hat{W}^L (\nabla_{x^L} f^* + \nabla_{x^L} c^* \mu^*) = 0$.

Proof. To construct such matrices, we work backward starting with layer L and examine each component $j = 1, \ldots, N_L$. Backsubstituting $\nu_2^{L,*}$ using (5.19d) and (5.19e), we have

$$-\nu_1^{L-1,*} - \nu_2^{L-1,*} + W^L (\nabla_{x^L} f^* + \nabla_{x^L} c^* \mu^* - \nu^{L,*}) = 0.$$

For a given component j we can determine the appropriate scaling factor κ_j^L in three cases:

1. $w_j^{L^T} x^{L-1,*} + b_j^L < 0$: Then it follows that $x_j^{L,*} = 0$, $\nu_{2j}^{L,*} = 0$ and ν_{1j}^* is not sign-constrained. It follows from (5.19e) that $\nu_{1j}^{L,*} = \left[\nabla_{x^L} f^* - \nabla_{x^L} c^* \mu^*\right]_j$ and hence that $\kappa_j^L = 0$.

2. $w_j^{L^T} x^{L-1,*} + b_j^L > 0$: Then it follows that $x_j^{L,*} = w_j^{L,T} x^* + b_j > 0$ and hence that $\nu_{1j}^{L,*} = 0$. Thus, we can set $\kappa_j^L = 1$.

3. $w_j^{L^T} x^{L,*} + b_j^L = 0 = x_j^{L,*}$: Then it follows that $\nu_{1j}^{L,*}, \nu_{2j}^{L,*} \geq 0$ from (5.19g). From (5.19e), we obtain that $\left[\nabla_{x^L} f^* + \nabla_{x^L} c^* \mu^*\right]_j \geq \nu_{1j}^{L,*}, \nu_{2j}^{L,*} \geq 0$; and given $\nu_{1j}^* \in \left[0, \left[\nabla_y f^* + \nabla_y c^* \mu^*\right]_j\right]$, we choose $\kappa_j^L = \frac{\left[\nabla_{x^L} f^* + \nabla_{x^L} c^* \mu^*\right]_j - \nu_j^{L,*}}{\left[\nabla_{x^L} f^* + \nabla_{x^L} c^* \mu^*\right]_j}$, which is in $[0, 1]$.

Hence we can write $W^L\left(\nabla_{x^L} f^* + \nabla_{x^L} c^* \mu^* - \nu^{L,*}\right) = \hat{W}^L\left(\nabla_{x^L} f^* + \nabla_{x^L} c^* \mu^*\right)$, where $\hat{W}^L = W \mathrm{diag}(\kappa^L)$. Repeating this process for each ℓ, we have

$$\nabla_{x^0} f^* + \nabla_{x^0} c^* \mu^* + \hat{W}^1 \cdots \hat{W}^L \left(\nabla_{x^L} f^* + \nabla_{x^L} c^* \mu^*\right) = 0.$$

□

We can now prove the following theorem.

Theorem 2. *Given x^*, suppose the assumptions of Proposition 30 hold. Then x^* is a stationary point of* (5.8) *if and only if $(x^{0,*}, x^{1,*}, \ldots, x^{L,*})$ is a strongly stationary point of* (5.18) *with $x^{\ell,*}$ being the activations of layer ℓ in the neural network.*

Proof. Since the given expression lies in the generalized gradient for (5.8) and equals zero, strong stationarity of the MPCC formulation clearly implies stationarity in the embedded formulation. On the other hand, if 0 is in the generalized gradient for the embedded formulation, as we know the form of the generalized gradient, we can find $\hat{W}^1, \ldots, \hat{W}^L$ such that $\nabla_{x^0} f^* + \nabla_{x^0} c^* \mu^* + \hat{W}^1 \cdots \hat{W}^L \left(\nabla_{x^L} f^* + \nabla_{x^L} c^* \mu^*\right) = 0$. The choice of scaling factors κ for the matrices will then determine the values of ν as in Proposition 34 so that the conditions for strong stationarity hold. □

5.5 Numerical Experiments

Here we present our numerical experiments with neural-network surrogates for the three sample applications introduced in Section 5.2.

We assess the performance of solvers for the MIP, MPCC, and embedded formulations of (5.3). We use the commercial solver CPLEX [IBM09] to solve the MIP formulation of the problem and the solver Ipopt [WB06] to solve the MPCC and embedded formulations of the problem. Both solvers are run with the default options. All experiments are performed on a single thread on an Intel Xeon Gold 6130 CPU and 188 GB of memory.

5.5.1 Numerical Experiments with Engine Design Optimization

We have applied our models and algorithms to a collection of neural networks with varying numbers of layers to study how well each formulation scales. The architectures we tested all have the simple structure of an input layer with the three input variables, followed by n hidden layers of 16 nodes, and then the output layer with the three output variables. We consider networks with 1, 3, and 5 hidden layers and train each of these networks for 20 epochs on the simulation data produced by the simulator of [AB19]. We use the `adam` solver in TensorFlow [ABC16] to train the neural network.

The number of auxiliary variables for the MIP and MPCC formulations scales with the number of ReLU neurons as well as with the number of time steps. The last layer has no ReLU neurons since we want the final layer to be able to take all real values, but each of the hidden layers uses ReLU neurons. We then have $T \times (\#\text{Hidden Layers}) \times (\#\text{Nodes Per Layer})$ additional sets of auxiliary variables.

We will see that using this formulation to solve the full integer program with 1,500 time steps to optimality is generally intractable (even for a single-layer network, this amounts to $1500 \times 16 \times 1 = 24000$ binary variables), so we consider instead a coarser discretization using larger time steps.

Instead of using data at each second as is presented in the original data, we consider time intervals of 750, 500, 250, 150, 30, 15, 10, 5, and 1 seconds. Since we have 1,500 seconds of data, this corresponds to 3, 6, 10, 50, 100, 150, 300, and 1,500 time steps for which we are evaluating our neural network. At each of the larger time steps, our prescribed torque profile will be the average of the prescribed torques in that interval. This approach gives us separate problems for each choice of coarseness of the discretization and for each choice of neural network architecture. As the number of time steps increases, we have observed that the computed solutions converge to the solution of the full problem.

For each configuration, we consider ten instantiations of the same neural networks trained on the same data and we run each solve until convergence to a solution, 3 hours have passed, or Ipopt has performed 3,000 iterations. If the solver has not converged at the end of the experiment, we record the best solution discovered that is feasible which we define as having a constraint violation under 10^{-6}. Our results are all averaged over the ten instantations.

Figure 5.10 shows the average amount of time needed until a solver found its best solution with a time limit of 3 hours on each architecture with each time step size. This time does not include the time to find a warmstart solution, which takes at most a few seconds to compute for the instances with the most timesteps. The percentage gap between the computed objective and the best-known objective for each solver is shown in Figure 5.11. Note that each time recorded in Figure 5.10 is not the full time that each solver ran for and instead records only the time taken until the solver's iterate was optimal. We believe this more fairly represents the solves of the embedded formulation as Ipopt can often find good solutions among its iterates, but fails to converge to these points due to the issues of nondifferentiability discussed in 5.3.1. The full run times for each solve are included in the appendix as well as the fully tabulated results that comprise Figure 5.10 and Figure 5.11.

