Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

Enhancing GNNs: An Exploration of Iterative Solving and Augmentation Techniques

Bachelor's Thesis

Andy Tran

andtran@ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

Supervisors:
Joël Mathys, Florian Grötschla
Prof. Dr. Roger Wattenhofer

August 24, 2023

Acknowledgements

I would like to express my gratitude to my supervisors Joël Mathys and Florian Grötschla for their consistent guidance and support throughout this project. Their feedback was invaluable in shaping this thesis. I also want to thank Prof. Dr. Roger Wattenhofer and the Distributed Computing Group, for giving me the opportunity to work in such a stimulating environment. Lastly, a thank you to my peers and family for their encouragement and support during this period.

Abstract

In this thesis, we seek to enhance the predictive accuracy of Graph Neural Networks (GNNs) through adapted inference, driven by the motivation that certain problems, due to restricted computational power, are challenging to solve in a one-shot approach. Recognizing that for algorithmic problems or logic puzzles, solutions can be found by breaking up the problem and solving the subproblems (a method similiar to human reasoning), we explore this path. Our approach leverages hints during the training phase, a technique effective even when hints are absent during inference, and applies them to a spectrum of problems including Maximum and Maximal Independent Sets, and complex puzzles like Sudoku and Kakuro (both NP-Complete). To facilitate the handling of partially solved problems that might not naturally occur during training, we integrate data preprocessing techniques. The sequential problem-solving method employed achieves substantial improvements in performance, utilizing GNNs to discern patterns and connections within graphs. The applications extend to solving Sudoku and Kakuro puzzles, showcasing the adaptability and far-reaching implications of our techniques.

Contents

Chapter 1

Introduction

The rapid advancement of Graph Neural Networks (GNNs) has brought fresh perspectives to many complex problems, such as those found in network design, routing optimization, and intricate logic puzzles. [Zhou et al., 2020]. Traditional algorithms have often found these problems to be intractable, due to some falling into NP-Complete [YATO and SETA, 2003]. In complicated tasks, humans naturally gravitate towards iterative methods, breaking down the problem into smaller, manageable parts [Newell et al., 1972] which is what we would like to mimic.

In human problem-solving, breaking down complex tasks into smaller stages is a natural and often necessary strategy. This step-by-step approach enables us to navigate challenges by solving intermediate, simpler problems [Elman, 1993]. However, Graph Neural Networks (GNNs) do not always have the luxury of access to these intermediate stages during training. In puzzles like Sudoku and other intricate challenges, the solutions at some stages may never be encountered during the training process. This lack of access to partial or intermediate solutions can hinder the model's learning and generalization abilities. Augmentation comes into play here, as a method to bridge this gap. By supplementing the original training data with additional information or 'hints', such as partially solved problems, the augmentation process transforms the learning approach into a more semi-supervised one.

In the context of Graph Neural Networks (GNNs), iterative solving offers a compelling advantage. By predicting one node at a time and integrating this information with the original data, the model can continuously refine its predictions, zeroing in on yet-to-be-solved nodes. This is not merely a replication of conventional one-shot methods but a significant enhancement, allowing for the inclusion of intermediate hints as constructive feedback.

In this thesis, we delve into essential questions regarding the integration of augmentation and iterative solving with Graph Neural Networks (GNNs). How does augmentation enhance training and testing? In what ways can iterative solving contribute to increasing accuracy? How do these techniques compare to traditional one-shot approaches within GNNs? Furthermore, how do these

methodologies stand in comparison to other neural network architectures like Recurrent Neural Networks (RNNs) and Convolutional Neural Networks (CNNs)? The exploration of these questions aims to not only increase our understanding of these processes but also to identify potential advancements and optimizations within the field of machine learning.

Chapter 2

Methodical Approach

Augmentation

In graph-based learning, the dataset may be either unsupervised or semi-supervised, lacking sufficient information to represent various intermediate states. This can include scenarios such as partially completed puzzles, almost fully completed puzzles, or, in the context of node classifications in graphs, all possible states ranging from incomplete to nearly fully-labeled cases. To navigate these challenges and transition a unsupervised dataset into a semi-supervised dataset, our approach employs an augmentation strategy designed to fill these gaps.

The first step involves recognizing the various intermediate states or degrees of completion that the problem can exhibit. This encompasses scenarios such as partially completed puzzles, nearly fully completed puzzles, or, in the case of node classifications in graphs, all possible states ranging from incomplete to nearly fully labeled cases.

The label augmentation process selectively augments labels for each node, creating new augmented graphs that are included alongside the original, non-augmented graphs. This inclusion ensures that the Graph Neural Network (GNN) is capable of making predictions not only for augmented graphs but also for original data points. By representing all possible states a GNN might encounter, the augmentation process not only transitions unsupervised datasets into a semi-supervised learning context but also enriches the learning experience by exposing the model to a complete spectrum of states. The comprehensive nature of this approach enhances the model's ability to generalize and adapt, reflecting a more realistic range of problem-solving scenarios.

Iterative Solving

In this section, we introduce an iterative solving approach for node classification problems within a graph-based learning framework. The process is formulated

through the following steps:

Given an input graph $G = (V, E)$, the trained model predicts the labels for all nodes, formulated as:

$$\hat{Y} = \text{Model}(G), \tag{2.1}$$

where $\hat{Y} \in \mathbb{R}^{|V| \times c}$ represents the predicted labels for all nodes in V, and c is the number of classes.

Among the predicted labels, the node with the highest certainty for its end label is selected:

$$i = \arg\max_{i} \text{certainty}(\hat{Y}_i), \tag{2.2}$$

where i denotes the index of the node with the highest certainty.

The graph and node attributes is subsequently updated to include the predicted label for the selected node:

$$G' = \text{update}(G, i, \hat{Y}_i). \tag{2.3}$$

The process outlined above is exexcuted iteratively until all nodes are predicted.

The iterative solving approach in graph-based learning allows the model to adapt to complex relationships, building upon nodes with a high degree of certainty.

Chapter 3

Preliminaries

Many phenomena in our world can be effectively represented as graphs. In these graphs, nodes and edges encode various types of information. For instance, social media platforms can be visualized as graphs with users as nodes and their interactions or connections as edges. Similarly, molecular structures, transportation networks, the World Wide Web, power grids, and collaboration networks all lend themselves naturally to graph representation.

At the core of our interest is understanding the inherent properties of these models, especially the characteristics of individual nodes. This leads us to the task of 'node classification', wherein we aim to determine specific attributes of nodes. Additionally, there's 'graph classification', a broader task that focuses on discerning the overarching attributes of an entire graph.

Historically, classical algorithms were employed to address these challenges, where the objectives were well-defined. However, in recent years, a new paradigm has emerged: Graph Neural Networks (GNNs). In GNNs, data is processed through neural networks using a series of message-passing layers. Our research objective is to set benchmark performances for GNNs and evaluate their capabilities. We are also keen on investigating whether tweaking our datasets can lead to enhanced performance.

3.1 Models

3.1.1 RecGNN

In the realm of Graph Neural Networks (GNNs), scalability and adaptability are paramount. As GNNs inherently operate at the node level by exchanging messages with neighboring nodes, they possess the potential to be trained on smaller graphs and then be seamlessly applied to considerably larger ones. However, the innate flexibility of GNNs, which allows them to be trained for a predefined number of message-passing rounds, can sometimes be a limiting factor. This is particularly evident in scenarios where information must be propagated across

the entire graph, a task which cannot always be achieved within a fixed number of rounds.

To tackle this challenge, a model, termed RecGNN, was introduced [Grötschla et al., 2022]. The essence of RecGNN lies in its recurrent architecture, which empowers it to learn graph algorithms end-to-end. This design choice means that while training is conducted on smaller graphs, during inference, the model can judiciously adapt the number of convolutions and run for more rounds on larger graphs.

Modification to RecGNN

In the development of the RecGNN model, its primary architecture is optimized for node classifications, focusing exclusively on node attributes. However, in many real-world graph scenarios, edge attributes play a significant role in determining the graph's structure and features. To enhance the utility and versatility of the RecGNN model, we have integrated edge attributes into the message-passing step.

The original formula for the RecGRU-E convolution is:

$$h_v^{t+1} = \text{GRU}\left(\sum_{w \in N(v)} \Theta(h_v^t || h_w^t), h_v^t\right) \tag{3.1}$$

Where:

- h_v^{t+1} represents the hidden state of node v at the next time step.
- $N(v)$ denotes the neighboring nodes of node v.
- h_v^t and h_w^t are the hidden states of nodes v and w at the current time step t.
- Θ is a weight matrix.
- $||$ denotes concatenation.

To integrate edge attributes into the RecGNN model's convolution process, we propose a modified formula:

$$h_v^{t+1} = \text{GRU}\left(\sum_{w \in N(v)} \Theta(h_v^t || h_w^t + \text{MLP}(e_{vw})), h_v^t\right) \tag{3.2}$$

Where:

- e_{vw} represents the edge attribute between nodes v and w.
- MLP is the transformation function that projects the edge attributes into a representation compatible with the node hidden states.

This formula allows the model to consider both node and edge attributes during the message passing. With this integration, the RecGNN can harness the intricacies of edge data, enriching its understanding and predictions on the graph.

3.1.2 Convolutional Neural Network (CNN)

Many real-world problems exhibit patterns that can be captured using a grid-like structure, especially when spatial information is crucial. Convolutional Neural Networks (CNNs) serve as the ideal architecture to exploit such spatial structures. CNNs function by systematically scanning the input data through a series of convolutional layers using a small, overlapping window termed as a kernel. Through this method, CNNs can adaptively learn spatial hierarchies of features from images, starting from basic patterns like edges and corners in initial layers to more complex ones in deeper layers.

The primary advantage of CNNs lies in their capability to retain and process spatial information. This inherent ability makes them indispensable for problems where spatial context is pivotal. By conserving the relative positions of data, CNNs can discern patterns fundamentally rooted in spatial configurations.

For the specific problems under consideration, our CNN model is defined as follows:

$$L = \text{Total layers (7 in our case)} \tag{3.3}$$
$$K = \text{Kernel size for each layer (3x3)} \tag{3.4}$$
$$C = \text{Number of channels produced by each convolution (512)} \tag{3.5}$$
$$P = \text{Padding, which duplicates edge information to preserve input size} \tag{3.6}$$

The model's architecture can be represented as:

$$\text{CNN}(L, K, C, P) = \bigoplus_{i=1}^{L} \text{Conv2D}(K, C) + \text{BatchNorm}() + \text{ReLU}() \tag{3.7}$$

Where:

- $\bigoplus$ denotes the sequential combination of layers.

- Conv2D represents the 2-dimensional convolutional layer.
- BatchNorm signifies the batch normalization layer, instrumental in stabilizing and expediting the training.
- ReLU is the activation function employed.

3.1.3 Recurrent Neural Network (RNN)

Sequential data, like the series of numbers in a Sudoku puzzle, often contain dependencies across time or sequence positions. To effectively learn and represent these dependencies, Recurrent Neural Networks (RNNs) are employed. Unlike conventional neural networks which assume independent inputs, RNNs maintain a memory, encapsulated in their hidden state, which captures information about previous parts of the sequence. This makes them particularly apt for tasks with temporally correlated data or where the problem's structure involves a series of interlinked decisions.

In our experiments involving Sudoku puzzles, RNNs were chosen as a baseline due to their inherent nature of retaining memory and state. This choice reflects our approach of utilizing intermediate outputs as feedback, thereby continuously refining the predictions based on historical data. Specifically, the method mirrors human intuition, where one might iterate through a Sudoku puzzle multiple times, making corrections and filling in numbers based on previous observations.

We settled on using GRU (Gated Recurrent Units) over the traditional RNN structures because of its efficiency. GRUs are designed to combat the vanishing gradient problem in RNNs without increasing computational demands as much as its counterpart, the LSTM (Long Short Term Memory). Empirical studies have further demonstrated that the performance of GRUs is often on par with LSTMs, despite their relative simplicity and smaller model size.

For our problems, the RNN model specifics are:

$$H = \text{Hidden Dimension (128)} \tag{3.8}$$

$$L = \text{Number of layers (3)} \tag{3.9}$$

$$B = \text{Bidirectional GRU} \tag{3.10}$$

Chapter 4

Related Works

Graph Neural Networks (GNNs) were first introduced to facilitate learning on graphs, enabling the edges to propagate information Scarselli et al. [2008]. This innovation opened the door for graph-structured data to be processed and learned from using neural network architectures, bridging the gap between traditional machine learning and graph theory. Since its inception, there has been an extensive evolution in the architectures designed to operate over graphs.

Recurrent Neural Networks (RNNs), for instance, were foundational in processing sequence-based tasks, which paved the way for more advanced variations such as Long Short-Term Memory (LSTM) networks Hochreiter and Schmidhuber [1997]. They manage longer sequences by introducing additional gates to retain or forget information selectively. While our main focus is not on RNNs, understanding their mechanisms is beneficial because of their conceptual overlap with certain graph-based processes, especially when considering the temporal dimension in graph signals.

GNNs' design also borrows insights from Convolutional Neural Networks (CNNs) LeCun et al. [2015]. CNNs, recognized for their prowess in handling spatial hierarchies in image data, introduced techniques like padding to preserve spatial dimensions between layers, which find analogous applications in graph data.

Normalization techniques such as Layer Normalization Ba et al. [2016] and Batch Normalization Ioffe and Szegedy [2015] were introduced to stabilize and expedite training in deep networks. Their adaptability to graph-based architectures proves instrumental in achieving consistent training across different graph sizes and topologies.

An important reference in graph algorithms that has influenced GNN architectures is the Weisfeiler-Lehman Test of Isomorphism Leman and Weisfeiler [1968]. It has inspired GNN variants that are capable of distinguishing non-isomorphic structures and understanding intricate node and edge relationships in graphs.

Previous works on solving Sudokus has been done [Palm et al., 2018], where they introduced a Recurrent Relational Network (RRN) model specifically tai-

lored for solving complex problems requiring interdependent relational inferences. They applied the RRN to the task of solving Sudoku puzzles, representing the puzzle grid as a graph where each cell was a node and relationships between cells were captured as edges. The RRN applied iterative updates to the nodes, processing both local and global constraints inherent in the puzzle.

Next to that work has been done on solving Kakuros by [Daniec, 2020], where the model from [Palm et al., 2018] has been adapted to solve Kakuro puzzles, one of the models we want to adopt is augmenting our dataset to also solve it iteratively.

In this thesis, we are primarily focused on employing a convolution that integrates Gated Recurrent Units (GRUs) Li et al. [2015]. GRUs, an offshoot of RNNs, retain the ability to handle sequential data but with a more efficient gating mechanism, making them particularly apt for certain graph-related tasks. The choice of integrating them with GNNs is to leverage their ability to model and remember intricate relationships in graph data, aiding in tasks like the ones we explore, where understanding deep structural nuances and iterative reasoning becomes vital.

Chapter 5

Independent Set Problems

The Independent Set Problem (ISP) seeks to identify the largest subset S of vertices in a graph $G = (V, E)$ such that no two vertices in S are adjacent, i.e., $\forall u, v \in S, (u, v) \notin E$. Within the scope of ISPs, two primary categories can be recognized:

- **Maximal Independent Set:** A set S of vertices is considered maximal if no additional vertex can be added to S without violating the independent set condition. Formally, S is maximal if $\forall v \in V \setminus S, \exists u \in S$ such that $(u, v) \in E$.
- **Maximum Weighted Independent Set:** Each vertex v has an associated weight $w(v)$. The goal here is to find an independent set S maximizing the total weight, i.e., the sum of the weights of its vertices. A special case of this problem is the **Maximum Independent Set**, where each vertex has the same weight. This denotes the independent set that is of the largest size in a graph. If $|S|$ denotes the size of set S, then for any other independent set S' in G, $|S| \geq |S'|$.

For our investigations, we target Tree Graphs and Erdos-Renyi Graphs when considering Maximal Independent Set problems. For challenges related to Maximum (Weighted) Independent Set, our attention is solely on trees, with both weighted and unweighted versions being examined.

5.1 Maximal Independent Set (MIS)

A maximal independent set (MIS) represents a set of nodes that, once chosen, ensures that no other nodes adjacent to them can be selected. This process is continued until no more nodes can be selected, and the nodes selected so far form the MIS. Given the intrinsic non-uniqueness of the MIS, pinpointing a unique solution is not feasible. For our purposes, we are keen on identifying a starting set of nodes that give rise to a unique MIS. Even though the problem appears local, with an obvious restriction that two neighboring nodes cannot simultaneously be part of the MIS, understanding its nature sets the stage for tackling more complex problems later in our research.

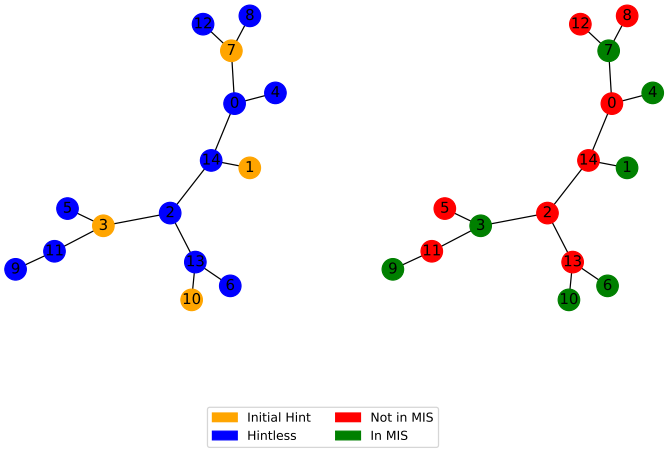


Fig. 5.1: Example of a Maximal Independent Set, on the left side initial input and on the right side the target

5.1.1 Dataset

Our dataset generation follows a specific procedure. Initially, an MIS is formed, after which nodes are progressively removed. At each step, we ensure that the nodes left still dictate a single maximal independent set. Our training focus is largely on understanding the base performance of a GNN, especially its extrapolation strengths as discussed in 'Learning Graph Algorithms With Recurrent Graph Neural Networks' Grötschla et al. [2022]. To this end, we perform training on 10-node graphs, and subsequently extrapolate these models to tackle larger graphs. This approach aids in gauging proficiency of the network with simpler tasks and observing if it can adapt and evolve for more challenging tasks. This approach is taken for the two datasets we compare, namely MIS on trees and on random graphs (Erdos-Renyi)

5.1.2 Results

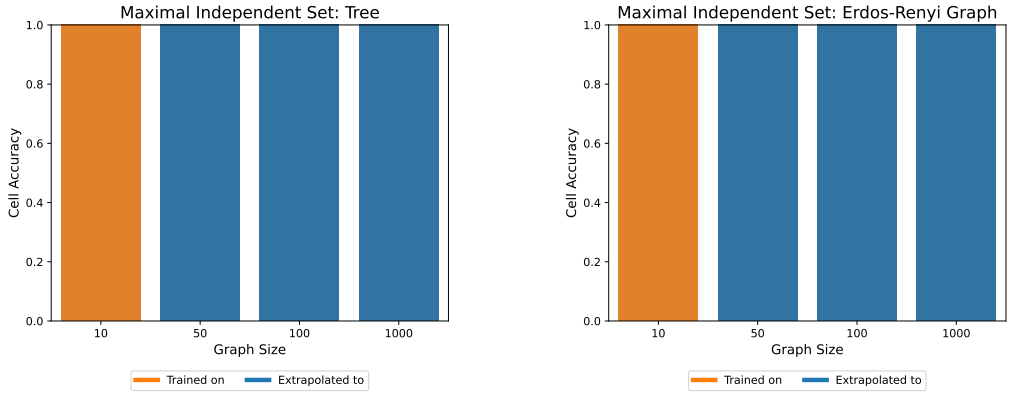


Fig. 5.2: Result for MIS of training on graphs with 10 nodes, and extrapolating to bigger graphs, with an increased amount of layers for bigger trees and graphs.