Except for the smallest problems, CPLEX fails to converge to the optimal solution of the MIP within the allotted time limit of 3 hours. Even worse, it fails to even find a feasible solution for most problems with architectures with 3 or 5 hidden layers. This result is to

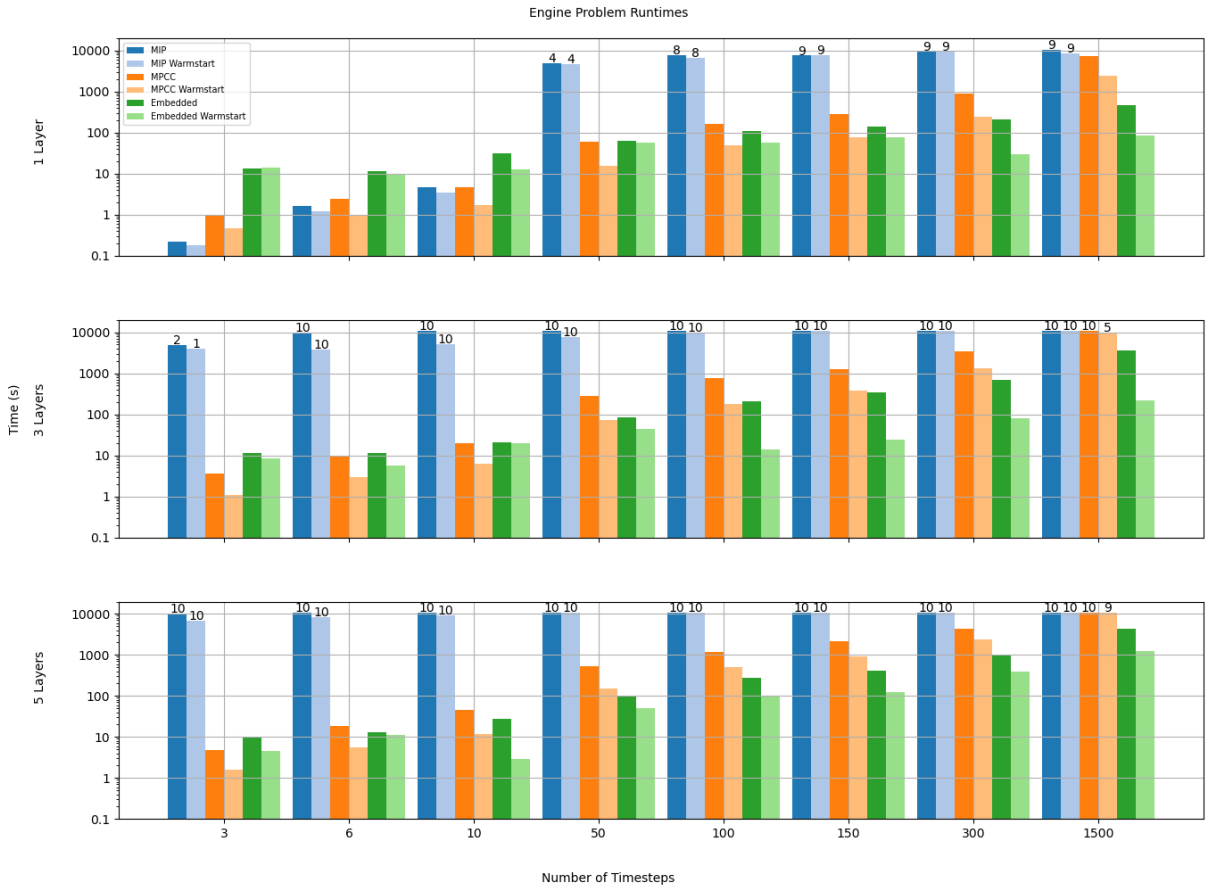

Figure 5.10: Average time until a solver found its best solution to the engine design problem within 3 hours (10,800 seconds). Experiments are averaged over ten runs on different neural networks. If a runtime of 10,800 seconds is recorded, this indicates the solver failed to find a feasible solution in the full 3 hours (if a warmstart is used, this means it failed to find a feasible solution better than the warmstart). If a number is present over a bar, this indicates the number of solves which did not terminate in the full 3 hours given.

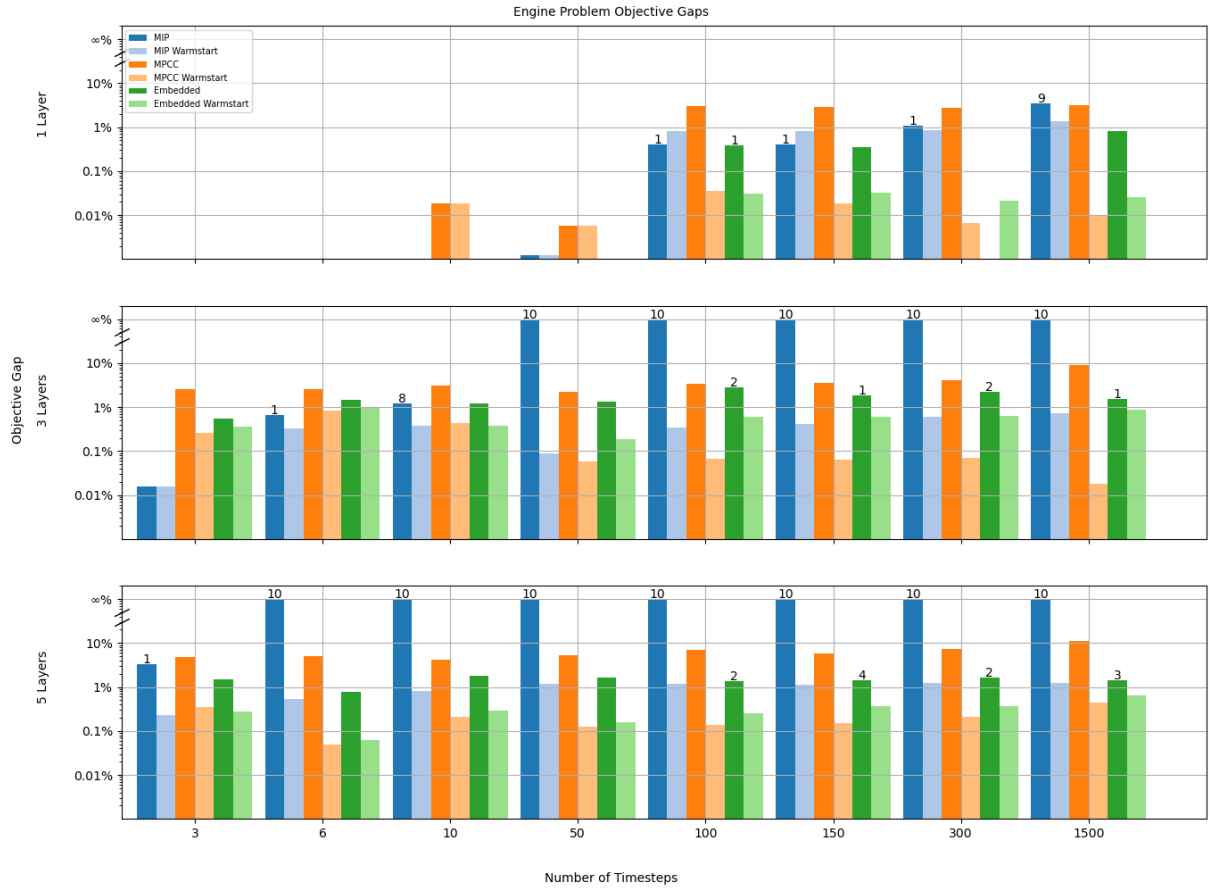

Figure 5.11: Percentage gap in objective between final objective value and best-known objective value of the engine design problem. Experiments are averaged over ten runs on different neural networks. A value of 0 indicates the solver found the best-known objective value, and a value of ∞ indicates that no solution was found over any solve. If a number is displayed over a bar, this indicates the number of solves of the ten runs which failed to determine a feasible solution. Cases where the solver found no feasible solution are not included in the average.

be expected because, except for the smallest cases, these problems can involve thousands of binary variables. With the warmstart, CPLEX is actually able to find solutions with significantly better objective values for each of the configurations. It still fails to prove optimality for any of these solutions, however, and times out for the same set of problems as without the warmstart.

For the MPCC, on the other hand, Ipopt always finds locally optimal solutions within the time limit with the exception of the problems with 1500 timesteps. The time taken is often in seconds for problems with a small number of time steps. These solutions may be suboptimal, however, worse by up to 10% in some instances. With the warmstart solution, Ipopt solves the MPCC in terms of speed generally by a factor of 2-3. Furthermore, the objective solution tends to have a significantly smaller objective. Solving the MIP without warmstart finds a better solution than does the MPCC formulation in only three problem instances, and in all three cases the MPCC solution is within 0.1% of the MIP's globally optimal solution.

In almost all cases the embedded network formulation is able to find its best solution more quickly than does either the MIP or the MPCC formulation. Without the warmstart, the embedded ReLU network formulation performs better in almost all cases where it found a solution. However, we observe that in the problems with more timesteps, there are difficulties with finding a feasible solution, and for problems with more than 100 timesteps on 3 or 5 layers, Ipopt fails to find a feasible solution for between 1 and 4 of the 10 trial runs. With the warmstart, the objective values are comparable between the MPCC and the embedded formulation, with the MPCC formulation edging out the embedded formulation in almost all cases.