From Figure 5.2, it is evident that GNNs demonstrate a significant aptitude for solving the Maximal Independent Set problem. One potential reason for this effectiveness is the inherent locality of the problem. The MIS problem fundamentally revolves around understanding local neighborhoods and ensuring that adjacent nodes are not simultaneously part of the set. This property aligns well with the capabilities of GNNs, which excel at capturing and processing local structures and relationships within graphs.

Our training predominantly centered on 10-node graphs, which may lead one to believe that the GNN's strong performance is restricted to these simpler structures. However, this is not the case. When we extrapolated the models to graphs with a larger number of nodes, not only did they adapt, but they also consistently achieved perfect scores. This demonstrates the GNN's robustness and ability to generalize, not just memorize, across different graph sizes.

5.2 Maximum Weighted Independent Set (MumIS)

In this section, we delve into the challenge of determining the Maximum Weighted Independent Set (MumIS) within trees. By definition, the maximum independent set entails a collection of nodes in a graph where no two nodes share an edge, and the size of this collection is maximized. As a result this task is a node classification task. Our primary goal is to analyze how GNNs perform when tasked with classifying tree nodes based on their participation in the maximum independent set.

A brief recap of the formula for the Maximum Weighted Independent Set is as follows:

Given a graph $G = (V, E)$ with a weight function $w : V \to \mathbb{R}^+$ that assigns a positive weight to each vertex, the Maximum Weighted Independent Set is:

$$\text{MumIS}(G) = \underset{S \subseteq V, \forall u,v \in S, (u,v) \notin E}{\arg\max} \sum_{v \in S} w(v)$$

5.2.1 Dataset

Given the complexity and often infeasibility of isolating maximum (weighted) independent sets in generic graphs, we have chosen to focus our efforts on trees. Trees provide a more straightforward landscape for pinpointing the maximum weighted independent set. As in the previous section, we are interested in determining the extrapolation strength of the GNN.

Our computational strategy hinges on determining two values for every subtree within the graph:

- $A(i)$: Denotes the size of the maximum independent set in the subtree rooted at i, with the stipulation that node i is part of the set.
- $B(i)$: Represents the size of the maximum independent set in the subtree rooted at i, but under the constraint that node i is excluded from the set.

These values are recursively derived through the contemplation of two scenarios:

1. Exclusion of the subtree's root:

$$B(i) = \sum_{j \in \text{children}(i)} \max(A(j), B(j)) \tag{5.1}$$

2. Inclusion of the subtree's root:

$$A(i) = w(i) + \sum_{j \in \text{children}(i)} B(j) \tag{5.2}$$

The metric for the maximum independent set across the entire tree is derived from the larger of $A(\text{root})$ and $B(\text{root})$. To pinpoint the exact set, it is also necessary to record the maximum independent set for every induced subtree and subsequently integrate these results.

For the unweighted case of the Maximum Independent Set, we adopt a strategy where we assign a uniform weight to every node, ensuring each node is treated with equal significance.

5.2.2 Results

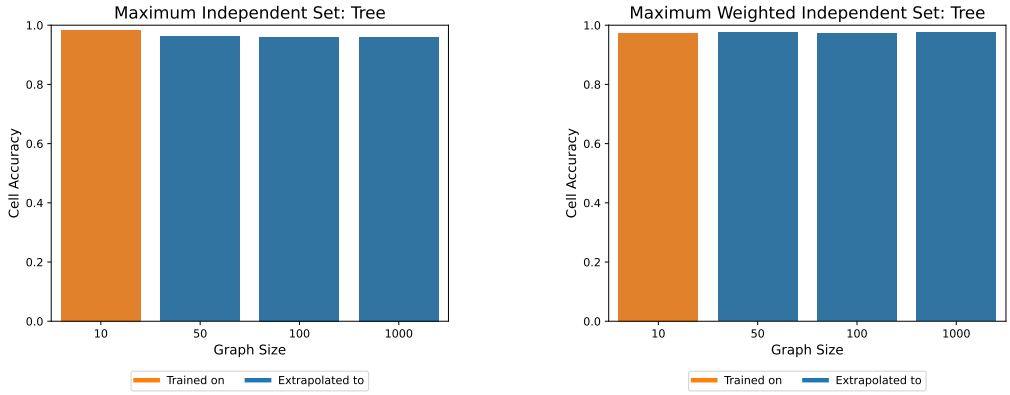


Fig. 5.3: Result for Maximum (Weighted) Independent Set of training on trees with 10 nodes, and extrapolating to bigger graphs, with an increased amount of layers for bigger trees.

The presented graphs in Figure 5.3 reveal a nuanced contrast to the results observed in the Maximal Independent Set problem. While the trained models demonstrate strong performance, they do not achieve perfection. When considering the trees, where all the nodes of the maximum (weighted) independent set have to be be identified, the models fall short, failing to completely solve any instances of the Maximum Weighted Independent Set problem. Several factors may contribute to this difference.

Firstly, unlike the Maximal Independent Set problem, there is an absence of an initial set of hints to guide the model. This means that the GNN has the dual challenge of inferring relationships and understanding the task without any preliminary hints. Additionally, the inherent uniqueness of the problem makes it harder.

Moreover, this problem places a premium on global information, requiring the model to consider broader relationships and patterns rather than solely focusing on localized node interactions. Despite these challenges, the models display impressive performance. Even in scenarios without direct hints or cues, the GNN demonstrates its robustness and versatility, achieving an accuracy of approximately 96% in the most challenging cases.

In short, even though this problem is slightly tougher for the GNN, with some more adjustments and focused training, we can likely get closer to perfect results.

5.2.3 Training Augmentation & Iterative Solving

We explored the potential of boosting the GNN's performance by introducing hints during its training. By enhancing our dataset with two additional features for certain nodes, chosen at random, the GNN is exposed to varying degrees of hint information. These features essentially guide the GNN about a node's definite presence or absence in the final Maximum Independent Set. With this approach, we aim for the GNN to better discern the interdependencies and apply this knowledge when it encounters datasets without hints.

Having exposed the GNN to different hint intensities, our next focus is on its capability to iteratively classify nodes. This exploration not only gauges potential performance improvements but also lays groundwork for tackling more intricate problems in the future.

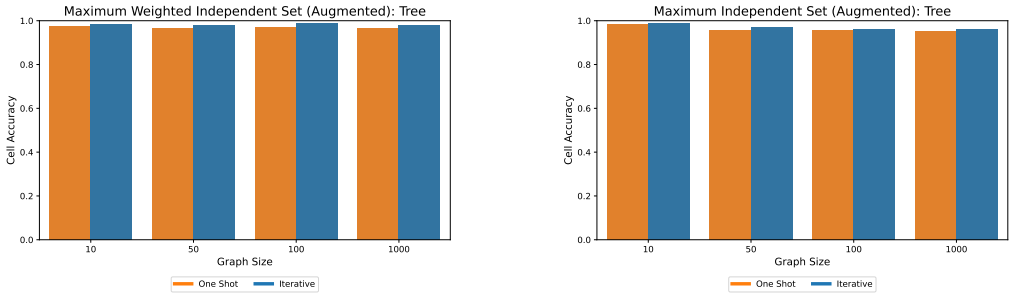


Fig. 5.4: Extrapolation Performance of Augmented Dataset: These graphs show the node accuracy when iteratively solving with a constant amount of layers, specifically trained on 12 layers and tested on 15 layers.

Figure 5.4 paints a clear picture: introducing hints to the dataset and iteratively solving enables the GNN to achieve superior outcomes. The rationale behind using augmentation was to streamline the GNN's learning curve, ensuring it grasps underlying patterns with increased efficiency. From our results, it is evident that the hints sharpen the GNN's decision-making process. In wrapping up, both the dataset augmentation and iterative node classification methods have demonstrated their potential and might pave the way for future enhancements.

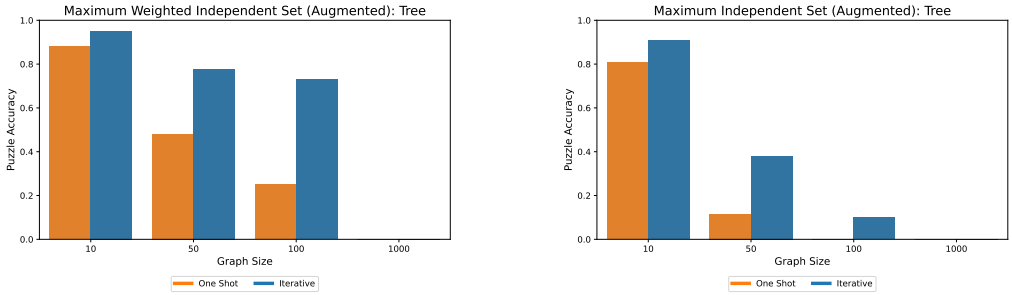


Fig. 5.5: Extrapolation Performance of Augmented Dataset: These graphs show the complete graph accuracy when iteratively solving with a constant amount of layers, specifically trained on 12 layers and tested on 15 layers.

In Figure 5.5 we can clearly see iterative solving also leads to higher complete accuracy, helping us to achieve better results.

Chapter 6

Reachability Problems

In this chapter, we are interested in researching GNN's capability of distinguishing Strongly Connected Components (SCCs) and identifying Strongly Connected Nodes (SCNs) in the case of directed graphs. Strongly Connected Nodes (SCNs) are nodes in a graph that have a directed path to every other node in the graph.

Strongly Connected Components (SCCs) A Strongly Connected Component (SCC) in a directed graph is a maximal subgraph such that for every pair of vertices u and v in the subgraph, there exists a directed path from u to v and from v to u.

$$\forall u, v \in \text{SCC},\ \exists \text{path}(u, v) \wedge \exists \text{path}(v, u)$$

Strongly Connected Nodes (SCNs) A Strongly Connected Node (SCN) is a node that has a directed path to every other node in the graph.

$$\forall u \in \text{Nodes},\ \exists v \in \text{SCN},\ \exists \text{path}(v, u)$$

The goal of this chapter is to identify a GNN's strength and also its global message passing capabilities.

6.1 Strongly Connected Components

6.1.1 Dataset

In the study of Strongly Connected Components (SCCs), we generate directed graphs to create maximal subgraphs where directed paths exist between every pair of vertices. This exploration provides insights into global dependencies and connectivity patterns within complex networks.

Training is carried out on graphs of size 15 to evaluate the extrapolation capabilities of the modified RecGNN model with edge attributes, for potentially harder or larger problems.

6.1.2 Results

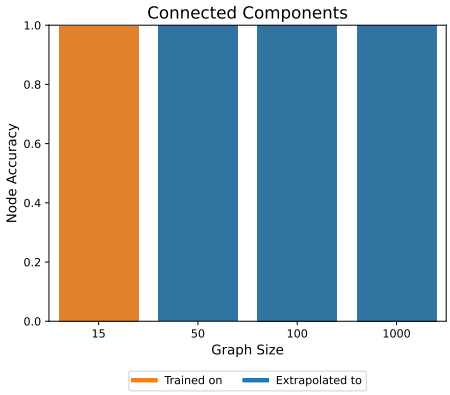


Fig. 6.1: GNN performance (node accuracy) of assigning nodes to the correct Strongyly Connected Components

In Figure 6.1, the results demonstrate that the GNN is capable of perfectly distinguishing which nodes belong to specific Strongly Connected Components (SCCs). This sophisticated classification suggests that the GNN can recognize and categorize nodes based on their features and relationships within the graph, accurately determining to which component each node belongs.

6.2 Strongly Connected Nodes

6.2.1 Dataset

Investigating Strongly Connected Nodes (SCNs) extends our understanding of network structures, especially in problems that require identifying key elements within a system. Our SCNs are encoded with labeled bidirectional edges, providing a connection to every other node in the graph, and to allow for backward propagation of information.

Training is done on graphs of size 15, again using the modified RecGNN model with edge attributes, to explore the model's ability to handle more complex or larger-scale problems.

6.2.2 Results

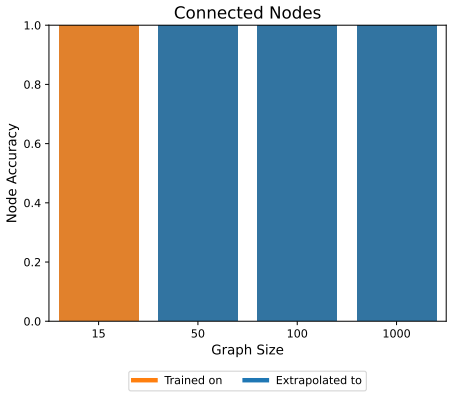


Fig. 6.2: GNN performance (node accuracy) of distinguishing which nodes are able to reach every other node in the graph by some path

As illustrated in Figure 6.2, the GNN achieves perfect accuracy in classifying Strongly Connected Nodes (SCNs). This remarkable result indicates the model's ability to identify the specific nodes that can reach every other node within the graph. Understanding which nodes possess this influential property is valuable, as it reveals the GNN's capacity to discern the importance of individual nodes and to recognize which information is essential to propagate through the network.

6.3 Final Conclusion

In the context of SCCs, the ability to recognize and categorize nodes based on their features and connections aligns with the underlying principles of pattern recognition in Sudoku and connectivity in Kakuro, explored in other chapters of this thesis.

Similarly, the identification of SCNs could inform more refined strategies for solving Sudoku and Kakuro by emphasizing the importance of certain nodes and their connections. By understanding these key elements, GNNs may enhance information flow, offering new insights into the solution process for these and other interconnected problems.

Together, these findings reinforce the versatility and potential of GNNs in various problem domains and contribute to a broader understanding of their applicability within the scope of this thesis.

Chapter 7

Solving Sudoku Puzzles

Sudoku is a logic-based, combinatorial number-placement puzzle that has gained immense popularity worldwide. The objective of the Sudoku puzzle is to fill a partially completed 9x9 grid with digits in such a way that each row, each column, and each of the nine 3x3 subgrids (also called boxes) contains all of the digits from 1 to 9 exactly once. The puzzle has a very simple set of rules, which makes it easy to understand but often challenging to solve.

The Sudoku puzzle typically starts with a grid that has a certain number of cells pre-filled with digits. These pre-filled cells are known as *clues*. The number of hints provided can vary, and the difficulty of the puzzle often depends on the number and distribution of the given clues. The general rule is that a Sudoku puzzle should have only one unique solution.

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

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

Fig. 7.1: Example of a Sudoku Puzzle, with the puzzle on the left side and solution on the right side

7.1 Dataset

We possess a collection of unique Sudoku puzzles and their corresponding solutions. Since the difficulty of a Sudoku is mostly dependent on the amount of clues we get, it is possible to divide the Sudoku puzzles into three different categories based on the amount of clues.

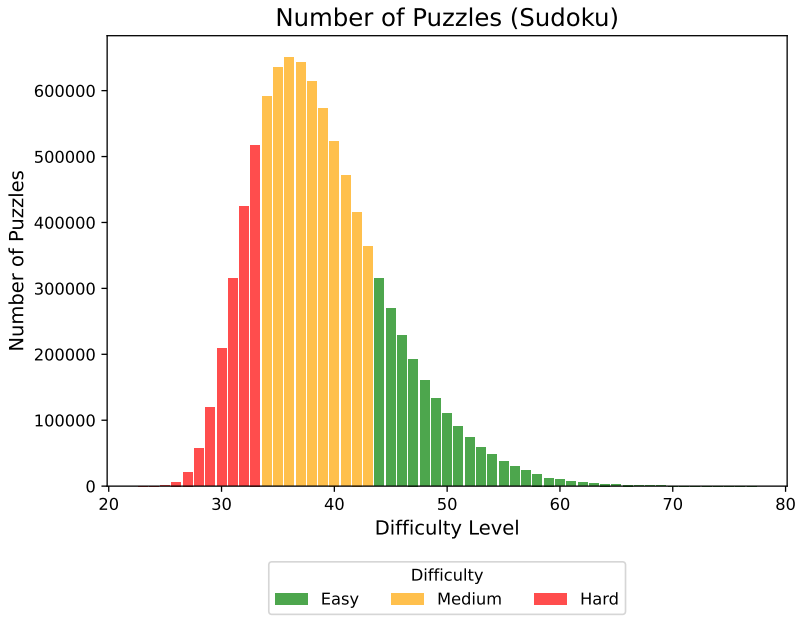


Fig. 7.2: Puzzle Count by Clue Count

The distribution of our puzzles can be seen in Figure 7.2. Due to the dataset containing over 1 million puzzles, it is not feasible to train any model on this size, for that reason, we reduce all the training sets to 200,000 puzzles of each difficulty, unless noted otherwise.

The goal is making as few mistakes as possible when solving each individual puzzle, but also secondly to be able to solve puzzles correctly. For most models it is relatively easy to determine the end value of pre-filled cells, for this reason we are mostly interested in the correctly predicted unfilled cells, which we will denote by cell accuracy, and the total puzzle accuracy, i.e. how many puzzles it is able to solve completely.

7.2 Solving Sudoku using RNNs

In our exploration of more traditional models, Recurrent Neural Networks (RNNs) emerge as fundamental contenders. RNNs are specifically designed to recognize

patterns in sequences of data, making them suitable for sequential tasks and problems with temporal dynamics. For our Sudoku-solving architecture, we harness the power of Gated Recurrent Units (GRUs). The output of the GRU is then processed through a linear layer to generate predictions for each cell in the Sudoku grid. We opted for RNNs primarily because of their inherent capability to maintain memory from previous inputs, which can be pivotal when considering the interdependencies of Sudoku cells.