Overall, these experiments confirm our prior suspicions that using the MIP formulation without warmstart quickly becomes computationally intractable as the size of the neural network increases past modest architectures. Switching to the MPCC formulation of the problem offers significant speedup at the cost of losing global optimality, although when the

MPCC formulation obtains a worse objective value, it tends to be by only a marginal amount, and when it does better, the improvement can be by a significant amount. The embedded formulation, on the other hand, can provide speed and scalability but often encounters difficulty with convergence finding slightly better values than the MPCC solutions without warmstart and slightly worse objective values than the MPCC solutions with warmstart. For all formulations, significant gains can be realized in terms of both solving time and solution quality by providing the solver with a high-quality warmstart solution.

5.5.2 Numerical Experiments with Adversarial Attack Generation

Next we consider each of the formulations of the adversarial attack generation problem in (5.6) by considering neural networks trained on the MNIST handwritten digit recognition data set [LeC98]. We consider 10 different architectures corresponding to having an input layer with 28×28 nodes followed by either 1 or 2 fully connected layers with 20, 40, 60, 80, or 100 hidden ReLU nodes each and then the output layer with 10 nodes and a softmax activation function.

If we use the mixed-integer formulation from (5.15) for the DNN constraints, then depending on our choice of norm, it becomes a mixed-integer linear program (for L^1 or L^∞ norms) or a mixed-integer quadratic program (for L^2 norm) that we can solve using CPLEX. We will use the L^2 norm in experiments.

For each neural network, we train the model by minimizing the categorical cross-entropy loss function for 10 epochs using the 60,000 digits of training data. Each neural network attains a test accuracy of 96–98% when tested on the 10,000 digits of testing data.

We set $\alpha = 1.2$, meaning the probability for the given classification l is 1.2 times higher than for any other. On each architecture, we solve the problem for 100 digits from the training data, with the goal of finding the closest image to the given digit that will be classified as a zero. As a warmstart, we initialize the solution in each solve to be a digit that

Architecture	MIP				MPCC			Embedded ReLU			
	Avg. Time	Avg. Obj.	Num. Opt.	Num. Feas.	Avg. Time	Avg. Obj.	Num. Feas.	Avg. Time	Avg. Obj.	Avg. Feas. Obj.	Num. Feas.
784,20,10	1.8	**1.79**	100	100	**0.3**	1.80	100	35.5	14.67	1.80	88
784,40,10	76.9	**1.75**	100	100	**1.5**	1.77	100	28.4	8.64	1.77	94
784,60,10	2302	**1.85**	52	100	**4.8**	1.87	100	31.8	1.91	1.91	100
784,80,10	2571	**2.04**	38	100	**14.2**	2.07	100	48.1	17.35	2.08	86
784,100,10	3078	**2.16**	16	100	**22.9**	2.18	100	30.1	10.66	2.09	92
784,20,20,10	106.1	**1.20**	100	100	**0.8**	1.22	100	26.6	4.04	4.04	100
784,40,40,10	3100	1.90	16	100	**4.4**	**1.77**	100	38.8	19.97	1.68	84
784,60,60,10	3513	6.85	5	100	**14.3**	**1.88**	100	29.8	12.61	12.61	100
784,80,80,10	3587	27.59	1	76	36.1	**1.46**	100	**27.8**	3.83	3.83	100
784,100,100,10	3600	54.83	0	42	57.0	**1.78**	100	**35.1**	1.83	1.83	100

Table 5.3: Comprehensive results for solves of the MIP formulation using CPLEX, the MPCC formulation using Ipopt, and the embedded ReLU formulation also using Ipopt. Times are recorded in seconds.

is classified as the desired digit, so that the solver starts at a feasible point. We run each iteration until convergence to the globally optimal solution (for the MIP formulation) or to a locally optimal solution (for the MPCC and embedded network formulations) or until one hour has passed or 3,000 iterations have occurred in Ipopt, at which point we terminate with the best feasible solution seen so far.

In Table 5.3 we tabulate the results of the experiments for all three formulations. For each neural network architecture we record the average solve times and objectives for CPLEX and Ipopt over the 100 solves. We also record how many times CPLEX finds the optimal solution to the MIP as well as how many times in each formulation a feasible solution that is better than the initial solution is found. For the embedded formulation we also include the average objective excluding the infeasible problems since we observed a significant difference between these cases.

The average time to convergence for the CPLEX solves of the MIP formulation is significantly higher than that of the Ipopt solves of the MPCC formulation. Most CPLEX solves for the larger networks time out at 1 hour before finding an optimal solution; for the smaller networks, it can take 10 to 100 times longer to converge to a solution. Ipopt, on the other hand, takes less than a minute on average to converge to a locally optimal solution of the MPCC formulation. The embedded formulations are fairly uniform in how long they take to find their best solution, which makes sense given that the problem does not change in size with the number of neurons in the network.

The longer solve times of the MIP formulation have the advantage of eventually uncovering better solutions the majority of the time even if they are not provably optimal. For all the single hidden layer architectures and the two smallest double hidden layer architectures, the MIP formulation almost always produces a better solution than the MPCC formulation does. These solutions generally offer only a marginal improvement on the order of 1–2% difference in the objective. For perturbations of this magnitude, the differences are essentially imperceptible.

For the largest networks the MIP formulation fares poorly and cannot find a better feasible solution than the initial solution in all cases for the two largest networks. The feasible solutions it does find are of noticeably lower quality: of all instances from the three largest networks, CPLEX found the best solution in only 13 cases out of 300. The MIP solutions for the larger networks are visually worse, as can be seen in a comparison between the perturbed images generated from CPLEX and Ipopt in Figure 5.13 with respect to the original images in Figure 5.12.

When compared with the MIP and MPCC formulations, the embedded formulation, while often faster and more scalable, can be more unstable in the sense that while using it, Ipopt can fail to find feasible solutions even though for this problem they certainly exist. The ReLU networks are more prone to failure and exhibit the poor convergence behavior as in Figure 5.6 leading to failure to find solutions in about 5% of instances. In a few cases the

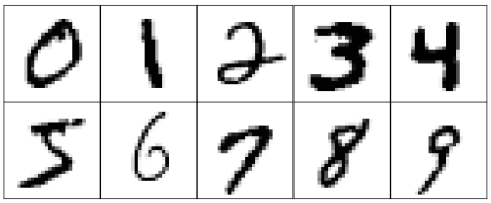

Figure 5.12: Suite of 10 digits from the MNIST training data for which adversarial attacks are generated.

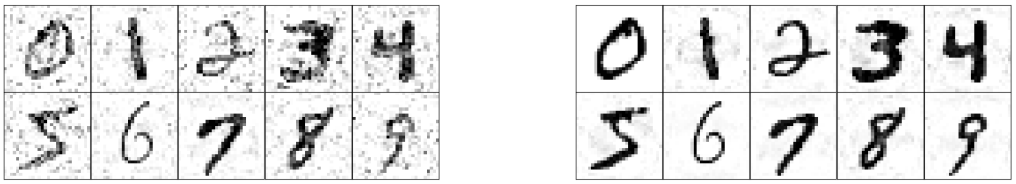

Figure 5.13: Perturbed images produced after 1 hour of computation when using CPLEX to solve the MIP formulation (left) and Ipopt to solve the NLP formulation (right) with given digits in Figure 5.12 for the neural network with 2 hidden layers of 100 nodes.

objective value of the found solutions is also noticeably worse.

5.5.3 Numerical Experiments with Oil Well Networks

Next we consider solving the oil well problem using each of our formulations. Following the authors in [GA19], we consider two different configurations of neural networks for this problem: a "shallow" network with few layers of many nodes and a "deep" network with many layers of fewer nodes. The shallow configuration had two hidden layers of 20 nodes for the well networks f_i and two hidden layers of 50 nodes for the riser networks g_e. The deep configurations had four hidden layers of 10 nodes for the well networks and 5 layers of 20 nodes for the riser networks. All in all, this corresponds to 520 ReLU nodes for each configuration.

We again used CPLEX for the MIP formulation and solved the problem with a time limit of 1 hour, recording the best time for each neural network configuration. This problem involves binary variables even in the complementary constraint formulation, so we needed to use an MINLP solver and elected to use the solver Bonmin [BL07]. We used the default branch-and-bound scheme with Ipopt as the NLP solver. Because no MINLP solver is currently supported on JuMP for handling both integer variables and user-defined nonlinear functions, we could not solve the problem as stated using the embedded neural network formulations.