7.2.1 Training & Results

Figure 7.3 sheds light on the RNN model's capabilities. For simpler puzzles, the RNN showcases commendable accuracy, identifying numerous cells correctly. However, as the complexity scales up, the model's performance plateaus, with the accuracy hovering around 60% for the most challenging puzzles. It is noteworthy that models trained on advanced puzzles tend to outperform in simpler scenarios, whereas the inverse does not hold true. An intriguing observation is the analogous performance between models trained on mixed datasets and those trained solely on the medium dataset. This might stem from the fact that the mixed dataset primarily mirrors the general distribution where medium-difficulty puzzles dominate. Even though one might assume that exposure to a diverse set of puzzles would improve performance, the mixed-trained model's outcomes align more with the medium-trained counterpart.

7.2.2 Iterative Solving

The human way of solving Sudoku is identical to the iterative solving method we introduced earlier, where we fill in a cell one at a time. In this method, the cell filled at each step is the one where the model is the most certain, i.e., the cell associated with the highest probability. The model iteratively makes predictions for the puzzle filled with the new cells, until each cell has a prediction in it, after which the model outputs this results.

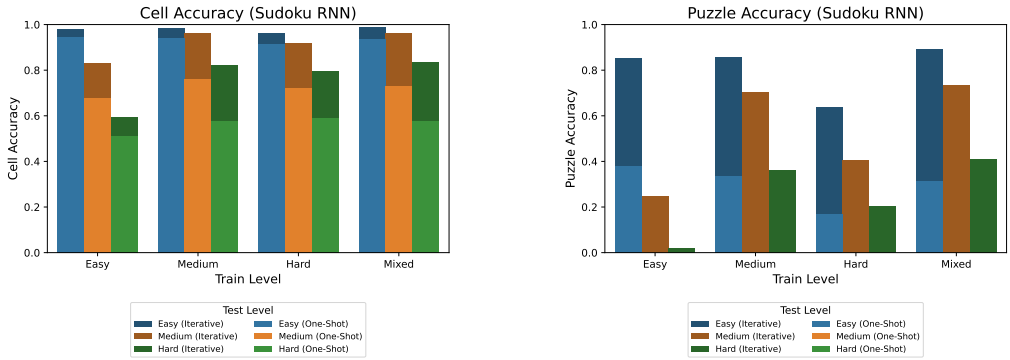


Fig. 7.3: Performance of RNN model on Sudoku, one-shot vs iterative, where the improvement of the iterative variant can be seen stacked on top of the standard one-shot model

In our testing we found out that the iterative solving leads to an increase in performance in predicting the class for each cell, but also in the final puzzle accuracy.

7.3 Solving Sudoku using CNNs

Sudoku is a puzzle inherently reliant on spatial positioning, where each cell's value is influenced by its location. We leverage this characteristic to devise a solution using a Convolutional Neural Network (CNN).

Every Sudoku puzzle is transcribed into a 9x9 input with 10 distinct channels. Each channel operates as a one-hot representation of the initial cell value.

7.3.1 Training & Results

From the visualized results in Figure 7.4, it becomes evident that CNNs exhibit a consistent trend in their performance across various Sudoku difficulties. Each model excels in the easy category and maintains this proficiency when applied to its corresponding training dataset. However, a discernible drop in performance is observed when these models grapple with puzzles of higher difficulty.

Surprisingly, the performance of the model trained on a mixed dataset is akin to that of the model trained solely on the easy dataset. Considering that the mixed dataset predominantly consists of medium puzzles, this outcome suggests the model may be exhibiting a middle-ground performance, where it is not fully optimized for the extremities of difficulty.

Given the spatial structure inherent to Sudoku grids, it is no surprise that CNNs, known for capturing local patterns in data, manage to resolve a substantial

portion of the puzzles effectively.

7.3.2 Iterative Solving

Iterative solving is an approach that breaks down complex problems into more manageable steps. By revisiting the puzzle multiple times, the model has the opportunity to refine its predictions, relying on the accumulated knowledge from previous iterations. This method mirrors the way many humans approach Sudoku, iteratively filling in numbers as the relationships between cells become clearer with each pass.

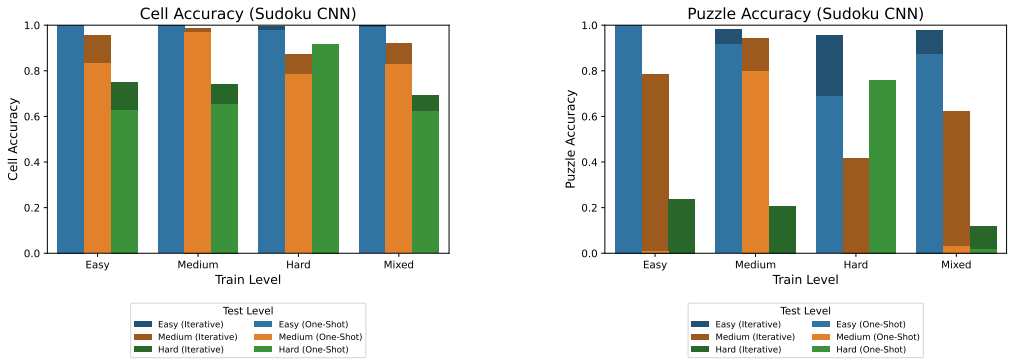


Fig. 7.4: Performance of CNN model on Sudoku, one-shot vs iterative, where the improvement of the iterative variant can be seen stacked on top of the standard one-shot model

After employing iterative solving, a noticeable enhancement in the CNN's performance is observed as can be seen in Figure 7.4. Especially significant is the rise in accuracy for medium puzzles, particularly for models initially trained on the easy and hard datasets. This progression suggests that iterative solving not only improves the model's overall accuracy but also widens its capability range, enabling it to tackle puzzles of varied complexities more effectively.

7.4 Solving Sudoku using GNNs

During our testing phase, we explored various Graph Neural Network (GNN) architectures. One key observation was that GNNs using Graph Isomorphism Networks (GIN) failed to converge. Even when we coupled GIN with a Multilayer Perceptron (MLP), convergence was not achieved. This failure could be attributed to GIN's inability to retain memory and state, a shortcoming that Gated Recurrent Units (GRUs) can address. In our experiments, GRUs in combination with an MLP performed better than without MLP.

We also compared the use of GRU and LSTM (Long Short-Term Memory) units. Our findings indicated that GRU offered faster training times while delivering similar performance to LSTM. When we created a model using LSTMs instead of GRU, the performance was found to be inferior. This could be because LSTMs have more parameters, potentially hindering the GNN's ability to learn optimal weights. It has been reported in [Cahuantzi et al., 2021] that GRUs can outperform LSTMs for certain problems. Therefore, our primary focus remained on the standard RecGNN model, with some modifications.

One experimental approach was loop unrolling, where instead of reusing the same trained convolution for multiple layers, we created multiple convolutions for one-time use. Unfortunately, this method failed to lead to convergence, possibly due to the increased number of parameters that needed to be learned. Consequently, we concluded that reusing the same convolution layers for multiple rounds was the most effective approach.

7.4.1 Encoding

Grid-Style

The grid-style encoding takes inspiration from the way Convolutional Neural Networks (CNNs) propagate information. In this method, we treat each cell in the Sudoku puzzle as a node and connect it to its immediate horizontal and vertical neighbors. This approach has the benefit of simplicity, mirroring the physical layout of the Sudoku grid. It enables the model to capture local dependencies and relationships within the puzzle, much like how humans often approach solving Sudoku by considering immediate neighboring cells.

In the grid-style encoding the row and column are also encoded, to further enrich the dataset with more information.

However, this grid-style approach also has its limitations. By focusing solely on direct neighbors, the model might miss more complex global patterns that exist within the puzzle. Furthermore, it might not fully capture all the constraints and rules of Sudoku, particularly those that involve non-adjacent cells in the same row, column, or subgrid. This can make the encoding less expressive and potentially limit the model's ability to solve more complex puzzles.

Standard (Implicit) Style

The standard implicit style of encoding takes a different approach, aiming to embed the fundamental rules of Sudoku directly into the model. In this method, each cell is treated as a node, and connections are established with all other nodes in the same horizontal and vertical directions, as well as those in the same subgrid.

This encoding method is more comprehensive, capturing a richer set of information and more resembling the constraints of Sudoku. It considers relationships not just with neighboring cells, but with all cells that share the same row, column, or subgrid. This gives the model a more nuanced understanding of the puzzle and the flexibility to adapt to various structures.

However, this method also has its drawbacks. The increase in connections and the larger graph size can make this approach more complex and computationally challenging. Additionally, the added complexity might cause the model to overfit to specific training data patterns, potentially reducing its ability to generalize to unseen puzzles.

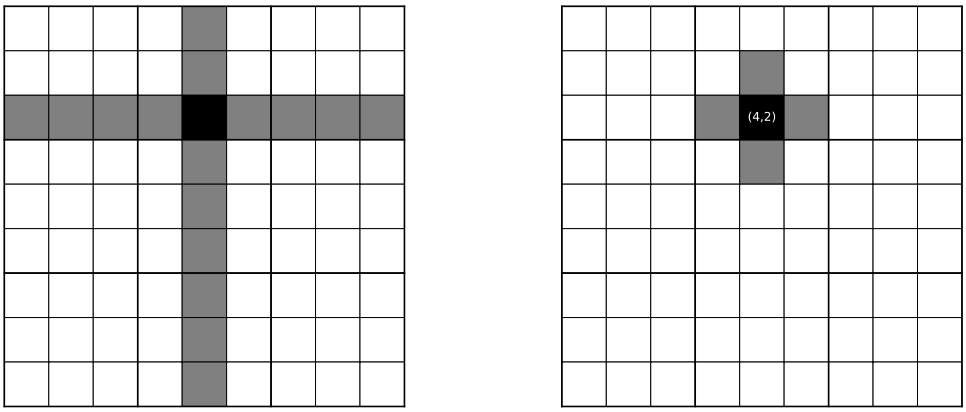


Fig. 7.5: Representations of the two different encodings for a cell at (4,2), on the left side the Standard encoding and right side the grid-style encoding. Black denotes cell at (4,2), with grey the cells it shares an edge with. In the Grid-Style encoding, it additionally has the (column, row) encoded as one of its features. The cell value is also a node feature, but this is omitted in the graphics.

7.4.2 Training & Results

Firstly we would like to analyze the performance for difference encodings. We can see in Figure 7.6 we achieve higher cell accuracy for the Standard encoding, where cells are connected to cells in the same row, column, or subgrid. This is not surprising, due to the encoding implicitly enforcing the rules of Sudoku and propagating cell values to its directly related cells. What we can also see, is that the models trained on Standard Encoding is able to solve some puzzles to a certain degree, while the models trained on Grid Encoding failed to do so.

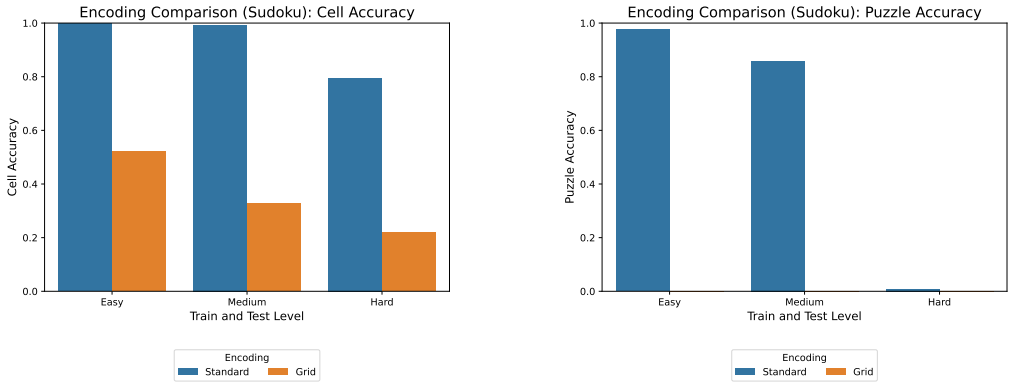


Fig. 7.6: Comparison of performance for different encodings, on the left side the cell accuracy and right side puzzle accuracy

We only include Figure 7.7, due to the Grid-Style encoding achieving lackluster performance in comparison to the standard style. Due to that we omit, and from here on forward we only analyze the Standard Encoding.

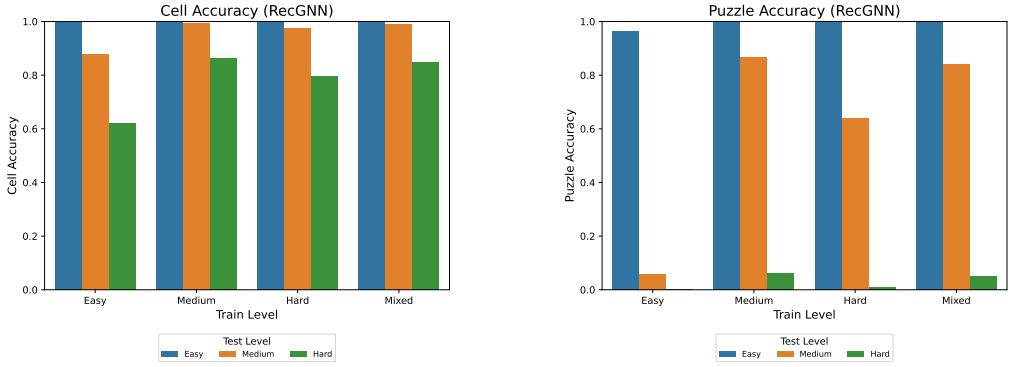


Fig. 7.7: Performance of RecGNN models on Sudoku trained on different difficulties, with the Standard Encoding

Our analysis reveals that all models achieve impressive performance, all reaching perfect accuracy on puzzles categorized under the easy difficulty level. An unexpected observation is that the model trained on medium puzzles outperforms the one trained on hard puzzles when faced with the latter. This discrepancy may stem from the hard training dataset's complexity, leading the model to struggle in discerning dependencies and rules efficiently. This outcome suggests that augmenting the dataset with different puzzle varieties and complexities could enhance the models' performance on more difficult puzzles.

Increase of Layers

Although graph sizes remain constant in the context of Sudoku, the influence of varying the number of layers on performance is still of interest. In the standard RecGNN model as highlighted by [Grötschla et al., 2022], the number of layers increases with the number of nodes. A logical question that emerges is how many times to reuse the same trained convolution. However, since the number of nodes in Sudoku is fixed, we cannot dynamically determine the number of layers based on the node count. Our testing demonstrated that more layers generally led to better performance. Interestingly, we also found that training on fewer layers led to faster training times but still allowed for improved performance when using more layers during inference. This finding offered an encouraging balance between training efficiency and predictive capability.

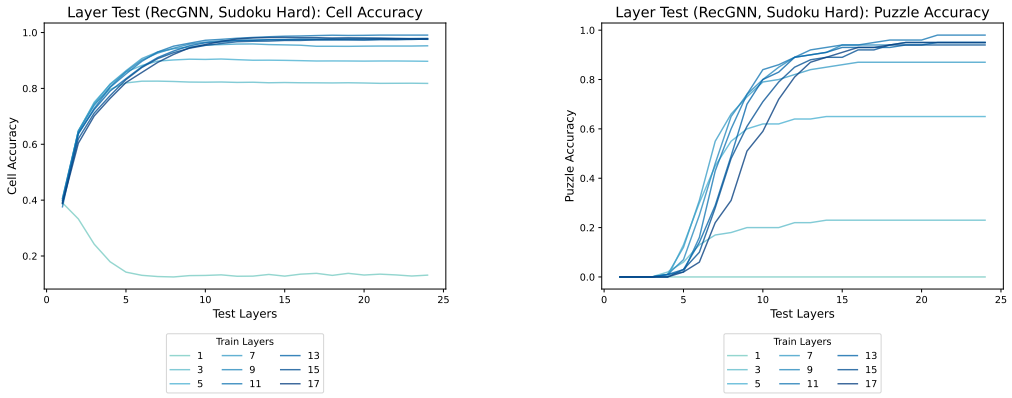


Fig. 7.8: Performance of training on a different amount of layers and testing on a different amount of layers, with cell and puzzle accuracy

As illustrated in Figure 7.8, there is a clear pattern showing that most models enjoy enhanced accuracy when tested with an increased number of layers. This trend is particularly prominent in models trained on at least 9 layers, where accuracy continually improves, unlike other models trained on fewer layers that exhibit a noticeable threshold in both cell and puzzle accuracy. The observation suggests an opportunity to optimize the training process by training on 10 layers and extrapolating to 15 layers. This demonstrates the model's ability to learn how the recurrent application of layers can yield superior results, something that seems less pronounced in models operating with a reduced number of layers.

7.4.3 Augmentation & Iterative Solving

As we have seen before, augmenting datasets is a common practice to improve model robustness and performance. While this technique might seem unnec-

essary for Sudoku—given that puzzles are always defined with certain clues or hints—our experimentation sought to determine if augmentation could indeed make a difference in model performance.

To test this, we augmented our dataset by incorporating puzzles in varying completion stages, from wholly unsolved to nearly solved. This expanded dataset design aligns more closely with how humans often approach Sudoku, progressing through iterative solving stages. Additionally, this approach allows for more realistic iterative solving within the model, mirroring human problem-solving behavior.

The augmentation was implemented by adding two modified puzzles for each original puzzle, thus tripling the dataset size. However, we ensured to test on the non-augmented dataset to gauge any performance improvements accurately.