The results for the solves on the full problem are presented in Table 5.4. We observe that in this small set of problems, the MPCC formulation outperforms the MIP formulation in both network configurations in terms of time taken as well as objective value produced. CPLEX fails to even produce a solution for the deep neural network configuration. These are hard problems; indeed, in 3 of the 4 solves performed, the solutions found are not proven optimal.

To obtain a problem that did not have binary variables, we considered fixing each of the

	Shallow		Deep	
Solver	Time (s)	Objective	Time (s)	Objective
CPLEX	3388*	1.271	3600*	-
bonmin	1865*	**1.283**	1275.3	**1.304**

Table 5.4: Time for each solver to find its best feasible solution for each instance of the full oil well problem within a time limit of 1 hour. The * indicates that the solver did not terminate within 1 hour. For these starred problems, a time of 3600 seconds indicates no solution was found; otherwise, the solution found was not deemed optimal (locally optimal, when using bonmin).

binary variables randomly to either 0 or 1 while still ensuring feasibility. If we represent the neural networks using complementary constraints or using the embedded formulation, the problems become standard NLPs that we can directly solve using Ipopt as before. In our setup we considered 10 different configurations of the binary variables, and we solved them using CPLEX for the MIP formulation and Ipopt for the rest. We ran each instance until convergence or 1 hour had passed, and we recorded the best solution found.

The results for the experiments performed with the binary variables fixed to arbitrary configurations are presented in Table 5.5, which details the solve times, and in Table 5.6, which details the objective values produced. We observe that, overall, the MPCC formulation clearly outperforms all the other formulations in terms of speed in that it is able to find solutions in about 2 seconds where the embedded network formulations take 10 to 20 seconds and the MIP formulation can take much longer. In fact, the MIP formulation fails to find a single solution for any of the instances using the deep neural network configuration, which suggests that optimization problems with deeper networks are more difficult than networks with shallow networks even if the number of nodes is the same.

In terms of objective value, for the shallow network all formulations were able to find

	Shallow			Deep		
Instance	MIP	MPCC	Embedded	MIP	MPCC	Embedded
1	17.5	**1.8**	10.3	3600*	**1.6**	16.7
2	22.0	**1.2**	8.2	3600*	**1.4**	19.4
3	23.9	**1.5**	7.5	3600*	**1.7**	14.2
4	40.3	**1.5**	6.2	3600*	**1.4**	7.2
5	19.0	**1.4**	4.0	3600*	**1.6**	13.0
6	21.0	**2.1**	3.0	3600*	**1.7**	7.1
7	169.4	**1.7**	5.4	3600*	**1.2**	14.1
8	14.9	**1.5**	5.7	3600*	**1.4**	7.3
9	86.3	**1.9**	4.3	3600*	**1.6**	12.1
10	15.5	**1.7**	5.8	3600*	**1.5**	5.8

Table 5.5: Time until a solver finds its best feasible solution within an hour time limit for each formulation and for both shallow and deep neural networks on each particular fixing of the oil well problem. The * indicates that the solver did not terminate within 1 hour. For these starred problems, a time of 3600 seconds indicates no solution was found; otherwise, the solution found was not deemed optimal (locally optimal, when using Ipopt).

	Shallow			Deep		
Instance	MIP	MPCC	Embedded	MIP	MPCC	Embedded
1	**1.231**	**1.231**	**1.231**	-	**1.248**	1.183
2	**1.228**	**1.228**	**1.228**	-	**1.228**	1.155
3	**1.219**	**1.219**	**1.219**	-	1.229	**1.274**
4	**1.264**	**1.264**	**1.264**	-	**1.274**	1.255
5	**1.182**	**1.182**	**1.182**	-	1.192	**1.253**
6	**1.252**	**1.252**	**1.252**	-	1.256	**1.266**
7	**1.229**	**1.229**	**1.229**	-	**1.244**	1.215
8	**1.207**	**1.207**	**1.207**	-	1.201	**1.225**
9	**1.239**	**1.239**	**1.239**	-	1.249	**1.282**
10	**1.263**	**1.263**	**1.263**	-	**1.276**	1.243

Table 5.6: Objective value of the best feasible solution for each formulation for both shallow and deep neural networks on each particular fixing of the oil well problem. - indicates that no feasible solution was found.

the global optimum in every instance relatively quickly. For the deep networks the different formulations outperformed each other on different instances, leaving no clear winner. The MPCC formulation solves each instance the quickest, because of the relatively small size of each of the neural networks.

Unlike the other two problems, we observed for these instances that the embedded ReLU network formulation did not have trouble with convergence and that, in spite of nondifferentiability, the dual infeasibility was brought down to zero, signifying convergence in all instances. We postulate that the reason for the success here as compared with the other instances may be due to the simplicity of the networks: 8 of the networks are single input and single output, and so the difficulties that might emerge in multiple dimensions do not appear.

5.6 Conclusion

We have presented three alternative formulations of ReLU deep-neural network constraints as a mixed-integer problem, an optimization problem with complementarity constraints, and a problem with the neural network directly embedded. The MIP and MPCC formulations can be viewed as lifted formulations, and we have shown that the lifting convexifies optimization problems with deep neural network constraints in the case of the mixed-integer formulation. We have also presented a warmstart technique that uses training data of the neural network to construct good initial solutions. We have compared the three formulations on three examples arising in the design of engines, the design of images that "fool" a given classifier, and the assignment of flow in an oil well network. Each formulation has its advantages. We have shown the MIP formulation to be useful in finding an optimal solution, but at the cost of a potentially long solve time. We observed that the new complementarity constraint formulation generally outperforms the mixed-integer formulation in terms of solution time but may not find the optimal solution, although it often comes close. We also observed that

the embedded neural network formulation has the advantage of being scalable and quick to solve but has difficulties with convergence related to the nondifferentiability of the ReLU activation function (which could be rectified by using a smooth activation function, e.g., the swish function).

We can assess from these experiments some limitations of the MIP and MPCC formulations in that the number of variables in these models scale with the number of neurons of a network suggesting these models are intractable for networks with millions of neurons which boast the highest accuracy. The embedded formulation scales better and may be a better fit for these large models, but given the difficulties with convergence exhibited in this paper, we see that explicit care must be taken if we want to ensure that the solutions we find are in fact locally optimal. The experiments reported in this paper portray the difficulty in incorporating neural network models into general optimization problems, and provides three possible solutions each with potential advantages and drawbacks.

In the future, we hope to address through experimentation, the question of surrogate quality in the optimization process. We would like to develop algorithms which incorporate retraining of the neural network surrogates and explore conditions necessary to ensure convergence of the solution to local optima of the original optimization problems. In addition, our study was limited to problems involving generally linear and quadratic objectives, and linear constraints outside of the DNN constraints. Future work would involve a study of this problem with more complex nonlinear objectives and constraints. For example one can consider the problem of synthesizing process systems [DG86a] with additional physical constraints modeled by neural networks as is done in the oil well setting or the optimal product location problem [DG86b] with the cost function replaced by a surrogate. These problems and similar MINLPs are still convex and so the theorems in this paper still hold, but work remains to be done to empirically assess the utility of these formulations in solving these problems.

CHAPTER 6

Conclusion

In this thesis, we addressed two separate problems: developing efficient algorithms for subgraph discovery and creating methods for incorporating neural network surrogates into optimization problems. Through our work on subgraph isomorphism, we introduced a collection of techniques which improve the ability to discover patterns in highly symmetric graphs as well as succinctly characterize these solutions. Our work on surrogate optimization problems allows for the easy incorporation of computationally expensive models into optimization programs through the use of neural network surrogates.

Through our work on characterizing symmetry in subgraph isomorphism, we have defined several different methods for handling equivalence in the subgraph matching problem. We established conditions under which vertices may be interchanged safely in a given subgraph isomorphism to produce more subgraph isomorphisms. We also established a hierarchy for these different equivalence methods which a user could use to tune their solver to incorporate more equivalence at the cost of more computation time. Through comprehensive assessment on a large benchmark set, we demonstrated that for certain highly symmetric instances, addressing symmetry is essential and results in exponentially more solutions found and orders of magnitude reductions in computation time.

We established a concrete formulation for how one might try to apply active learning to the subgraph isomorphism problem. Through rigorous argument, we establish that this problem is NP-complete, and give in certain circumstances, a method for determining the optimal query sequence. We present many different approaches for how to solve this problem,

and through empirical assessment, we establish that many of these methods improve upon a naive approach of picking vertices at random.

In our work on surrogate optimization problems, we posed three different techniques by which one can plug a complicated and computationally intensive model into an optimization problem. We established through both rigorous analysis and empirical evaluation different reasons for considering each technique, and we can envision this method allowing for the solving of optimization problems which may have before appeared intractable due to the computational costs.

There are many future directions that would improve upon the research presented in this thesis. A key component lacking in the discussion of symmetry in subgraph isomorphism is how to incorporate both template and world symmetry simultaneously. We discuss this topic in the context of the simple forms of structural equivalence, but for dynamic candidate equivalence, there is work to be done to produce a method which utilizes both while still lending itself to easy counting. We would also like to develop methods which can incorporate a dynamic form of automorphic equivalence (or at least certain classes of automorphisms like reflections and rotations) into a tree search routine.

For the active learning problem, there still remains a significant gap between the best methods presented and the generally computationally intractable optimal routine. There is research which can be done to develop more sophisticated methods which may close this gap further. We would also like to consider mixed strategy methods which allow for queries on both the template and the world. There is also work to be done using different models for the information gained from a query; for example, we may gain information about a label or additional edges instead of the exact match.

As for the research on surrogate optimization problems, there remains much work on improving on the formulations presented in this work. For example, we could consider tighter formulations for the integer programming formulation. We would also like to expand our work to handle other types of neural networks or machine learning models other than

simply ReLU neural networks. We would also like to come up with formulations which could be scaled to the size of the models which are used in industry involving potentially millions or billions of neurons.

[AB19] S. M. Aithal and P. Balaprakash. "MaLTESE: Large-Scale Simulation-Driven Machine Learning for Transient Driving Cycles." In *High Performance Computing*, pp. 186–205, Cham, 2019. Springer International Publishing.

[ABC16] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al. "TensorFlow: A system for large-scale machine learning." In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pp. 265–283, 2016.

[ABK07] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives. "Dbpedia: A nucleus for a web of open data." In *The semantic web*, pp. 722–735. Springer, 2007.

[AHT19] R. Anderson, J. Huchette, C. Tjandraatmadja, and J. P. Vielma. "Strong Mixed-Integer Programming Formulations for Trained Neural Networks." In *International Conference on Integer Programming and Combinatorial Optimization*, pp. 27–42, 2019.

[AJB99] R. Albert, H. Jeong, and A.-L. Barabási. "Diameter of the world-wide web." *Nature*, **401**(6749):130–131, 1999.

[ALS14] G. Audemard, C. Lecoutre, M. Samy-Modeliar, G. Goncalves, and D. Porumbel. "Scoring-based neighborhood dominance for the subgraph isomorphism problem." In *Int. Conf. on Principles and Practice of Constraint Programming*, pp. 125–141. Springer, 2014.

[BBC14] S. Boccaletti, G. Bianconi, R. Criado, C. I. Del Genio, J. Gómez-Gardenes, M. Romance, I. Sendina-Nadal, Z. Wang, and M. Zanin. "The structure and dynamics of multilayer networks." *Physics reports*, **544**(1):1–122, 2014.

[BBL06] M.-F. Balcan, A. Beygelzimer, and J. Langford. "Agnostic active learning." In *Proceedings of the 23rd international conference on Machine learning*, ICML '06, pp. 65–72, Pittsburgh, Pennsylvania, USA, June 2006. Association for Computing Machinery.

[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.

[Bel20] P. Belotti. "Couenne: A user's manual." Technical report, FICO, 2020.

[BFG10] V. Bonnici, A. Ferro, R. Giugno, A. Pulvirenti, and D. Shasha. "Enhancing graph database indexing by suffix tree structure." In *Pattern Recognition in*

Bioinformatics: 5th IAPR International Conference, PRIB 2010, Nijmegen, The Netherlands, September 22-24, 2010. Proceedings 5, pp. 195–203. Springer, 2010.

[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):1–13, 2013.

[BHB21] D. Bergman, T. Huang, P. Brooks, A. Lodi, and A. U. Raghunathan. "JANOS: an integrated predictive and prescriptive modeling framework." *INFORMS Journal on Computing*, 2021.

[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 (Big Data)*, pp. 3986–3990, 10 2018.

[BL07] P. Bonami and J. Lee. "BONMIN user's manual." *Numer Math*, **4**:1–32, 2007.

[Bla92] K. D. Blaha. "Minimum bases for permutation groups: the greedy approximation." *Journal of Algorithms*, **13**(2):297–306, 1992.

[BMM06] D. Bruschi, L. Martignoni, and M. Monga. "Detecting self-mutating malware using control-flow graph matching." In *Detection of Intrusions and Malware & Vulnerability Assessment: Third International Conference, DIMVA 2006, Berlin, Germany, July 13-14, 2006. Proceedings 3*, pp. 129–143. Springer, 2006.

[BP20] J. Bolte and E. Pauwels. "Conservative set valued fields, automatic differentiation, stochastic gradient methods and deep learning." *Mathematical Programming*, pp. 1–33, 2020.

[BRB08] B. Baumrucker, J. Renfro, and L. Biegler. "MPEC problem formulations and solution strategies with chemical engineering applications." *Computers & Chemical Engineering*, **32**(12):2903–2913, 2008.

[CFS04a] D. Conte, P. Foggia, C. Sansone, and M. Vento. "Thirty years of graph matching in pattern recognition." *International journal of pattern recognition and artificial intelligence*, **18**(03):265–298, 2004.

[CFS04b] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento. "A (sub) graph isomorphism algorithm for matching large graphs." *IEEE Trans. Patt. Anal. Mach. Int.*, **26**(10):1367–1372, 2004.

[CFS07] D. Conte, P. Foggia, C. Sansone, and M. Vento. *How and Why Pattern Recognition and Computer Vision Applications Use Graphs*, pp. 85–135. Springer Berlin Heidelberg, 2007.

[CFS17] V. Carletti, P. Foggia, A. Saggese, and M. Vento. "Introducing VF3: A New Algorithm for Subgraph Isomorphism." *Graph-Based Representations in Pattern Recognition*, pp. 128–139, 2017.

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

[CGZ13] A. Cardillo, J. Gómez-Gardenes, M. Zanin, M. Romance, D. Papo, F. Del Pozo, and S. Boccaletti. "Emergence of network features from multiplexity." *Scientific reports*, **3**(1):1–6, 2013.

[Che21] M.-S. Cheon. "An outer-approximation guided optimization approach for constrained neural network inverse problems." *Mathematical Programming*, pp. 1–30, 2021.

[CKN07] J. Cheng, Y. Ke, W. Ng, and A. Lu. "Fg-index: towards verification-free query processing on graph databases." In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pp. 857–872, 2007.

[CLJ96] L. Clarke, J. Linderoth, E. Johnson, G. Nemhauser, R. Bhagavan, and M. Jordan. "Using OSL to Improve the Computational Results of a MIP Logistics Model." *EKKNEWS*, **16**, 1996.

[CNR17] C.-H. Cheng, G. Nührenberg, and H. Ruess. "Maximum resilience of artificial neural networks." In *International Symposium on Automated Technology for Verification and Analysis*, pp. 251–268. Springer, 2017.

[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 (Big Data)*, pp. 3947–3950, 10 2018.

[CS20] D. Conte and F. Serratosa. "Interactive online learning for graph matching using active strategies." *Knowledge-Based Systems*, **205**:106275, 2020.

[CW17] N. Carlini and D. Wagner. "Towards Evaluating the Robustness of Neural Networks." In *2017 IEEE Symposium on Security and Privacy (SP)*, pp. 39–57, 2017.

[CZZ13] W. Cai, Y. Zhang, and J. Zhou. "Maximizing Expected Model Change for Active Learning in Regression." In *2013 IEEE 13th International Conference on Data Mining*, pp. 51–60, December 2013. ISSN: 2374-8486.

[DAT20] A. Delarue, R. Anderson, and C. Tjandraatmadja. "Reinforcement Learning with Combinatorial Actions: An Application to Vehicle Routing." *Advances in Neural Information Processing Systems*, **33**, 2020.

[DG86a] M. A. Duran and I. Grossmann. "A mixed-integer nonlinear programming algorithm for process systems synthesis." *AIChE journal*, **32**(4):592–606, 1986.

[DG86b] M. A. Duran and I. Grossmann. "An Outer-Approximation Algorithm for a Class of Mixed-Integer Nonlinear Programs." *Mathematical Programming*, **36**:307–339, 1986.