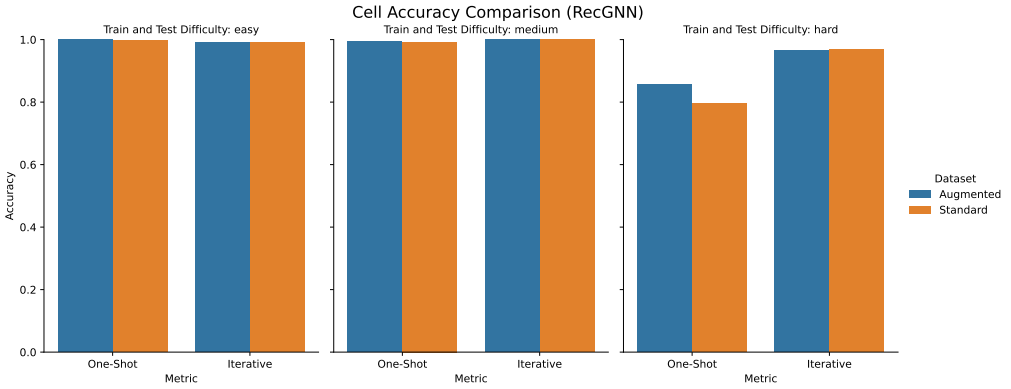


Fig. 7.9: Comparison of GNN models trained on augmented and non-augmented datasets (Cell Accuracy)

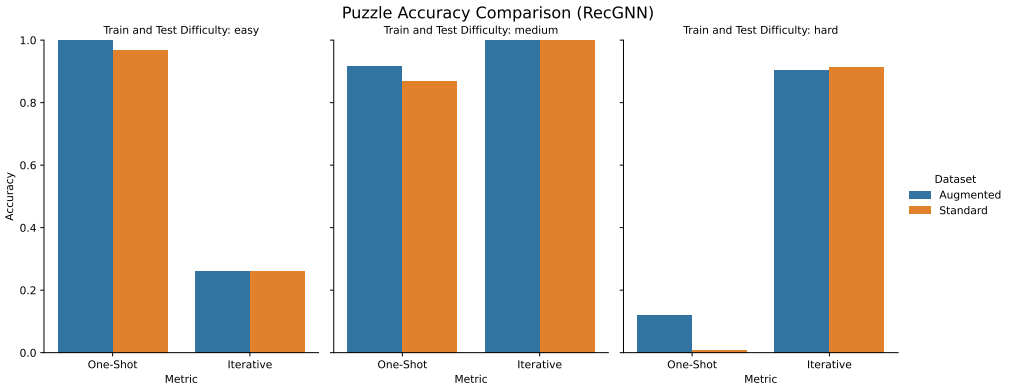


Fig. 7.10: Comparison of GNN models trained on augmented and non-augmented datasets (Puzzle Accuracy)

Our findings present a fascinating insight into the role of augmentation in solving Sudoku puzzles. It is evident that augmentation does indeed enhance performance in most cases when using the One-Shot approach. However, the impact becomes less distinguishable when employing an Iterative approach, where various models exhibit similar behavior. Interestingly, models trained and tested on easy puzzles often fall short in achieving high puzzle accuracy while managing to attain a significant cell accuracy, as illustrated in Figure 7.10. This discrepancy highlights that iterative approaches can lead to more precise and consistent results for Sudoku, aligning them better with both the performance and predictive behavior observed in CNN and RNN models. The iterative methodology enables the model to fine-tune and recalibrate its predictions, adapting to the intricacies of the puzzle.

Example

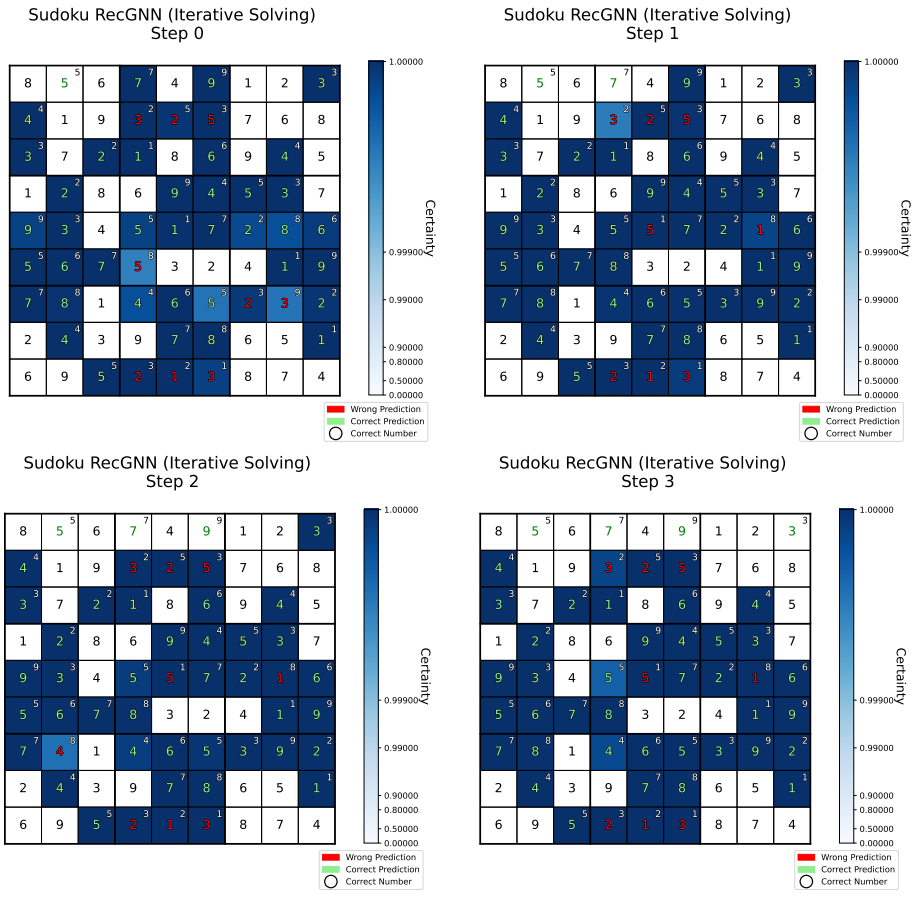


Fig. 7.11: First 4 steps of iteratively solving a Sudoku puzzle with a GNN, initially if we were to mispredict we would have a few cell mispredictions

As depicted in Figure 7.11, the GNN demonstrates an initial confidence in its predictions, selecting the numbers with the highest certainty. As the iterations progress, the model fine-tunes its predictions, resulting in fewer errors. This gradual refinement is particularly evident in Figure 7.12, where the GNN eventually solves the Sudoku puzzle. This iterative process of focusing on the most certain parts and progressively improving predictions mimics a more human-like approach to solving such puzzles.

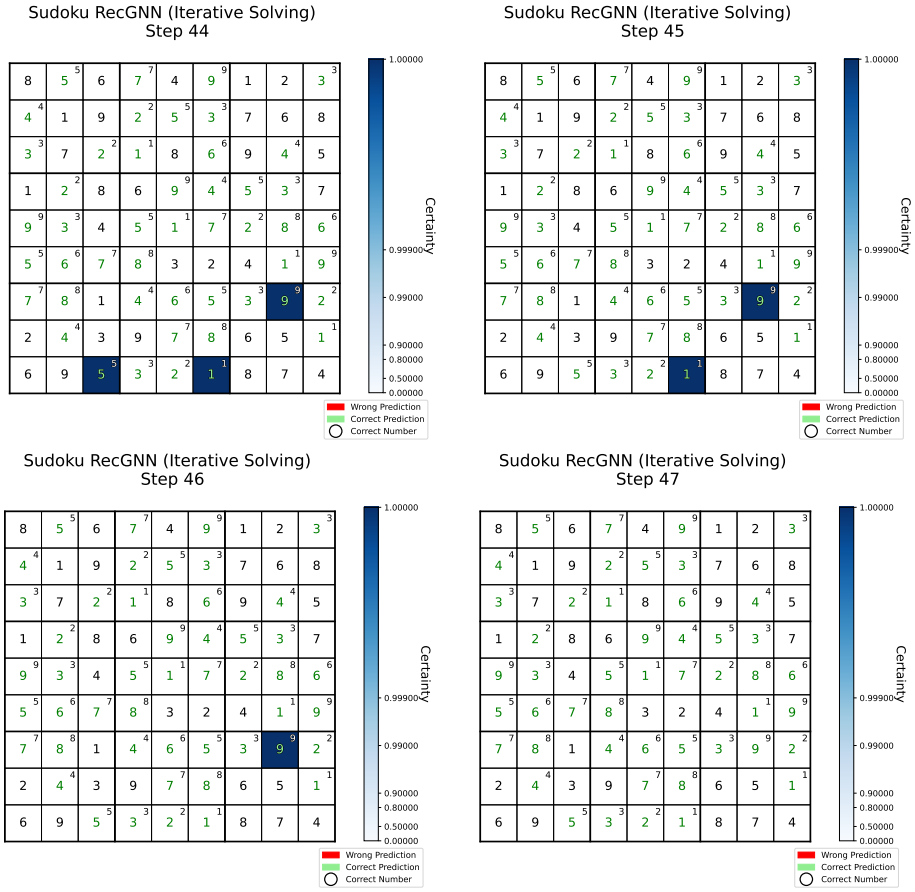


Fig. 7.12: Last 4 steps iteratively solving a Sudoku puzzle with a GNN, correctly solved in the end

Permutation

An intriguing augmentation approach is achieved through permuting the dataset. Since the numerical values in Sudoku are essentially symbolic placeholders, a bijective reassignment of values will not alter the puzzle's unique solvability. This property unlocks up to 9! permutations for each puzzle, providing an immense expansion to our dataset.

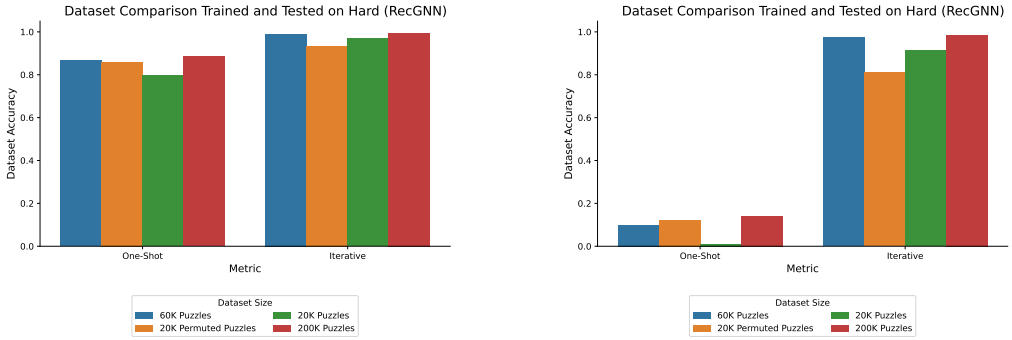


Fig. 7.13: Comparison of RecGNN on different datasets for hard puzzles

As illustrated in Figure 7.13, a general trend emerges that larger training sets tend to yield better results, assuming all other variables remain consistent. An unexpected finding, however, is the performance discrepancy between the model trained on 60K puzzles and the one trained solely on 20K puzzles when compared to permuted puzzles. Specifically, in the iterative context, these models trained on the permuted datasets perform noticeably worse, whereas in the one-shot context, they either outperform or exhibit similar performance to the permuted puzzles.

The observation that models trained on more puzzles consistently perform better is not surprising in itself, but the subtle nuances in performance, especially with permutation, deserve further exploration. For example, the practice of permuting puzzles during training, ensuring that the model encounters only new challenges, could offer significant benefits. This strategy, particularly when coupled with techniques such as augmentation or iterative solving. The relative simplicity of permutation and its potential benefits make it an appealing area for further investigation in our pursuit of improved efficacy.

7.5 Final Conclusion

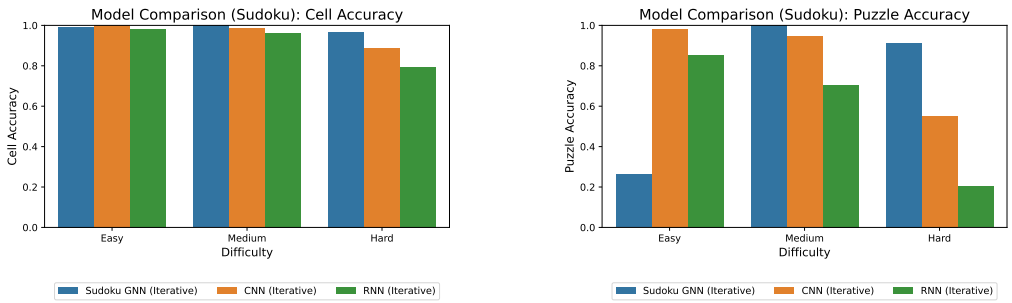


Fig. 7.14: Performance comparison of all the best models for Sudoku

Sudoku presents unique challenges for computational problem-solving. Our research involving both GNNs and other models has led to a set of intriguing discoveries.

Iterative solving emerged as a vital strategy in our approach, enabling models to progressively build upon certain predictions in a manner akin to human problem-solving. This method proved particularly effective in improving model performance. Augmentation, however, had a more modest impact.

In Figure 7.14, we can observe that GNNs generally outperformed the other models. A key part of this success was the encoding strategy, where cells in the same row, column, and subgrid were encoded to help the model understand the inherent relationships within a Sudoku puzzle.

Surprisingly, we noticed a decline in GNN's puzzle accuracy on easier puzzles. This was traced back to a few mistakes per puzzle leading to entire solutions being marked incorrect, disproportionately affecting the accuracy measurement.

Moreover, the encoding approach we adopted - which acknowledged the constraints of cells being in the same row, column, and subgrid - played a critical role in the effectiveness of the models. This insight can lead to refined strategies for Sudoku solving and potentially be applied to similar graph-based computational problems.

In conclusion, the study into Sudoku solving with Graph Neural Networks and other models has illuminated specific strategies, limitations, and nuances. Through careful application of iterative solving, along with a nuanced understanding of encoding and the role of augmentation, we have forged a path towards enhanced accuracy in solving Sudoku puzzles. These findings not only further the field of computational Sudoku solving but also hold the potential to inform new methodologies in other domains where similar graph-based structures and iterative methods might be beneficial.

Chapter 8

Solving Kakuro Puzzles

Kakuro, also known as Cross Sums, is a mathematical and logic puzzle game that draws parallels to both Sudoku and Crossword puzzles. It consists of a blank grid with sum clues provided in various places. The player's objective is to fill the grid with digits from 1 to 9 in such a way that the sum of the numbers in each row or column matches the clue associated with it. Additionally, no digit can be repeated within a single sum. Despite its straightforward rules, Kakuro puzzles range widely in complexity and size, providing engaging challenges that test both arithmetic skills and logical reasoning.

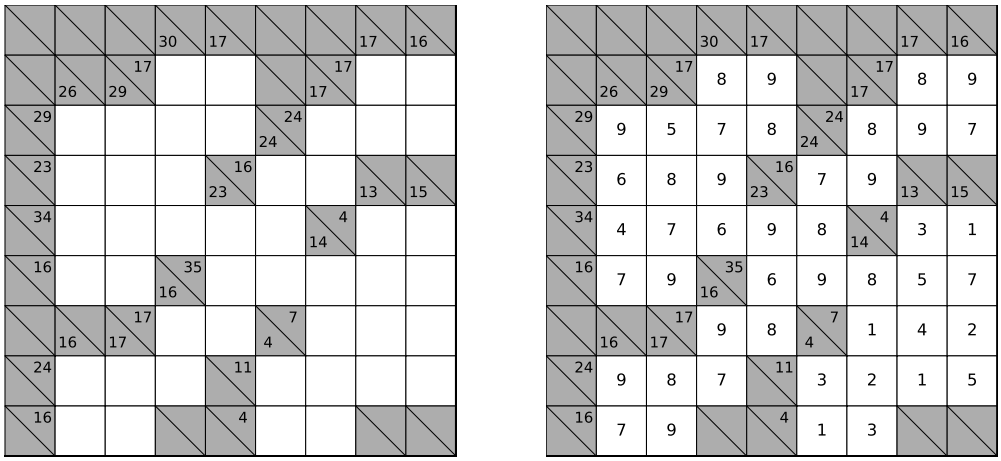


Fig. 8.1: Instance of a Kakuro Puzzle on the left side with the solution on the right side

8.1 Dataset

The dataset for this research was sourced from [Kakuro Conquest, 2023]. However, as this site does not provide puzzle solutions, a Linear Programming (LP) solver was used to derive the answers which was found in [Daniec, 2020].

The puzzles in this dataset all posses a unique solution and it has a range of different levels, going from, easy, medium, hard, intermediate, expert and elite. The difficulty categorizations were predefined by the source platform and was not established by us during our study.

The difficulty of a Kakuro puzzle is typically influenced by factors such as 'cross sum' complexity and cell interdependency. The former refers to the difficulty of finding a unique combination of numbers that fits a given sum. For instance, a clue of 16 for a two-cell word (with only one solution, 7 and 9) is easier compared to a clue of 15 for a three-cell word (with multiple solutions such as 2-6-7, 3-4-8, etc.).

On the other hand, puzzles are more challenging when they have a high degree of interdependency between cells, where the solution to one cell is intertwined with potential solutions to others. This attribute and 'cross sum' complexity largely contribute to the puzzle difficulty categories in our dataset.

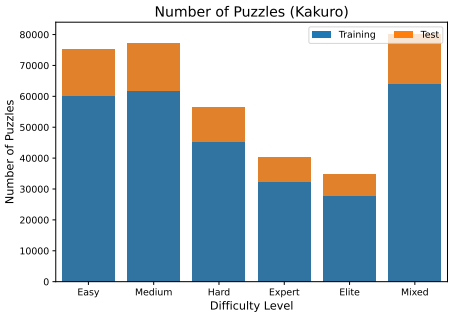


Fig. 8.2: Distributions of our Kakuro Puzzle Dataset

As we can see in Figure 8.2, we also possess a dataset of mixed difficulty. This dataset is uniformly sampled from all other datasets and set to an amount that matches the two largest datasets, namely those of easy and medium difficulty to not put any bias into having a much larger dataset when training. We are mostly interested in, whether a mixed dataset provides better generalization results, due to the model having seen puzzles across all difficulties.

8.2 Solving Kakuro using CNNs

Encoding Kakuro puzzles for them to be suitable for CNNs requires a bit more work, due to the way puzzles are structured. Nonetheless, Kakuro puzzles are structured in a grid, which allows the CNN to process the data in a similiar fashion as in Sudoku. Here we have one issue, where certain clue cells can have different clues for the horizontal and vertical direction. Since there are also cells, which do not need to be predicted at all, we have to encode it as well.

To encode all these restrictions, we came up with the following method. A 9x9 grid is created with three channels, where the first channel represents the values of the horizontal clues, the second channel the values of the vertical clues, and the third channel has a '1' in it if the cell needs to be predicted and '0' otherwise.

8.2.1 Training & Results

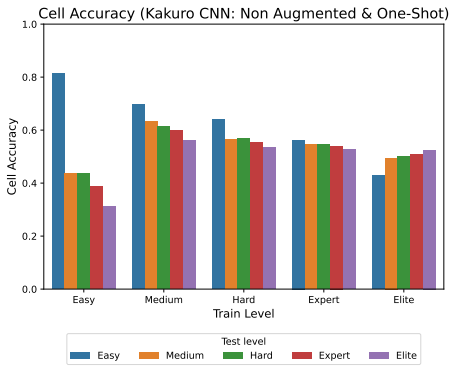


Fig. 8.3: Performance of CNN model: Cell Accuracy

In Figure 8.3, models trained on various difficulties show better performance on easier puzzles. The highest accuracy is usually achieved by models trained and tested on the same difficulty. A model trained on easy puzzles predicts fewer than 40% of cells correctly for other difficulties, while other models exhibit accuracy ranging from 50% to 60%. The model trained on medium puzzles performs similarly across different difficulties, possibly due to a larger dataset or exposure to more puzzle types. Since none of the models solved the puzzles completely, the graph displaying puzzle accuracy has been omitted.

8.2.2 Augmentation & Iterative Solving

To have the CNN allow for augmentation we add an additionally channel which can encode the end value for a cell. The model is adapted slightly to process 4

channels for the first layer instead of 3 channels, otherwise the model is completely identical. We can augment the dataset by uniformly choosing nodes where we encoded in the end value. We do this twice for each puzzle, and include the non-augmented puzzle with our dataset. This triples our dataset, but we are hoping to have it be able to recognize the rules better.

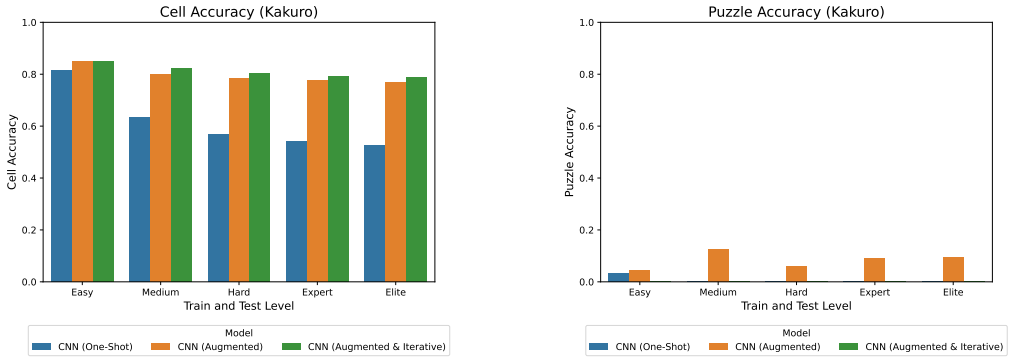


Fig. 8.4: Performance Comparison of CNN Models, where we compared model trained on the normal dataset, augmented dataset. Where on the augmented dataset we also have an iterative approach

Here we can clearly see in Figure 8.4 that the model trainedd on the augmented dataset leads to the best results. However, iterative solving does increase cell accuracy, but is not able to solve any puzzle correctly completely.

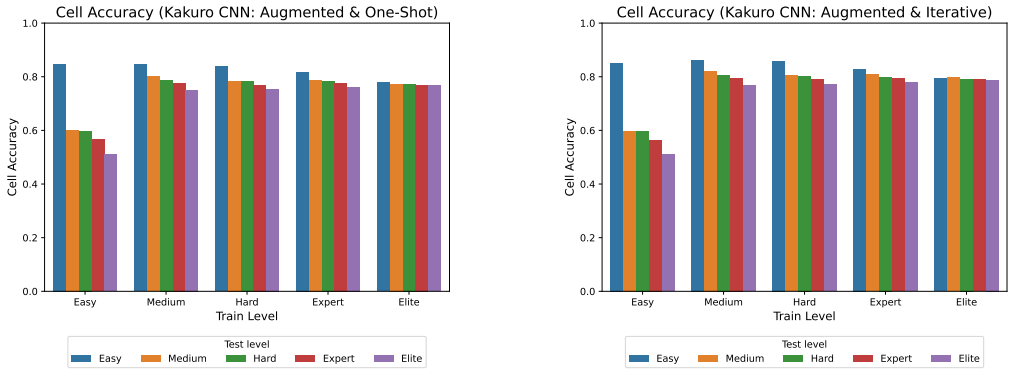


Fig. 8.5: Cell Accuracy extrapolation comparison for CNN models trained on augmented dataset (one shot and iterative solving)

8.2.3 Final Conclusion

We can see in Figure Figure 8.4, the results for the augmented training dataset with one-shot and with iterative solving. When putting the graphs next to each

other, the performance looks similar, but also the iterative solving leads to similar looking graphs. In the case of Kakuro iterative solving does not yield a significant increase in performance, but as we could see before augmentation of our dataset does help our CNN model learn better and perform better.

8.3 Solving Kakuro using GNN

8.3.1 Encoding

In this section we consider using a graph representation to model our Kakuro puzzles and use GNNs to try to solve our puzzles. One of the main benefits of using graphs, is that we can encode the problem in different ways, each with its own advantages and disadvantages.

Clique Encoding: Encoding Direct Dependencies

Kakuro puzzles inherently consist of dependent cells, where empty cells are directly dependent on their corresponding clue cell as well as other cells that share the same row or column. These dependencies can be represented through 'clique encoding.' This idea stems from [Daniec, 2020], this follows a similar representation as in Sudoku, because it directly encodes the direct dependencies.

A clique, in graph theory, represents a subset of vertices of an undirected graph where every two distinct vertices are connected by a unique edge. In Kakuro's context, we can form a 'clique' by selecting a clue cell, where each clue represents a node, and identifying the empty cells whose end sum must correspond to the clue cell's value. These puzzle cells will each be represented as a node, connected to the clue cell which is also represented as a node. The puzzles nodes will have no intial value, while the clue node will have the sum as its value. This clique encapsulates the direct dependencies amongst them, and this process is replicated for all clue nodes, creating a robust encoding of the Kakuro puzzle's direct dependencies.

Star Encoding: Radiating Dependencies

Another approach to encoding Kakuro puzzles can be 'star encoding'. In this approach, we consider each clue cell as a star's core, and the dependent puzzle cells as the arms of the star. Each clue node is connected with a unique edge to its dependent cells. We represent the clue cell and the dependent puzzle cells each as a node, where clue node is connected to all dependent puzzle nodes directly. Similarly, only the clue node has an initial value, which is the sum, and the puzzle nodes have no initial value. This approach visualizes the dependencies in a more

centralized way, with the clue node at the heart of the relationships, emphasizing its role in determining the possible values of the dependent cells.

Path Encoding: Tracing the Dependencies

Yet another way to encode Kakuro puzzles can be achieved by creating path graphs, which we call 'path encoding'. Given that a Kakuro puzzle is structured in a grid layout, this approach is ideal for tracing the dependencies across the cells.

In path encoding, each cell in the grid is connected to the cell above it or to its left (assuming such a cell exists) only if their values are interdependent. This method offers a representation of how values propagate through the puzzle grid, thereby effectively illustrating the dependencies.

8.3.2 Training & Results

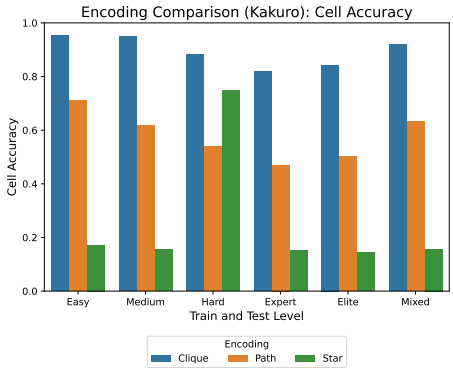


Fig. 8.6: Comparison of cell accuracy for all the three encodings: Clique, Star, Path

In the figure we see the models trained on different levels, and tested on the same train level. We can see in Figure 8.6 that the Clique Encoding leads to the best results. In a way this is not surprising, due to the way the direct dependencies are encoded implicitly. This would most likely make it easier for the GNN pick up the rules and we can see again, the models perform better for easier puzzles. Even though the source did not specify exactly, how the puzzles were sorted into different difficulties, we can see that the model performs similiar to the difficulties.

Due to these results, from here we continue on with only the clique encoding and ignore the other encodings, since they yield no significant results.

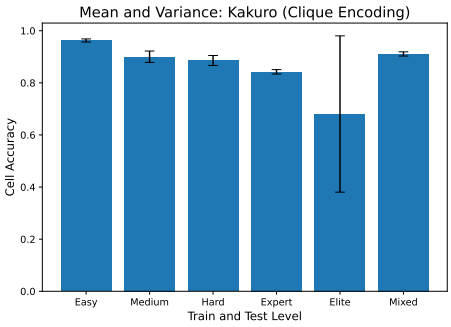


Fig. 8.7: Mean and Variance of the Cell Accuracy for the Clique Encoding over 5 runs for all the Kakuro puzzle levels

In Figure 8.7, we can also see the model is quite stable for different seeds. There does not seem to be much variance, except for the Elite difficulty. After inspecting the runs, there seems to be one run where the model does not converge at all, while for the other runs the performance is all quite similar. It looks promising, that the model generally does not have any issues converging and is quite stable regardless of seed.

Extrapolation Performance across different Difficulties

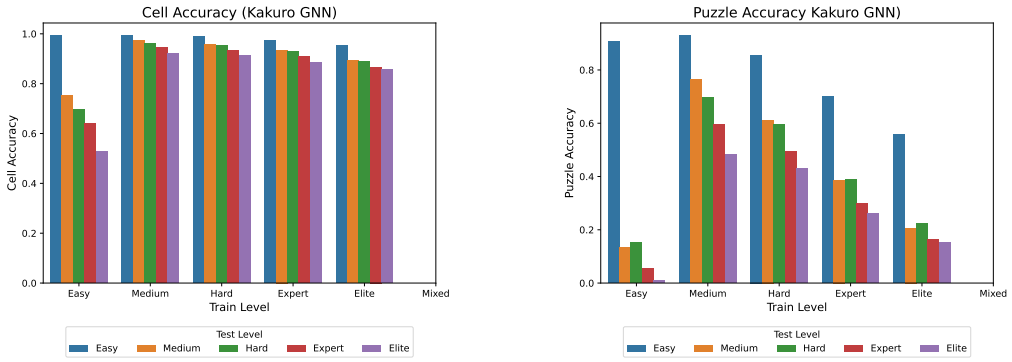


Fig. 8.8: Performance of GNN models trained on different level and tested on other puzzle levels

In Figure 8.8 we can see the model trained on the medium puzzles yields the highest accuracy for every difficulty, even beating models that are trained on that difficulty. This could be explained due to the model, when trained on medium puzzles, seeing a wider range and able to learn its rules. When trained on harder puzzles, it might fail at learning all the rules properly, and when trained on easy puzzles it might not learn how to solve more difficult instances, since it might have been to easy,

8.3.3 Augmenting & Iterative Solving

n Figure 8.8, the model trained on medium puzzles exhibits the highest accuracy across all difficulties, surpassing even those models trained specifically on each difficulty. This superior performance might be attributed to the medium-trained model encountering a more diverse range of puzzles, enabling it to learn their underlying rules more comprehensively. Conversely, training on harder puzzles might result in an incomplete understanding of the rules, while training on easier puzzles might not sufficiently equip the model to tackle more complex instances.

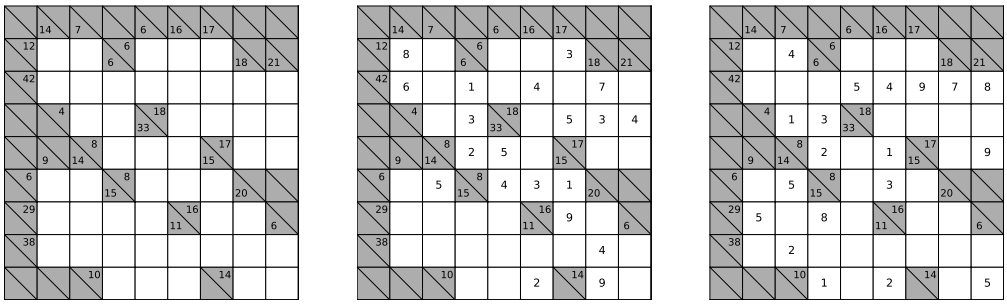


Fig. 8.9: Two augmentations of Kakuro Dataset

In Figure 8.9, the left graph presents an instance of a Kakuro puzzle as it is typically received, without rules provided. This can be seen as a case of unsupervised learning, as the cells to be filled do not contain any information about their final labels. Our approach is to augment the dataset, illustrated in the middle and right graphs, where we include the final classifications for some cells. This simplification helps the Graph Neural Network (GNN) to learn the relationships between the nodes more effectively and enables the model to solve the graph. The solution is reached by iteratively predicting nodes where the model has the highest certainty, continuing this process until all the nodes are completely predicted.

This approach is analogous to Sudoku and the Convolutional Neural Network (CNN) methodology. By augmenting our dataset and accommodating intermediate hints, we can iteratively solve Kakuro puzzles. The process involves filling out one cell of the puzzle at a time, specifically where the GNN is most confident about its value, and continuing this process until the puzzle is entirely filled out.

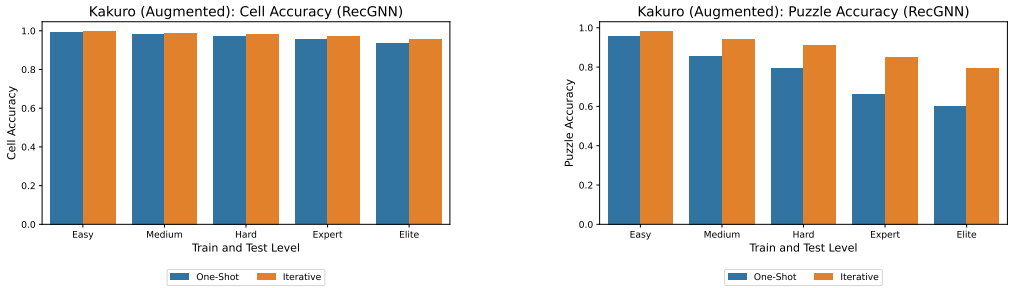


Fig. 8.10: Cell accuracy performance of GNN trained on augmented dataset, comparing the one-shot and iterative solving method

In Figure 8.10, we can see how for all of the difficulties we suddenly seem to have an almost perfect accuracy. This is surprising, especially for the one-shot variant, similar to the CNN, the GNN is able to pick up the dependencies much quicker. Again, similar to the CNN, we can employ an iterative solving, and here we also do see an increase in performance. This time it does lead to a great increase in the amount of puzzles that can be solved. The most likely explanation is that since we have been able to solve puzzles correctly to a high degree, we refine it and fewer mistakes are more unlikely to happen. For the easy difficulty we are able to solve almost all and for the hardest dataset we are able to solve almost 80% of the puzzles completely.

Example

The initial four steps illustrated in Figure 8.11 reveal some initial mispredictions by the GNN. However, the GNN quickly recalibrates, reducing these errors in subsequent steps, even for cells that are not directly connected. This capacity to propagate information to indirectly dependent cells highlights the GNN's ability to adapt to Kakuro's global puzzle nature.

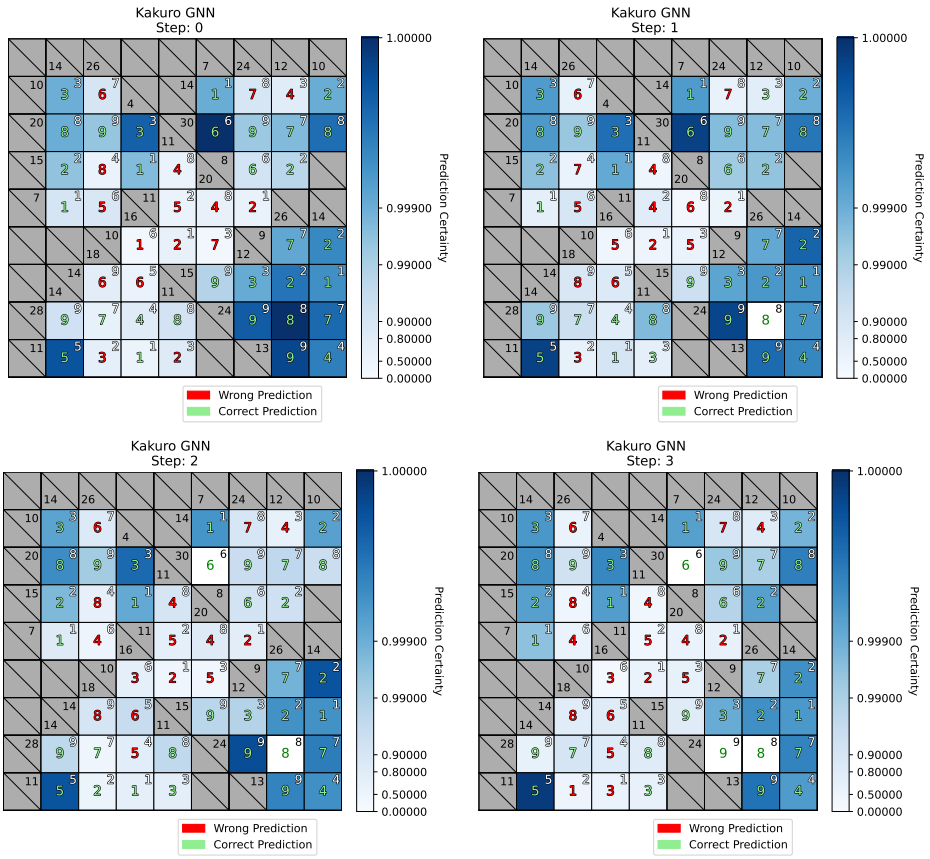


Fig. 8.11: Iterative Solving of Kakuro using GNN: first 4 iterations

Figure 8.12 showcases the GNN's learning progress in the latter stages, with the model making fewer mistakes compared to a one-shot approach. The GNN's strength lies in its ability to learn as it progresses, rather than guessing all values simultaneously. Interestingly, even when some predictions are incorrect, the GNN shows an understanding of Kakuro's summing rules, accurately predicting many values. This phenomenon highlights the GNN's ability to grasp global dependencies within the puzzle, leading to an improved approach to problem-solving.