[DGG92] A. C. Dumay, R. J. van der Geest, J. J. Gerbrands, E. Jansen, and J. H. Reiber. "Consistent inexact graph matching applied to labelling coronary segments in arteriograms." In *11th IAPR International Conference on Pattern Recognition. Vol. III. Conference C: Image, Speech and Signal Analysis,*, volume 1, pp. 439–442. IEEE Computer Society, 1992.

[DH08] S. Dasgupta and D. Hsu. "Hierarchical sampling for active learning." In *Proceedings of the 25th international conference on Machine learning*, ICML '08, pp. 208–215, Helsinki, Finland, July 2008. Association for Computing Machinery.

[DHL17] I. Dunning, J. Huchette, and M. Lubin. "JuMP: A Modeling Language for Mathematical Optimization." *SIAM Review*, **59**(2):295–320, 2017.

[DJS18] S. Dutta, S. Jha, S. Sankaranarayanan, and A. Tiwari. "Output range analysis for deep feedforward neural networks." In *NASA Formal Methods Symposium*, pp. 121–138. Springer, 2018.

[DLM13] M. De Domenico, A. Lima, P. Mougel, and M. Musolesi. "The anatomy of a scientific rumor." *Scientific reports*, **3**(1):1–9, 2013.

[DMF13] S. Demeyer, T. Michoel, J. Fostier, P. Audenaert, M. Pickavet, and P. Demeester. "The index-based subgraph matching algorithm (ISMA): fast subgraph enumeration in large networks using optimized search trees." *PloS one*, **8**(4):e61183, 2013.

[DSC13] M. De Domenico, A. Solé-Ribalta, E. Cozzo, M. Kivelä, Y. Moreno, M. A. Porter, S. Gómez, and A. Arenas. "Mathematical formulation of multilayer networks." *Physical Review X*, **3**(4):041022, 2013.

[DSH11] G. Damiand, C. Solnon, C. De la Higuera, J.-C. Janodet, and É. Samuel. "Polynomial algorithms for subisomorphism of nd open combinatorial maps." *Computer Vision and Image Understanding*, **115**(7):996–1010, 2011.

[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.

[DZP18] S. S. Du, X. Zhai, B. Poczos, and A. Singh. "Gradient Descent Provably Optimizes Over-parameterized Neural Networks." In *International Conference on Learning Representations*, 2018.

[EDS16] F. Emmert-Streib, M. Dehmer, and Y. Shi. "Fifty years of graph matching, network alignment and network comparison." *Information sciences*, **346**:180–197, 2016.

[FGK93] R. Fourer, D. M. Gay, and B. W. Kernighan. *AMPL: A Modeling Language for Mathematical Programming*. The Scientific Press, 1993.

[FJ18] M. Fischetti and J. Jo. "Deep neural networks and mixed integer linear optimization." *Constraints*, **23**(3):296–309, 2018.

[FL04] R. Fletcher and S. Leyffer. "Solving Mathematical Program with Complementarity Constraints as Nonlinear Programs." *Optimization Methods and Software*, **19**(1):15–40, 2004.

[FLR06] R. Fletcher, S. Leyffer, D. Ralph, and S. Scholtes. "Local convergence of SQP methods for Mathematical Programs with Equilibrium Constraints." *SIAM Journal on Optimization*, **17**(1):259—-286, 2006.

[FPV14] P. Foggia, G. Percannella, and M. Vento. "Graph matching and learning in pattern recognition in the last 10 years." *Int. J. of Pattern Recognition and Artificial Intelligence*, **28**(01):1450001, 2014.

[GA19] B. Grimstad and H. Andersson. "ReLU networks as surrogate models in mixed-integer linear programs." *Computers & Chemical Engineering*, **131**:106580, 2019.

[Gal63] D. Gale. "Neighborly and cyclic polytopes." In *Proc. Sympos. Pure Math*, volume 7, pp. 225–232, 1963.

[GB15] R. Gallotti and M. Barthelemy. "The multilayer temporal network of public transport in Great Britain." *Scientific data*, **2**:140056, 2015.

[GB21] Y. Ge and A. L. Bertozzi. "Active Learning for the Subgraph Matching Problem." In *2021 IEEE International Conference on Big Data (Big Data)*, pp. 2641–2649. IEEE, 2021.

[GBB11] X. Glorot, A. Bordes, and Y. Bengio. "Deep sparse rectifier neural networks." In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp. 315–323. JMLR Workshop and Conference Proceedings, 2011.

[GBB13] R. Giugno, V. Bonnici, N. Bombieri, A. Pulvirenti, A. Ferro, and D. Shasha. "Grapes: A software for parallel searching on biological graphs targeting multi-core architectures." *PloS one*, **8**(10):e76911, 2013.

[GBM17] A. M. Gleixner, T. Berthold, B. Müller, and S. Weltge. "Three enhancements for optimization-based bound tightening." *Journal of Global Optimization*, **67**(4):731–757, 2017.

[GFM14] S. Gay, F. Fages, T. Martinez, S. Soliman, and C. Solnon. "On the subgraph epimorphism problem." *Discrete Applied Mathematics*, **162**:214–228, 2014.

[GIG17] Y. Gal, R. Islam, and Z. Ghahramani. "Deep Bayesian active learning with image data." In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, pp. 1183–1192, Sydney, NSW, Australia, August 2017. JMLR.org.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and intractability*, volume 174. freeman San Francisco, 1979.

[GJ02] M. R. Garey and D. S. Johnson. *Computers and intractability*, volume 29. W.H. Freeman, New York, 2002.

[GS02] R. Giugno and D. Shasha. "Graphgrep: A fast and universal method for querying graphs." In *2002 International Conference on Pattern Recognition*, volume 2, pp. 112–115. IEEE, 2002.

[GSS15] I. Goodfellow, J. Shlens, and C. Szegedy. "Explaining and Harnessing Adversarial Examples." In *International Conference on Learning Representations*, 2015.

[Gur12] Gurobi Optimization, Inc. *Gurobi Optimizer Reference Manual, Version 5.0*, 2012.

[Gur18] L. Gurobi Optimization. "Gurobi optimizer reference manual.", 2018.

[GVS14] I. J. Goodfellow, O. Vinyals, and A. M. Saxe. "Qualitatively characterizing neural network optimization problems." *arXiv preprint arXiv:1412.6544*, 2014.

[GYB23] Y. Ge, D. Yang, and A. L. Bertozzi. "Iterative Active Learning Strategies for Subgraph Matching." 2023.

[HDM14] M. Houbraken, S. Demeyer, T. Michoel, P. Audenaert, D. Colle, and M. Pickavet. "The Index-based Subgraph Matching Algorithm with General Symmetries (ISMAGS): exploiting symmetry for faster subgraph enumeration." *PloS one*, **9**(5):e97896, 2014.

[HHG11] N. Houlsby, F. Huszár, Z. Ghahramani, and M. Lengyel. "Bayesian Active Learning for Classification and Preference Learning." *arXiv:1112.5745 [cs, stat]*, December 2011. arXiv: 1112.5745.

[HKG19] M. Han, H. Kim, G. Gu, K. Park, and W.-S. Han. "Efficient Subgraph Matching: Harmonizing Dynamic Programming, Adaptive Matching Order, and Failing Set Together." In *Proceedings of the 2019 International Conference on Management of Data*, pp. 1429–1446. ACM, 2019.

[HLL13] W.-S. Han, J. Lee, and J. hoon Lee. "Turboiso: towards ultrafast and robust subgraph isomorphism search in large graph databases." In *SIGMOD Conference*, 2013.

[HS06] H. He and A. K. Singh. "Closure-tree: An index structure for graph queries." In *22nd International Conference on Data Engineering (ICDE'06)*, pp. 38–38. IEEE, 2006.

[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.

[HZR16] K. He, X. Zhang, S. Ren, and J. Sun. "Deep residual learning for image recognition." In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[IBM09] IBM Corp. *IBM Ilog CPLEX V12.1: User's Manual for CPLEX*, 2009.

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

[JG19] H. Jiang and M. Gupta. "Minimum-Margin Active Learning." *arXiv:1906.00025 [cs, stat]*, May 2019. arXiv: 1906.00025.

[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 (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.

[JWP06] H. Jiang, H. Wang, S. Y. Philip, and S. Zhou. "Gstring: A novel approach for efficient search in graph databases." In *2007 IEEE 23rd International Conference on Data Engineering*, pp. 566–575. IEEE, 2006.