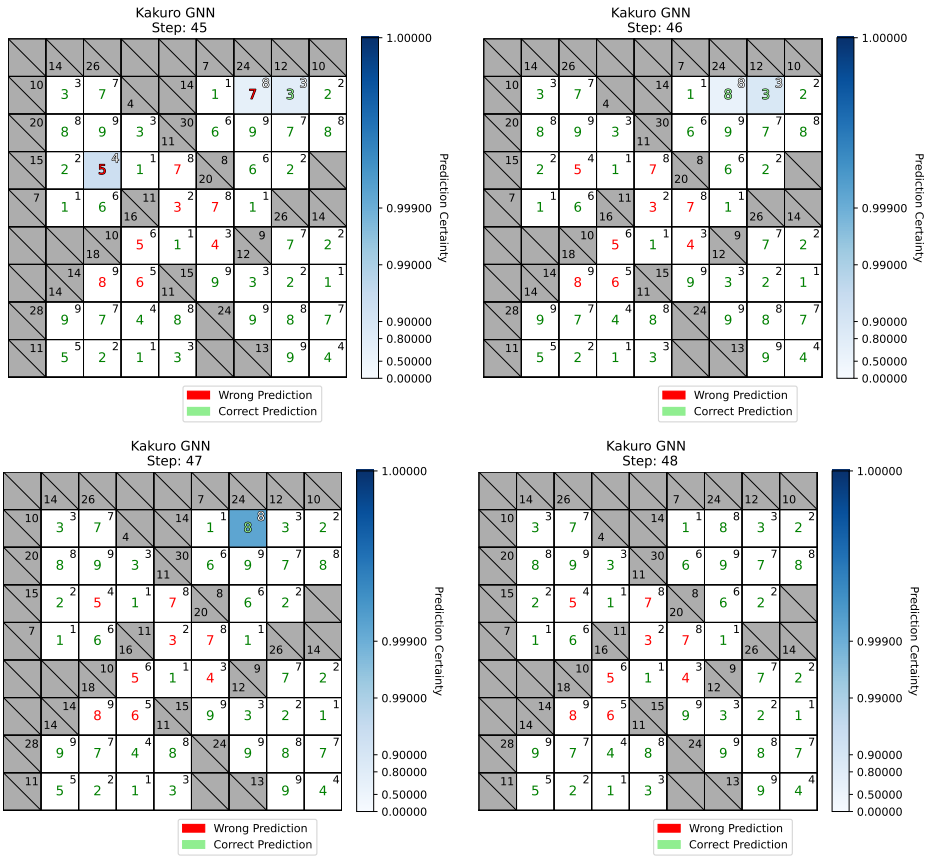


Fig. 8.12: Iterative Solving of Kakuro using GNN: last 4 iterations

8.4 Final Conclusion

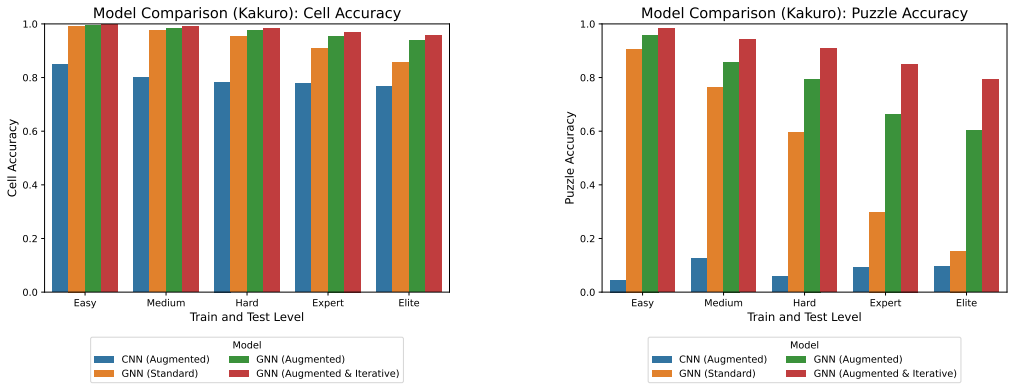


Fig. 8.13: Performance Comparison of all the four best models

Encoding Kakuro for analysis using CNNs presented unique challenges, leading us to investigate extrapolation capabilities and performance differences across various difficulties. Augmentation and iterative solving offered incremental improvements, but complete puzzle solving remained out of reach.

In contrast, the use of GNNs marked a significant leap forward for Kakuro solving. Clique Encoding proved highly effective, and the combination of augmentation and iterative solving with GNNs led to nearly perfect accuracy for some puzzle levels. As showcased in Figure 8.13, the GNN trained on the augmented dataset and employing iterative solving exhibited the highest performance, revealing a clear advantage for GNNs in solving Kakuro puzzles.

Throughout our study, we explored clever encoding strategies, dataset augmentation, and iterative solving techniques. These methods enhanced the performance of GNNs with regard to Kakuro puzzles. Specifically, iterative solving was implemented for both CNN and GNN, aiding the learning process.

Surprising to us was the extent to which GNNs outperformed traditional CNN models, likely due to the advanced encoding strategies employed. This discovery opens avenues for further research and potential refinements.

In conclusion, our findings underscore that a well-crafted combination of encoding, augmentation, and iterative solving can dramatically enhance a GNN's ability to solve Kakuro puzzles. The compelling results achieved by the GNN, especially in comparison to CNN models, highlight promising directions for future work and potential applications. The lessons drawn from the handling of both CNN and GNN may extend to other complex problem-solving domains, providing valuable insights for ongoing exploration and development.

Chapter 9

Solving Hitori Puzzles

Hitori is a logic puzzle originating from Japan. Played on a square grid initially populated with numbers, the core objective of Hitori revolves around the elimination of certain numbers. This process is symbolized by blacking out specific cells in the grid.

The puzzle reaches its solution when the grid is modified to a state wherein all three of the following conditions are concurrently satisfied:

1. *Uniqueness in rows and columns:* Each number in any given row or column must be unique. This implies that a specific number should not reappear in the same horizontal or vertical line.

2. *Non-adjacency of black cells:* Black cells, indicative of eliminated numbers, cannot be horizontally or vertically adjacent. However, they can be diagonally juxtaposed. This rule helps prevent the formation of blocks of black cells, ensuring the grid remains navigable.

3. *Connectivity of remaining cells:* All the remaining (non-black) cells must establish a connection with each other either horizontally or vertically. This rule ensures the existence of a path that can be traced from any non-black cell to any other using only horizontal or vertical steps.

The Hitori puzzle is solved using logical deduction, with each puzzle possessing only one unique solution. The complexity and difficulty of Hitori puzzles generally increase with the size of the grid, offering a variety of challenges to suit a wide range of players.

9.1 Dataset

For our research on Hitori puzzles, we collected the dataset from the website [Hanssen, 2023]. Unlike our Kakuro source, this site thankfully provides solutions for each puzzle, so we did not need to use a solver.

We divided the Hitori puzzles in a similar way to the Kakuro ones. We used 80% for training and set aside 20% for testing. This split led to a range of puzzles across seven levels of difficulty: easy, medium, hard, Expert, Elite, Master, Legend. These difficulty categories were already defined by the source platform, menneske.no, and were not set by us during our study.

The difficulty of a Hitori puzzle depends on a few key factors.

An important factor is how the numbers are spread out and repeated across the grid. If a puzzle has a lot of the same number, it can be more challenging because there are more instances of that number to isolate.

Lastly, the decisions to black out a cell or leave it uncolored affect the rest of the puzzle, which also contributes to its difficulty. This is because each decision you make can greatly influence what you can do with the other cells in the puzzle.

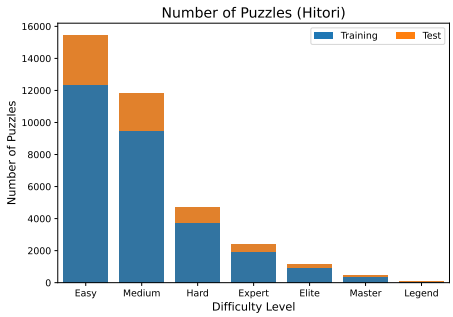


Fig. 9.1: Distribution of Hitori puzzles

Due to the small size of the dataset, except for the first three difficulties, we decided against training models on these datasets. In this chapter only models trained on easy, medium, and hard are considered and compared.

9.2 Solving Hitori with CNNs

Given the grid layout of Hitori puzzles, employing a CNN to solve them appears to be one of the most direct approaches. When compared to other grid-based puzzles like Sudoku and Kakuro, Hitori, with its 8x8 grid encoding, intuitively seems to be a more complex challenge. This complexity primarily arises from the global constraints that requires all unshaded cells must be interconnected.

One notable issue with complex problems is that neural networks may overfit or consistently predict the most probable class, even in situations where weighted cross-entropy adjustments are made. This was observed in our dataset, where the model consistently overfit or predicted '1' (unshaded).

No convergence could be achieved for any parameters in this case.

9.2.1 Augmentation and Iterative Solving

Drawing from the success in Kakuro and Sudoku, we explored augmentation and iterative solving techniques for Hitori, given that these strategies have previously aided in training, even without providing hints during inference.

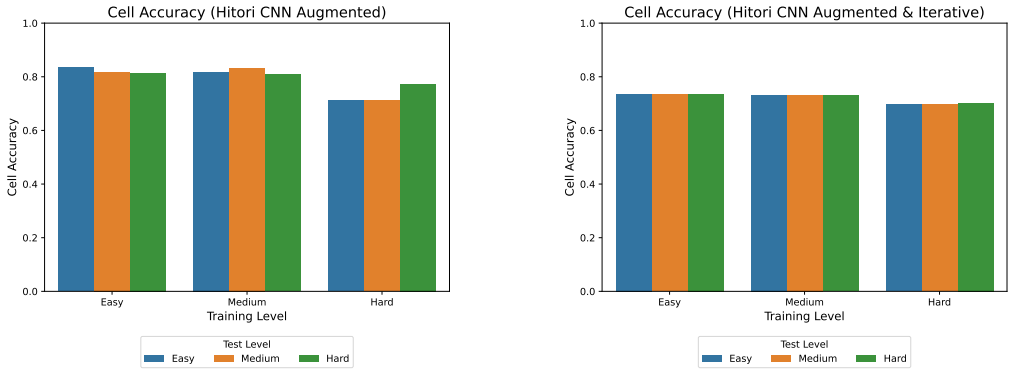


Fig. 9.2: Hitori CNN Performance on the Augmented Dataset, trained on different levels

As Figure 9.2 illustrates, augmentation indeed improves performance. Surprisingly, the extrapolation capability remains consistent across different difficulty levels, and the model performs similarly on all. The exception is the model trained on hard puzzles, which performs the worst. However, unlike previous puzzles, iterative solving does not enhance the solving process for Hitori.

We also experimented with permuting our dataset, as in the case of Sudoku. This approach, unfortunately, did not lead to any performance increase.

There could be two main explanations for why the mentioned approaches did not achieve full performance. Firstly, the Hitori puzzle is intrinsically complex and requires comprehensive global information, which might be beyond the reach of a CNN. Secondly, the lack of unique solutions for Hitori puzzles poses a significant challenge, as the neural network might struggle to converge on a definitive solution. Despite these obstacles, it is worth acknowledging that our model still performs better than naive guessing.

9.3 Solving Hitori with GNNs

9.3.1 Encoding

For GNNs to learn and tackle Hitori puzzles effectively, we encode them in a manner compatible with graph-based learning, reflecting the puzzle's rules. Each cell in the puzzle becomes a node in the graph, interconnected with other cells lying in the same row or column. This structure inherently represents the fundamental constraints of Hitori. Moreover, initial node features correspond to the cell values, aligning with the puzzle's numerical nature.

9.3.2 Training & Results

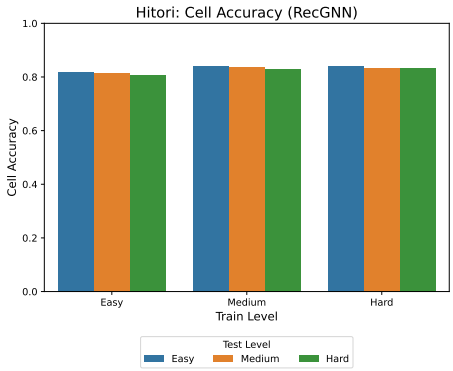


Fig. 9.3: Hitori (RecGNN) Extrapolation to different difficulties

Figure 9.3 illustrates our GNN results. Models trained on various difficulties exhibit similar performance, achieving roughly 81% accuracy. Although this demonstrates that the GNN can grasp some aspects of the rules, it also highlights shortcomings in distinguishing between different difficulty levels.

The reason for these challenges echoes some of the problems encountered with the CNN approach. Uniqueness in solutions is not guaranteed in Hitori, making convergence harder to achieve. Furthermore, unlike Sudoku or Kakuro, Hitori does not exhibit stark contrasts between its difficulty classes. Visual inspection reveals the model's failure to enforce the rule that black cells cannot be horizontally or vertically adjacent, which can be attributed to the more pronounced global dependencies inherent to Hitori.

9.3.3 Augmentation & Iterative Solving

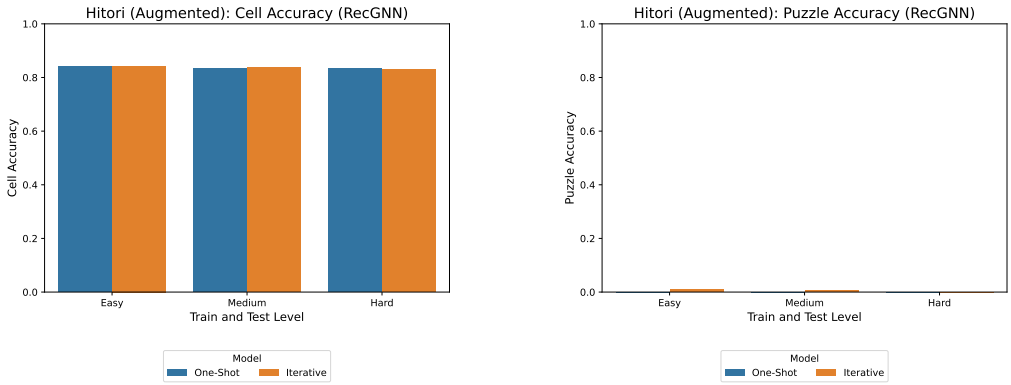


Fig. 9.4: Hitori GNN Performance on the Augmented Dataset, with iterative and one-shot solving. Left side cell accuracy, right side puzzle accuracy

Figure 9.4 explores the impacts of augmentation and iterative solving. Despite incorporating flags to indicate whether cells should be shaded or unshaded, augmentation failed to improve by anything noteworthy . This could be due to the complexity of Hitori, where such flags may not capture the underlying structure and logic. Iterative solving, shows the performance to be slightly better, but still rather, none of them seem to be able to complete solve any puzzles.

9.4 Final Conclusion

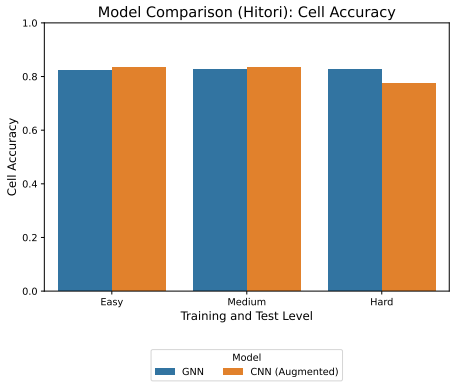


Fig. 9.5: Final Comparison of GNN and CNN (Augmented): cell accuracy

Our research illuminates the contrasting strengths and weaknesses between CNNs and GNNs in solving Hitori. As seen in Figure 9.5, GNNs performed marginally

worse in easy and medium puzzles but exhibited slight superiority in hard cases. This discrepancy might hint at GNNs being better equipped to maintain global dependencies, albeit subtly.

However, the similarity in accuracy between the two methods indicates that neither achieved perfection or the capacity to solve all puzzles. The main obstacles include the lack of unique solutions, complexity in differentiating between difficulty levels, and more nuanced global dependencies present in Hitori compared to other puzzles like Sudoku and Kakuro. The pursuit of perfect accuracy in Hitori thus remains an open challenge, requiring further exploration and potentially innovative modeling techniques tailored to this fascinating puzzle's unique characteristics.

Chapter 10

Conclusion

In this thesis, we embarked on an exploration of the application of augmentation and iterative solving techniques within Graph Neural Networks (GNNs), focusing on Algorithmic Datasets and Nikoli puzzles.

Our exploration of Maximum (Weighted) Independent Set problems revealed the GNN's proficiency in predicting the remaining nodes of a Maximal Independent Set, as this is largely a local problem. However, when applied to Maximum (Weighted) Independent Sets in trees, the model struggled. The introduction of iterative solving in these contexts helped overcome these challenges, leading to a small increase in performance.

In evaluating Reachability of nodes, our GNN performed exceptionally well, identifying core components and assigning nodes with near-perfect accuracy. This success highlighted the model's ability to process and understand interconnected structures without the need for deeper augmentation.

The implementation of iterative solving in Sudoku puzzles led to a significant performance boost in all except the easiest puzzles for the GNN. In this case the GNN mistakenly predicted a few cells per puzzle, which resulted in invalidating the whole solution, while still maintaining a high cell accuracy. Despite this, for the other difficulties, the result and improvement was significant. The GNN trained on an augmented dataset, combined with iterative inference, performed outstandingly, solving over 90% of the puzzles in all instances, a substantial improvement over CNNs. One thing we also discovered, is that encoding the graphs for the GNNs in such a way, which encapsulates the rules of the games, lead to the best results.

However, our experiments with Hitori were more mixed. While the CNN failed to converge, the use of augmentation made learning possible. In contrast to the other problems, iterative inference did not lead to improvement but rather a slight decrease in performance. The CNN and all the GNN models performed similarly, which could be attributed to the complexity of Hitori or non-uniqueness in the dataset.

Our research has shown that the integration of augmentation and itera-

tive solving within GNNs offers a promising pathway to enhancing learning and problem-solving. By employing these techniques, we have demonstrated substantial performance improvements across a diverse set of problems, ranging from traditional logic puzzles to more abstract algorithmic challenges. The varying success and limitations observed across different puzzles and scenarios provide valuable insights into the adaptive nature of these approaches, their dependencies, and their potential applicability.

Through this study, we have not only expanded the understanding of how GNNs can be effectively adapted and refined for complex problem-solving but also opened new doors for future research and innovation. The insights gained from our experiments, particularly in the intricate dynamics of puzzles like Kakuro and Sudoku, offer a roadmap for further exploration and refinement.

10.1 Future Work

Our current augmentation process could be further enriched by employing a wider variety of permutations during the training phase. By introducing a more extensive set of variations, the model might develop an even more nuanced understanding of the problem space.

While our methods have proven highly effective on Kakuro and Sudoku puzzles, there is room to test these techniques on more complex and diverse problems. This could include extending the approach to other NP-Complete problems or even problems outside the traditional puzzle domain.

Future work could also focus on fine-tuning the computational efficiency of these techniques. This includes optimizing both the iterative solving process and the augmentation strategies, to ensure they are not only effective but also resource-efficient.

Bibliography

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Roberto Cahuantzi, Xinye Chen, and Stefan Güttel. A comparison of lstm and gru networks for learning symbolic sequences. *arXiv preprint arXiv:2107.02248*, 2021.

Wiktor Daniec. *Solving Kakuro Problems using Recurrent Relational Networks*. PhD thesis, 09 2020.

Jeffrey L Elman. Learning and development in neural networks: The importance of starting small. *Cognition*, 48(1):71–99, 1993.

Florian Grötschla, Joël Mathys, and Roger Wattenhofer. Learning graph algorithms with recurrent graph neural networks. *arXiv preprint arXiv:2212.04934*, 2022.

Vegard Hanssen. Menneske.no, 2023. URL http://www.menneske.no. Accessed: 2023-05-01.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. pmlr, 2015.

Kakuro Conquest. Kakuro Conquest, 2023. URL http://www.kakuroconquest.com. Accessed: 2023-04-01.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521 (7553):436–444, 2015.

AA Leman and Boris Weisfeiler. A reduction of a graph to a canonical form and an algebra arising during this reduction. *Nauchno-Technicheskaya Informatsiya*, 2(9):12–16, 1968.

Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493*, 2015.