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

[Kar72] R. M. Karp. "Reducibility among combinatorial problems." In *Complexity of computer computations*, pp. 85–103. Springer, 1972.

[KBD17] G. Katz, C. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer. "Reluplex: An efficient SMT solver for verifying deep neural networks." In *International Conference on Computer Aided Verification*, pp. 97–117. Springer, 2017.

[KGB18] A. Kurakin, I. J. Goodfellow, and S. Bengio. "Adversarial examples in the physical world." In *Artificial intelligence safety and security*, pp. 99–112. Chapman and Hall/CRC, 2018.

[KGD18] E. B. Khalil, A. Gupta, and B. Dilkina. "Combinatorial Attacks on Binarized Neural Networks." In *International Conference on Learning Representations*, 2018.

[KKM11] K. Klein, N. Kriege, and P. Mutzel. "Ct-index: Fingerprint-based graph indexing combining cycles and trees." In *2011 IEEE 27th International Conference on Data Engineering*, pp. 1115–1126. IEEE, 2011.

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

[KP18] M. Kivelä and M. A. Porter. "Isomorphisms in Multilayer Networks." *IEEE Transactions on Network Science and Engineering*, **5**(3):198–211, 2018.

[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 (Big Data)*, pp. 3959–3967, 10 2018.

[KSH12] A. Krizhevsky, I. Sutskever, and G. E. Hinton. "Imagenet classification with deep convolutional neural networks." *Advances in neural information processing systems*, **25**:1097–1105, 2012.

[KX19] A. Kopylov and J. Xu. "Filtering Strategies for Inexact Subgraph Matching on Noisy Multiplex Networks." In *2019 IEEE International Conference on Big Data (Big Data)*, pp. 4906–4912, 2019.

[LCH06] C. Liu, C. Chen, J. Han, and P. S. Yu. "GPLAG: detection of software plagiarism by program dependence graph analysis." In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 872–881, 2006.

[LDT19] L. Liu, B. Du, H. Tong, et al. "G-Finder: Approximate Attributed Subgraph Matching." In *2019 IEEE International Conference on Big Data (Big Data)*, pp. 513–522. IEEE, 2019.

[LeC98] Y. LeCun. "The MNIST database of handwritten digits." 1998.

[Ley03] S. Leyffer. "Mathematical Programs with Complementarity Constraints." *SIAG/OPT Views-and-News*, **14**(1):15–18, 2003.

[LLN06] S. Leyffer, G. Lopez-Calva, and J. Nocedal. "Interior Methods for Mathematical Programs with Complementarity Constraints." *SIAM Journal on Optimization*, **17**(1):52–77, 2006.

[LMB17] M. Lombardi, M. Milano, and A. Bartolini. "Empirical decision model learning." *Artificial Intelligence*, **244**:343–367, 2017.

[LRS18] P. Laborie, J. Rogerie, P. Shaw, and P. Vilím. "IBM ILOG CP optimizer for scheduling: 20+ years of scheduling with constraints at IBM/ILOG." *Constraints*, **23**:210–250, 2018.

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

[LY17] Y. Li and Y. Yuan. "Convergence Analysis of Two-layer Neural Networks with ReLU Activation." *Advances in Neural Information Processing Systems*, **30**:597–607, 2017.

[Mac77] A. K. Mackworth. "Consistency in networks of relations." *Artificial intelligence*, **8**(1):99–118, 1977.

[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. IEEE, 2018.

[MF12] A. Murray and B. Franke. "Compiling for automatically generated instruction set extensions." In *Proceedings of the Tenth International Symposium on Code Generation and Optimization*, pp. 13–22, 2012.

[MGF18] G. Micale, R. Giugno, A. Ferro, M. Mongiovi, D. Shasha, and A. Pulvirenti. "Fast analytical methods for finding significant labeled graph motifs." *Data Mining and Knowledge Discovery*, **32**(2):504–531, 2018.

[MGT17] E. Malmi, A. Gionis, and E. Terzi. "Active network alignment: a matching-based approach." In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pp. 1687–1696, 2017.

[MLL11] A. Mahajan, S. Leyffer, J. Linderoth, J. Luedtke, and T. Munson. "MINOTAUR: A toolkit for solving mixed-integer nonlinear optimization." wiki-page, 2011.

[MM19] M. Maggioni and J. M. Murphy. "Learning by active nonlinear diffusion." *Foundations of Data Science*, **1**(3):271, 2019. Company: Foundations of Data Science Distributor: Foundations of Data Science Institution: Foundations of Data Science Label: Foundations of Data Science Publisher: American Institute of Mathematical Sciences.

[Moo21] J. Moorman. *Stochastic Optimization and Subgraph Search.* University of California, Los Angeles, 2021.

[MP14] B. D. McKay and A. Piperno. "Practical graph isomorphism, II." *J. of Symbolic Computation*, **60**:94–112, 1 2014.

[MP15] C. McCreesh and P. Prosser. "A parallel, backjumping subgraph isomorphism algorithm using supplemental graphs." In *International conference on principles and practice of constraint programming*, pp. 295–312. Springer, 2015.

[MPC14] G. F. Montufar, R. Pascanu, K. Cho, and Y. Bengio. "On the Number of Linear Regions of Deep Neural Networks." *Advances in Neural Information Processing Systems*, **27**:2924–2932, 2014.

[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**:723–759, 2018.

[MPT20] C. McCreesh, P. Prosser, and J. Trimble. "The Glasgow subgraph solver: using constraint programming to tackle hard subgraph isomorphism problem variants." In *International Conference on Graph Transformation*, pp. 316–324. Springer, 2020.

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

[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 (Big Data)*, pp. 4913–4920. IEEE, 2019.

[PGC17] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. "Automatic differentiation in PyTorch." 2017.

[PMB14] R. Pascanu, G. Montúfar, and Y. Bengio. "On the number of response regions of deep feed forward networks with piece-wise linear activations." In *International Conference on Learning Representations*, 2014.

[Pow69] M. Powell. "A method for nonlinear constraints in minimization problems in Optimization." In R. Fletcher, editor, *Optimization*. Academic Press, 1969.

[PPL20] H. G. Patsolic, Y. Park, V. Lyzinski, and C. E. Priebe. "Vertex nomination via seeded graph matching." *Statistical Analysis and Data Mining: The ASA Data Science Journal*, **13**(3):229–244, 2020.

[PTA21] T. Papalexopoulos, C. Tjandraatmadja, R. Anderson, J. P. Vielma, and D. Belanger. "Constrained Discrete Black-Box Optimization using Mixed-Integer Programming." *arXiv preprint arXiv:2110.09569*, 2021.

[QHS05] N. V. Queipo, R. T. Haftka, W. Shyy, T. Goel, R. Vaidyanathan, and P. K. Tucker. "Surrogate-based analysis and optimization." *Progress in aerospace sciences*, **41**(1):1–28, 2005.

[RB05] A. Raghunathan and L. T. Biegler. "An Interior Point Method for Mathematical Programs with Complementarity Constraints (MPCCs)." *SIAM Journal on Optimization*, **15**(3):720–750, 2005.

[RCA19] M. Ryu, Y. Chow, R. Anderson, C. Tjandraatmadja, and C. Boutilier. "CAQL: Continuous Action Q-Learning." In *International Conference on Learning Representations*, 2019.

[Reg94] J.-C. Régin. "A filtering algorithm for constraints of difference in CSPs." In *AAAI*, volume 94, pp. 362–367, 1994.

[RS14] P. Ribeiro and F. Silva. "Discovering colored network motifs." In *Complex Networks V*, pp. 107–118. Springer, 2014.

[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.

[RZL17] P. Ramachandran, B. Zoph, and Q. V. Le. "Searching for activation functions." *arXiv preprint arXiv:1710.05941*, 2017.

[Sah96] N. V. Sahinidis. "BARON: A general purpose global optimization software package." *Journal of Global Optimization*, **8**(2):201–205, 1996.

[SC15] F. Serratosa and X. Cortés. "Interactive graph-matching using active query strategies." *Pattern Recognition*, **48**(4):1364–1373, 2015.

[SDD15] C. Solnon, G. Damiand, C. De La Higuera, and J.-C. Janodet. "On the complexity of submap isomorphism and maximum common submap problems." *Pattern Recognition*, **48**(2):302–316, 2015.

[Set12] B. Settles. *Active Learning*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool, 2012.