Allen Newell, Herbert Alexander Simon, et al. *Human problem solving*, volume 104. Prentice-hall Englewood Cliffs, NJ, 1972.

Rasmus Berg Palm, Ulrich Paquet, and Ole Winther. Recurrent relational networks. *arXiv preprint arXiv:1711.08028*, 2018. doi: 10.48550/arXiv.1711.08028. URL https://arxiv.org/abs/1711.08028. Accepted at NIPS 2018.

F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1): 61–80, 2008.

Takayuki YATO and Takahiro SETA. Complexity and completeness of finding another solution and its application to puzzles. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E86-A, 05 2003.

Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *AI open*, 1:57–81, 2020.

Appendix A

Independent Set Results

A.1 Maximal Independent Set

type	size	amount	train_loss	train_acc	batch_acc	train_f1
tree-mis	10	1000	0.00	1.00	1.00	1.00
tree-mis	50	1000	0.00	1.00	1.00	1.00
tree-mis	100	1000	0.00	1.00	1.00	1.00
tree-mis	1000	100	0.00	1.00	1.00	1.00

type	size	amount	train_loss	train_acc	batch_acc	train_f1
graph-mis	10	1000	0.00	1.00	1.00	1.00
graph-mis	50	1000	0.00	1.00	1.00	1.00
graph-mis	100	1000	0.00	1.00	1.00	1.00
graph-mis	1000	100	0.00	1.00	1.00	1.00

type	size	amount	train_loss	train_acc	batch_acc	train_f1
tree-mwis	10	1000	0.07	0.97	0.06	0.98
tree-mwis	50	1000	0.34	0.97	0.00	0.98
tree-mwis	100	1000	0.41	0.97	0.00	0.97
tree-mwis	1000	100	0.38	0.97	0.00	0.98

type	size	amount	train_loss	train_acc	batch_acc	train_f1
tree-mumis	10	1000	0.03	0.98	0.03	0.99
tree-mumis	50	1000	0.11	0.96	0.00	0.97
tree-mumis	100	1000	0.12	0.96	0.00	0.97
tree-mumis	1000	100	0.12	0.96	0.00	0.97

type	size	amount	train_acc	train_acc_iterative
tree-mwis-aug	10	1000	0.97	0.98
tree-mwis-aug	50	1000	0.97	0.98
tree-mwis-aug	100	100	0.97	0.99
tree-mwis-aug	1000	100	0.96	0.98

type	size	amount	train_acc	train_acc_iterative
tree-mumis-aug	10	1000	0.98	0.99
tree-mumis-aug	50	1000	0.96	0.97
tree-mumis-aug	100	100	0.96	0.96
tree-mumis-aug	1000	100	0.95	0.96

type	size	amount	batch_acc	batch_acc_iterative
tree-mwis-aug	10	1000	0.88	0.95
tree-mwis-aug	50	1000	0.48	0.78
tree-mwis-aug	100	100	0.25	0.73
tree-mwis-aug	1000	100	0.00	0.00

type	size	amount	batch_acc	batch_acc_iterative
tree-mumis-aug	10	1000	0.81	0.91
tree-mumis-aug	50	1000	0.12	0.38
tree-mumis-aug	100	100	0.00	0.10
tree-mumis-aug	1000	100	0.00	0.00

Appendix B

Reachability Results

type	size	amount	train_loss	train_acc	batch_acc	train_f1
connected-nodes	15	1000	0.01	1.00	0.99	1.00
connected-nodes	50	1000	0.00	1.00	1.00	1.00
connected-nodes	100	1000	0.00	1.00	0.98	1.00
connected-nodes	1000	100	0.00	1.00	0.87	1.00

type	size	amount	train_loss	train_acc	batch_acc	train_f1
connected-components	15	1000	0.01	1.00	0.98	1.00
connected-components	50	1000	0.00	1.00	1.00	1.00
connected-components	100	1000	0.00	1.00	1.00	1.00
connected-components	1000	100	0.00	1.00	1.00	1.00

Appendix C

Sudoku Results

C.1 Sudoku: Encoding Comparison

model	train_level	test_level	loss	acc	batch_acc	f1
Standard	easy	easy	0.00	1.00	0.98	1.00
Grid	easy	easy	0.34	0.52	0.00	0.89
Standard	medium	medium	0.01	0.99	0.86	1.00
Grid	medium	medium	1.11	0.33	0.00	0.68
Standard	hard	hard	0.44	0.80	0.01	0.88
Grid	hard	hard	2.03	0.22	0.00	0.54

C.2 Sudoku CNN: Results

train_level	test_level	loss	total_acc	total_acc_iterative	cell_acc	cell_acc_iterative	puzzle_acc	puzzle_acc_iterative
Easy	Easy	0.00	1.00	1.00	1.00	1.00	1.00	0.98
Easy	Medium	0.53	0.92	0.98	0.83	0.95	0.01	0.79
Easy	Hard	1.55	0.78	0.85	0.63	0.75	0.00	0.24
Medium	Easy	0.01	1.00	1.00	0.99	1.00	0.92	0.98
Medium	Medium	0.10	0.99	0.99	0.97	0.99	0.80	0.94
Medium	Hard	1.63	0.79	0.85	0.65	0.74	0.00	0.21
Hard	Easy	0.05	0.99	1.00	0.98	0.99	0.69	0.95
Hard	Medium	0.95	0.90	0.94	0.78	0.87	0.00	0.42
Hard	Hard	0.46	0.95	0.93	0.92	0.89	0.76	0.55
Mixed	Easy	0.02	1.00	1.00	0.99	1.00	0.87	0.98
Mixed	Medium	0.69	0.92	0.96	0.83	0.92	0.03	0.62
Mixed	Hard	2.01	0.78	0.82	0.62	0.69	0.02	0.12

C.3 Sudoku RNN: Results

train_level	test_level	loss	total_acc	total_acc_iterative	cell_acc	cell_acc_iterative	puzzle_acc	puzzle_acc_iterative
Easy	Easy	0.04	0.99	1.00	0.95	0.98	0.38	0.85
Easy	Medium	0.45	0.84	0.92	0.68	0.83	0.00	0.25
Easy	Hard	0.88	0.70	0.76	0.51	0.60	0.00	0.02
Medium	Easy	0.04	0.98	1.00	0.94	0.99	0.34	0.86
Medium	Medium	0.25	0.89	0.98	0.76	0.96	0.00	0.71
Medium	Hard	0.55	0.75	0.90	0.58	0.82	0.00	0.36
Hard	Easy	0.07	0.97	0.99	0.91	0.96	0.17	0.64
Hard	Medium	0.29	0.87	0.96	0.72	0.92	0.00	0.40
Hard	Hard	0.53	0.76	0.88	0.59	0.80	0.00	0.20
Mixed	Easy	0.04	0.98	1.00	0.93	0.99	0.31	0.89
Mixed	Medium	0.27	0.87	0.98	0.73	0.96	0.00	0.73
Mixed	Hard	0.54	0.75	0.90	0.58	0.84	0.00	0.41

C.4 Sudoku GNN: Results

model	train_level	test_level	loss	acc	acc_iterative	cell_acc	cell_iterative_acc	batch_acc	batch_iterative_acc	f1
standard	easy	easy	0.00	1.00	0.99	1.00	0.99	0.97	0.27	1.00
standard	easy	medium	0.45	0.94	0.99	0.88	0.97	0.06	0.87	0.94
standard	easy	hard	1.92	0.78	0.88	0.62	0.79	0.00	0.35	0.78
standard	medium	easy	0.00	1.00	0.99	1.00	0.99	1.00	0.27	1.00
standard	medium	medium	0.01	1.00	1.00	0.99	1.00	0.87	1.00	1.00
standard	medium	hard	0.20	0.92	0.99	0.86	0.99	0.06	0.97	0.92
standard	hard	easy	0.00	1.00	0.99	1.00	0.99	1.00	0.27	1.00
standard	hard	medium	0.03	0.99	1.00	0.98	1.00	0.64	1.00	0.99
standard	hard	hard	0.44	0.88	0.98	0.80	0.97	0.01	0.91	0.88
standard	mixed	easy	0.00	1.00	0.99	1.00	0.99	1.00	0.27	1.00
standard	mixed	medium	0.01	1.00	1.00	0.99	1.00	0.84	1.00	1.00
standard	mixed	hard	0.24	0.91	0.99	0.85	0.98	0.05	0.96	0.91

C.5 Sudoku GNN: Standard and Iterative Results

model	difficulty	variable	value
Augmented	Easy	puzzle_acc	1.00
Standard	Easy	puzzle_acc	0.97
Augmented	Medium	puzzle_acc	0.92
Standard	Medium	puzzle_acc	0.87
Augmented	Hard	puzzle_acc	0.12
Standard	Hard	puzzle_acc	0.01
Augmented	Easy	puzzle_acc_iterative	0.26
Standard	Easy	puzzle_acc_iterative	0.26
Augmented	Medium	puzzle_acc_iterative	1.00
Standard	Medium	puzzle_acc_iterative	1.00
Augmented	Hard	puzzle_acc_iterative	0.90
Standard	Hard	puzzle_acc_iterative	0.91

model	difficulty	variable	value
Augmented	Easy	cell_acc	1.00
Standard	Easy	cell_acc	1.00
Augmented	Medium	cell_acc	0.99
Standard	Medium	cell_acc	0.99
Augmented	Hard	cell_acc	0.86
Standard	Hard	cell_acc	0.80
Augmented	Easy	cell_acc_iterative	0.99
Standard	Easy	cell_acc_iterative	0.99
Augmented	Medium	cell_acc_iterative	1.00
Standard	Medium	cell_acc_iterative	1.00
Augmented	Hard	cell_acc_iterative	0.97
Standard	Hard	cell_acc_iterative	0.97

C.6 Model Comparison: GNN, CNN, RNN

difficulty	cell_acc_iterative	model
easy	0.99	Sudoku GNN (Iterative)
medium	1.00	Sudoku GNN (Iterative)
hard	0.97	Sudoku GNN (Iterative)
easy	1.00	CNN (Iterative)
medium	0.99	CNN (Iterative)
hard	0.89	CNN (Iterative)
easy	0.98	RNN (Iterative)
medium	0.96	RNN (Iterative)
hard	0.80	RNN (Iterative)

difficulty	puzzle_acc_iterative	model
easy	0.26	Sudoku GNN (Iterative)
medium	1.00	Sudoku GNN (Iterative)
hard	0.91	Sudoku GNN (Iterative)
easy	0.98	CNN (Iterative)
medium	0.94	CNN (Iterative)
hard	0.55	CNN (Iterative)
easy	0.85	RNN (Iterative)
medium	0.71	RNN (Iterative)
hard	0.20	RNN (Iterative)

Appendix D

Kakuro Results

D.1 Kakuro CNN: Results

D.1.1 Cell Accuracy

Difficulty	CNN (One-Shot)	CNN (Augmented)	CNN (Augmented & Iterative)
Easy	0.814034	0.848406	0.850879
Medium	0.632859	0.801285	0.822070
Hard	0.569735	0.783091	0.801629
Expert	0.539720	0.776623	0.792996
Elite	0.524823	0.767522	0.787401

D.1.2 Puzzle Accuracy

Difficulty	CNN (One-Shot)	CNN (Augmented)	CNN (Augmented & Iterative)
Easy	0.032282	0.045527	0.0
Medium	0.000000	0.126688	0.0
Hard	0.000000	0.059118	0.0
Expert	0.000000	0.091575	0.0
Elite	0.000000	0.094383	0.0

D.2 Different Encoding Accuracies

difficulty	puzzle_type	cell_acc
easy	Clique	0.95
easy	Path	0.71
easy	Star	0.17
elite	Clique	0.84
elite	Path	0.50
elite	Star	0.14
expert	Clique	0.82
expert	Path	0.47
expert	Star	0.15
hard	Clique	0.88
hard	Path	0.54
hard	Star	0.75
medium	Clique	0.95
medium	Path	0.62
medium	Star	0.16
mixed	Clique	0.92
mixed	Path	0.63
mixed	Star	0.16

D.3 Kakuro GNN: Results

train_level	test_level	loss	acc	cell_acc	puzzle_acc	f1
easy	easy	0.02	1.00	0.99	0.91	1.00
easy	medium	0.48	0.85	0.75	0.13	0.85
easy	hard	0.57	0.81	0.69	0.15	0.81
easy	expert	0.66	0.78	0.64	0.06	0.78
easy	elite	0.84	0.70	0.53	0.01	0.70
medium	easy	0.02	1.00	0.99	0.93	1.00
medium	medium	0.08	0.98	0.97	0.76	0.98
medium	hard	0.11	0.98	0.96	0.70	0.98
medium	expert	0.15	0.97	0.95	0.60	0.97
medium	elite	0.23	0.95	0.92	0.48	0.95
hard	easy	0.04	0.99	0.99	0.86	0.99
hard	medium	0.13	0.98	0.96	0.61	0.98
hard	hard	0.16	0.97	0.95	0.59	0.97
hard	expert	0.21	0.96	0.93	0.49	0.96
hard	elite	0.29	0.94	0.91	0.43	0.94
expert	easy	0.10	0.98	0.97	0.70	0.98
expert	medium	0.21	0.96	0.94	0.39	0.96
expert	hard	0.24	0.96	0.93	0.39	0.96
expert	expert	0.29	0.94	0.91	0.30	0.94
expert	elite	0.37	0.93	0.89	0.26	0.93
elite	easy	0.10	0.97	0.95	0.56	0.97
elite	medium	0.22	0.94	0.89	0.21	0.94
elite	hard	0.24	0.93	0.89	0.22	0.93
elite	expert	0.28	0.92	0.87	0.16	0.92
elite	elite	0.31	0.91	0.86	0.15	0.91

D.4 Kakuro GNN (Augmented): Results

difficulty	loss	acc	acc_iterative	cell_acc	cell_acc_iterative	puzzle_acc	puzzle_acc_iterative	f1
easy	0.02	1.00	1.00	1.00	1.00	0.96	0.98	1.00
medium	0.08	0.99	0.99	0.98	0.99	0.86	0.94	0.99
hard	0.10	0.98	0.99	0.97	0.98	0.79	0.91	0.98
expert	0.15	0.97	0.98	0.95	0.97	0.66	0.85	0.97
elite	0.24	0.96	0.97	0.94	0.96	0.60	0.79	0.96

Appendix E

Hitori Results

E.1 Hitori CNN (Augmented): Results

train_level	test_level	loss	cell_acc	cell_acc_iterative	puzzle_acc	puzzle_acc_iterative
Easy	Easy	0.40	0.83	0.73	0.00	0.00
Easy	Medium	0.42	0.82	0.73	0.00	0.00
Easy	Hard	0.43	0.81	0.73	0.00	0.00
Medium	Easy	0.42	0.82	0.73	0.00	0.00
Medium	Medium	0.39	0.83	0.73	0.00	0.00
Medium	Hard	0.43	0.81	0.73	0.00	0.00
Hard	Easy	0.58	0.71	0.70	0.00	0.00
Hard	Medium	0.58	0.71	0.70	0.00	0.00
Hard	Hard	0.49	0.77	0.70	0.00	0.00

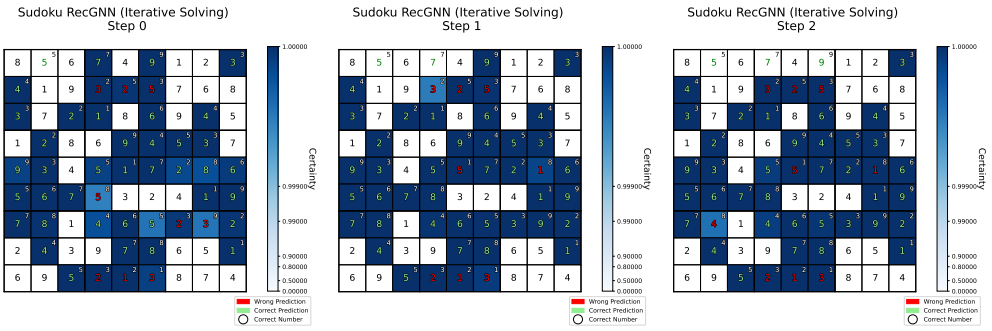
E.2 Hitori GNN (Augmented): Results

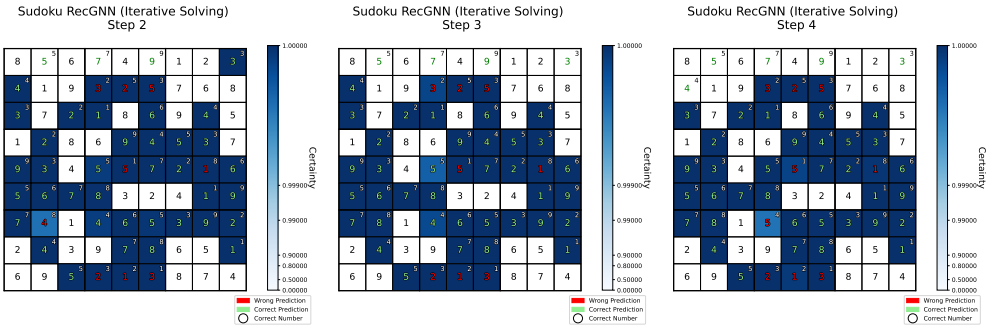
model	train_level	test_level	loss	acc	puzzle_acc	f1
One-Shot	easy	easy	0.38	0.84	0.00	0.89
Iterative	easy	easy	0.38	0.84	0.01	0.89
One-Shot	medium	medium	0.36	0.83	0.00	0.88
Iterative	medium	medium	0.36	0.84	0.01	0.88
One-Shot	hard	hard	0.40	0.84	0.00	0.88
Iterative	hard	hard	0.40	0.83	0.00	0.88