[SF83] A. Sanfeliu and K.-S. Fu. "A distance measure between attributed relational graphs for pattern recognition." *IEEE transactions on systems, man, and cybernetics*, (3):353–362, 1983.

[SM19] A. M. Schweidtmann and A. Mitsos. "Deterministic Global Optimization with Artificial Neural Networks Embedded." *Journal of Optimization Theory and Applications*, **180**(3):925–948, 2019.

[Sol10] C. Solnon. "AllDifferent-based Filtering for Subgraph Isomorphism." *Artificial Intell.*, **174**:850–864, August 2010.

[Sol19] C. Solnon. "Experimental evaluation of subgraph isomorphism solvers." In *International Workshop on Graph-Based Representations in Pattern Recognition*, pp. 1–13. Springer, 2019.

[SPP19] 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*, pp. 1–1, 2019.

[SR20] T. Serra and S. Ramalingam. "Empirical Bounds on Linear Regions of Deep Rectifier Networks." In *AAAI*, pp. 5628–5635, 2020.

[SS00] H. Scheel and S. Scholtes. "Mathematical Program with Complementarity Constraints: Stationarity, Optimality and Sensitivity." *Mathematics of Operations Research*, **25**:1–22, 2000.

[SS04] C. Schulte and P. J. Stuckey. "Speeding up constraint propagation." In *Principles and Practice of Constraint Programming–CP 2004: 10th International Conference, CP 2004, Toronto, Canada, September 27-October 1, 2004. Proceedings 10*, pp. 619–633. Springer, 2004.

[SZ14] K. Simonyan and A. Zisserman. "Very deep convolutional networks for large-scale image recognition." In *International Conference on Learning Representations*, 2014.

[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.

[SZS14] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. "Intriguing properties of neural networks." In *2nd International Conference on Learning Representations*, 2014.

[TK01] S. Tong and D. Koller. "Support Vector Machine Active Learning with Applications to Text Classification." *Journal of Machine Learning Research*, **2**(Nov):45–66, 2001.

[TKT21] C. Tsay, J. Kronqvist, A. Thebelt, and R. Misener. "Partition-based formulations for mixed-integer optimization of trained relu neural networks." *Advances in Neural Information Processing Systems*, **34**, 2021.

[TMY20] T. K. Tu, J. D. Moorman, D. Yang, Q. Chen, and A. L. Bertozzi. "Inexact attributed subgraph matching." *Proc. IEEE Cong. BIG DATA, Graph Techniques for Adversarial Activity Analytics (GTA3 4.0) workshop*, pp. 2575–2582, 2020.

[TS02] M. Tawarmalani and N. V. Sahinidis. *Convexification and Global Optimization in Continuous and Mixed-Integer Nonlinear Programming: Theory, Algorithms, Software, and Applications*. Kluwer Academic Publishers, Boston MA, 2002.

[TXT17] V. Tjeng, K. Xiao, and R. Tedrake. "Evaluating robustness of neural networks with mixed integer programming." *arXiv preprint arXiv:1711.07356*, 2017.

[Ukk92] E. Ukkonen. "Approximate string-matching with q-grams and maximal matches." *Theoretical computer science*, **92**(1):191–211, 1992.

[Ull76] J. R. Ullmann. "An Algorithm for Subgraph Isomorphism." *J. ACM*, **23**(1):31–42, January 1976.

[Val79] L. G. Valiant. "The complexity of computing the permanent." *Theoretical computer science*, **8**(2):189–201, 1979.

[VCL15] J. T. Vogelstein, J. M. Conroy, V. Lyzinski, L. J. Podrazik, S. G. Kratzer, E. T. Harley, D. E. Fishkind, R. J. Vogelstein, and C. E. Priebe. "Fast approximate quadratic programming for graph matching." *PLOS one*, **10**(4):e0121002, 2015.

[WB06] A. Wächter and L. T. Biegler. "On the Implementation of a Primal-Dual Interior Point Filter Line Search Algorithm for Large-Scale Nonlinear Programming." *Mathematical programming*, **106**(1):25–57, 2006.

[WDT12] X. Wang, X. Ding, A. K. Tung, S. Ying, and H. Jin. "An efficient graph indexing method." In *2012 IEEE 28th International Conference on Data Engineering*, pp. 210–221. IEEE, 2012.

[WKK97] L. Wiskott, N. Krüger, N. Kuiger, and C. von der Malsburg. "Face recognition by elastic bunch graph matching." *IEEE Trans. on Patt. Anal. and Mach. Int.*, **19**(7):775–779, 1997.

[WLC20] L. L. Wang, K. Lo, Y. Chandrasekhar, R. Reas, J. Yang, D. Burdick, D. Eide, K. Funk, Y. Katsis, R. M. Kinney, et al. "CORD-19: The COVID-19 Open Research Dataset." In *Proceedings of the 1st Workshop on NLP for COVID-19 at ACL 2020*, 2020.

[WLW21] Q. Wang, M. Li, X. Wang, N. Parulian, G. Han, J. Ma, J. Tu, Y. Lin, R. H. Zhang, W. Liu, et al. "COVID-19 Literature Knowledge Graph Construction and Drug Repurposing Report Generation." In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies: Demonstrations*, pp. 66–77, 2021.

[WSM14] J.-P. Watson, D. R. Strip, W. C. McLendon, O. D. Parekh, C. F. Diegert, S. B. Martin, and M. D. Rintoul. "Encoding and analyzing aerial imagery using geospatial semantic graphs." Technical report, Sandia National Lab.(SNL-NM), Albuquerque, NM (United States), 2014.

[WWY10] G. Wang, B. Wang, X. Yang, and G. Yu. "Efficiently indexing large sparse graphs for similarity search." *IEEE Transactions on Knowledge and Data Engineering*, **24**(3):440–451, 2010.

[YBL22] D. Yang, P. Balaprakash, and S. Leyffer. "Modeling design and control problems involving neural network surrogates." *Computational Optimization and Applications*, pp. 1–42, 2022.

[YGN23] D. Yang, Y. Ge, T. Nguyen, D. Molitor, J. D. Moorman, and A. L. Bertozzi. "Structural Equivalence in Subgraph Matching." *IEEE Transactions on Network Science and Engineering*, 2023.

[YM13] D. Yuan and P. Mitra. "Lindex: a lattice-based index for graph databases." *The VLDB Journal*, **22**:229–252, 2013.

[YYH04] X. Yan, P. S. Yu, and J. Han. "Graph indexing: a frequent structure-based approach." In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pp. 335–346, 2004.

[Zas75] T. Zaslavsky. *Facing up to arrangements: Face-count formulas for partitions of space by hyperplanes: Face-count formulas for partitions of space by hyperplanes*, volume 154. American Mathematical Soc., 1975.

[ZB17] Z. Zhang and M. Brand. "Convergent Block Coordinate Descent for Training Tikhonov Regularized Deep Neural Networks.", 2017.

[ZDS10] S. Zampelli, Y. Deville, and C. Solnon. "Solving subgraph isomorphism problems with constraint programming." *Constraints*, **15**:327–353, 07 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.

[ZHY06] S. Zhang, M. Hu, and J. Yang. "Treepi: A novel graph indexing method." In *2007 IEEE 23rd International Conference on Data Engineering*, pp. 966–975. IEEE, 2006.

[ZLG03] X. Zhu, J. Lafferty, and Z. Ghahramani. "Combining Active Learning and Semi-Supervised Learning Using Gaussian Fields and Harmonic Functions." In *ICML 2003 workshop on The Continuum from Labeled to Unlabeled Data in Machine Learning and Data Mining*, pp. 58–65, 2003.

[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.

[ZPM21] J. Zucker, K. Paneri, S. Mohammad-Taheri, S. Bhargava, P. Kolambkar, C. Bakker, J. Teuton, C. Hoyt, K. Oxford, R. Ness, and O. Vitek. "Leveraging Structured Biological Knowledge for Counterfactual Inference: A Case Study of Viral Pathogenesis." *IEEE Trans. on Big Data*, **7**(01):25–37, 1 2021.

[ZXL12] X. Zhao, C. Xiao, X. Lin, and W. Wang. "Efficient graph similarity joins with edit distance constraints." In *2012 IEEE 28th International Conference on Data Engineering*, pp. 834–845. IEEE, 2012.

[ZYY07] P. Zhao, J. X. Yu, and P. S. Yu. "Graph indexing: tree+ delta¡= graph." In *Proceedings of the 33rd international conference on Very large data bases*, pp. 938–949, 2007.