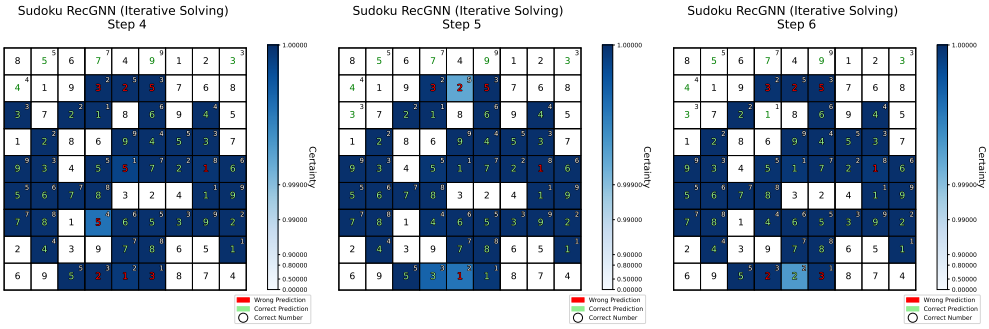
Appendix F

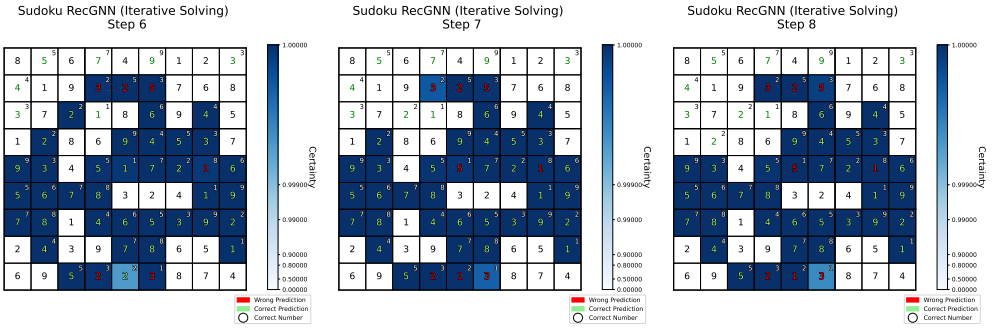
Iterative Solving

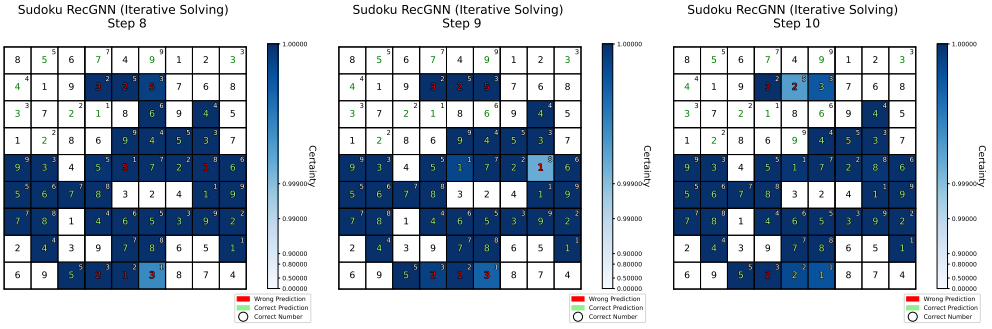
F.1 Sudoku GNN

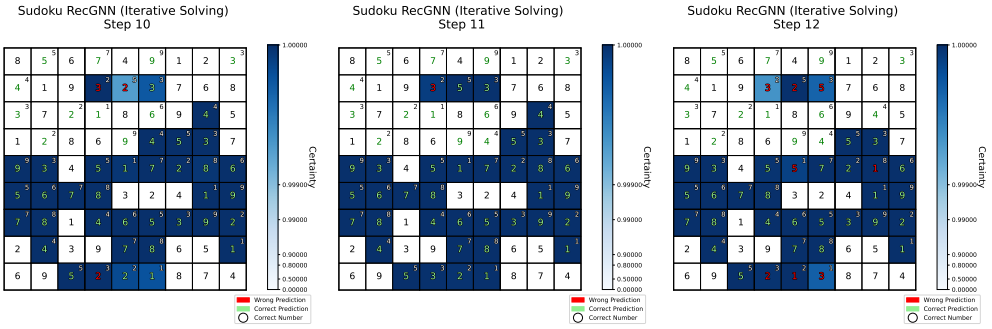


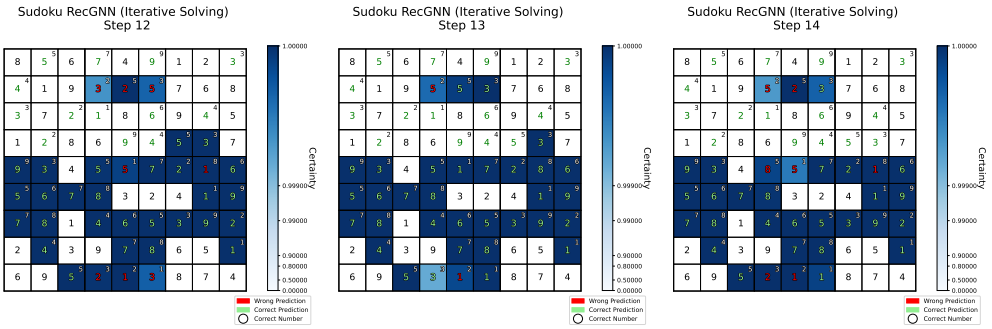


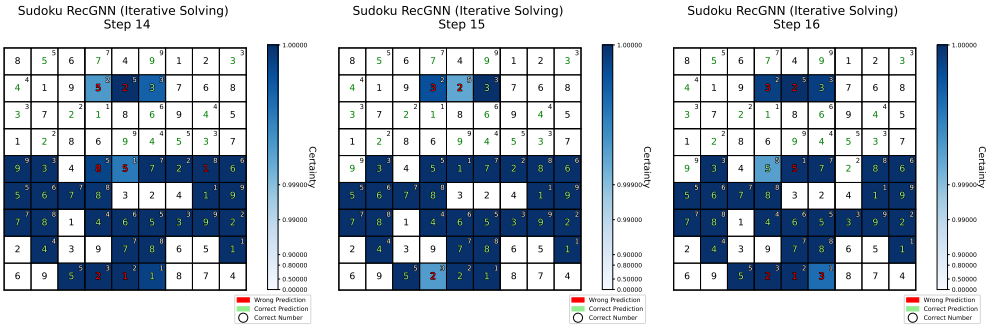

Sudoku RecGNN (Iterative Solving)
Step 4
Sudoku RecGNN (Iterative Solving)
Step 5
Sudoku RecGNN (Iterative Solving)
Step 6
Certainty
1.00000
0.99900
0.99000
0.90000
0.80000
0.50000
0.00000
Wrong Prediction
Correct Prediction
Correct Number

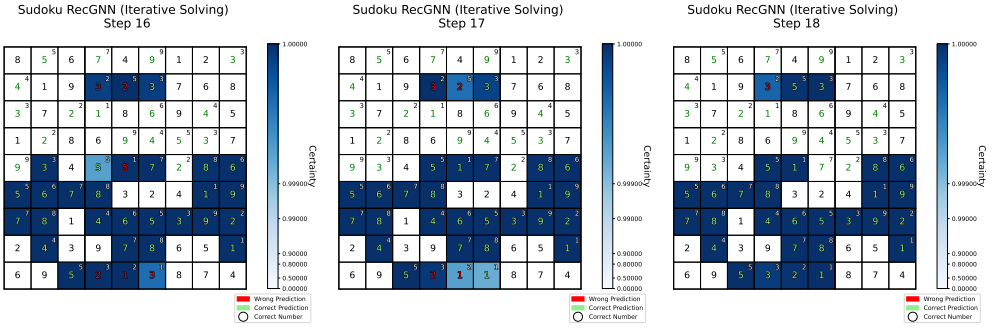

Sudoku RecGNN (Iterative Solving)
Step 6
Sudoku RecGNN (Iterative Solving)
Step 7
Sudoku RecGNN (Iterative Solving)
Step 8
Certainty
Wrong Prediction
Correct Prediction
Correct Number

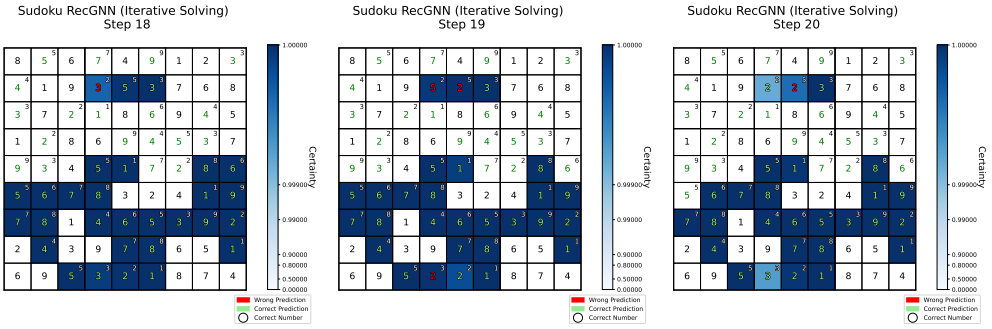

Sudoku RecGNN (Iterative Solving)
Step 8
Sudoku RecGNN (Iterative Solving)
Step 9
Sudoku RecGNN (Iterative Solving)
Step 10
Certainty
Wrong Prediction
Correct Prediction
Correct Number

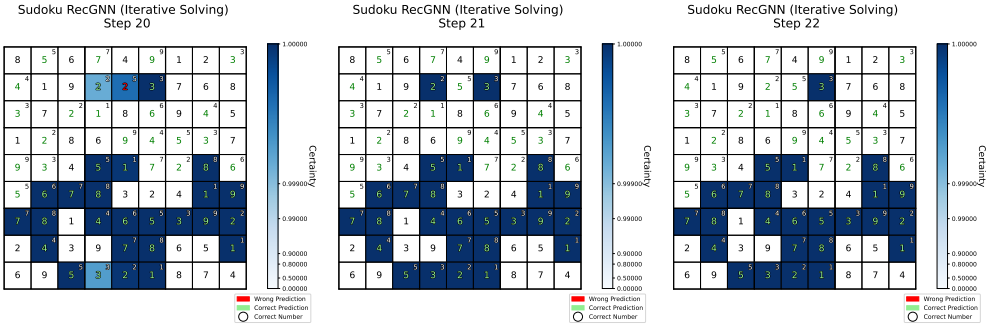

Sudoku RecGNN (Iterative Solving)
Step 10
Sudoku RecGNN (Iterative Solving)
Step 11
Sudoku RecGNN (Iterative Solving)
Step 12
Certainty
Wrong Prediction
Correct Prediction
Correct Number

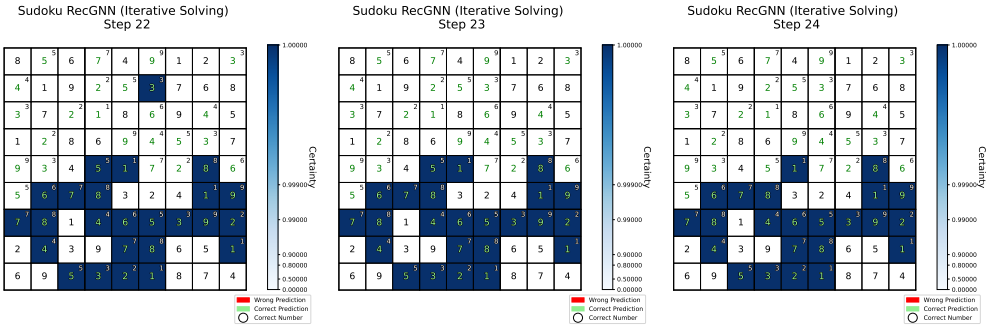

Sudoku RecGNN (Iterative Solving)
Step 12
Sudoku RecGNN (Iterative Solving)
Step 13
Sudoku RecGNN (Iterative Solving)
Step 14
Certainty
Wrong Prediction
Correct Prediction
Correct Number

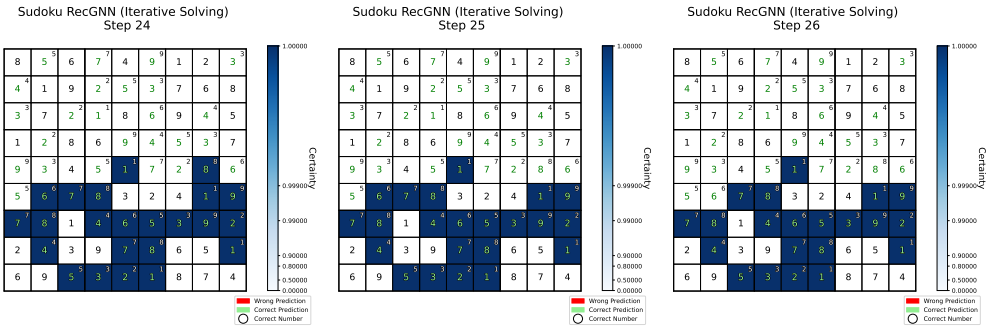

Sudoku RecGNN (Iterative Solving)
Step 14
Sudoku RecGNN (Iterative Solving)
Step 15
Sudoku RecGNN (Iterative Solving)
Step 16
Certainty
Wrong Prediction
Correct Prediction
Correct Number

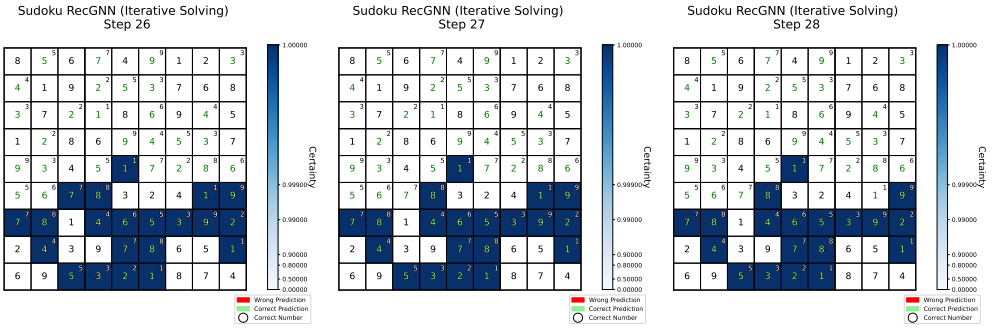

Sudoku RecGNN (Iterative Solving)
Step 16
Sudoku RecGNN (Iterative Solving)
Step 17
Sudoku RecGNN (Iterative Solving)
Step 18
Certainty
Wrong Prediction
Correct Prediction
Correct Number

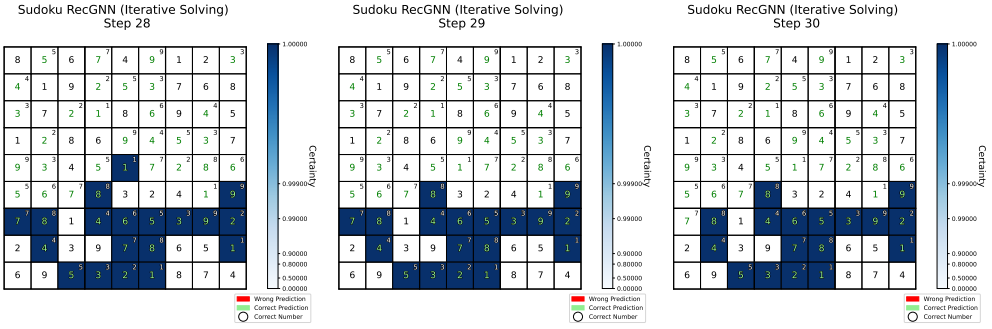

Sudoku RecGNN (Iterative Solving)
Step 18
Sudoku RecGNN (Iterative Solving)
Step 19
Sudoku RecGNN (Iterative Solving)
Step 20
Certainty
Wrong Prediction
Correct Prediction
Correct Number

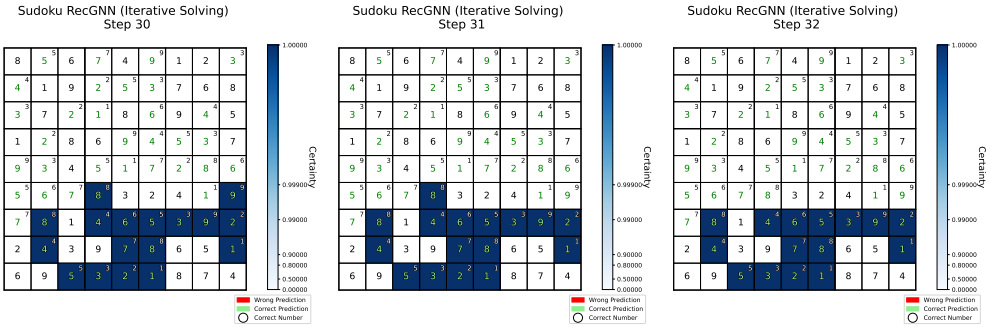

Sudoku RecGNN (Iterative Solving)
Step 20
Sudoku RecGNN (Iterative Solving)
Step 21
Sudoku RecGNN (Iterative Solving)
Step 22
Certainty
Wrong Prediction
Correct Prediction
Correct Number

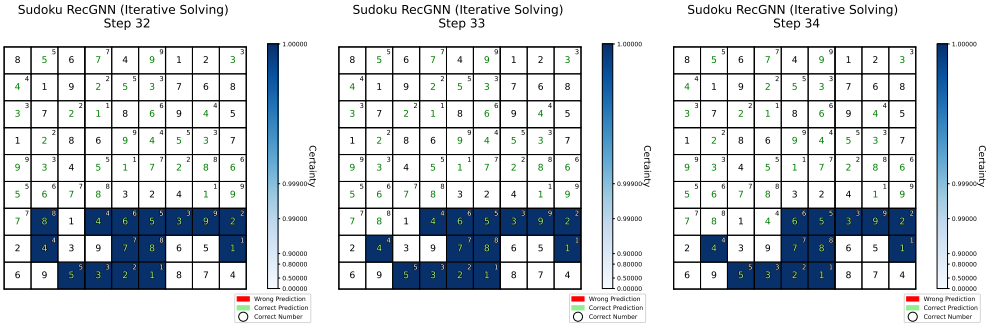

Sudoku RecGNN (Iterative Solving)
Step 22
Sudoku RecGNN (Iterative Solving)
Step 23
Sudoku RecGNN (Iterative Solving)
Step 24
Certainty
Wrong Prediction
Correct Prediction
Correct Number

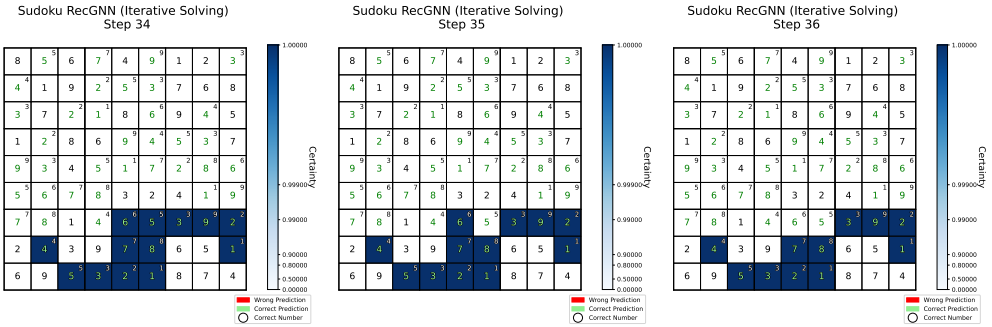

Sudoku RecGNN (Iterative Solving)
Step 24
Sudoku RecGNN (Iterative Solving)
Step 25
Sudoku RecGNN (Iterative Solving)
Step 26
Certainty
Wrong Prediction
Correct Prediction
Correct Number

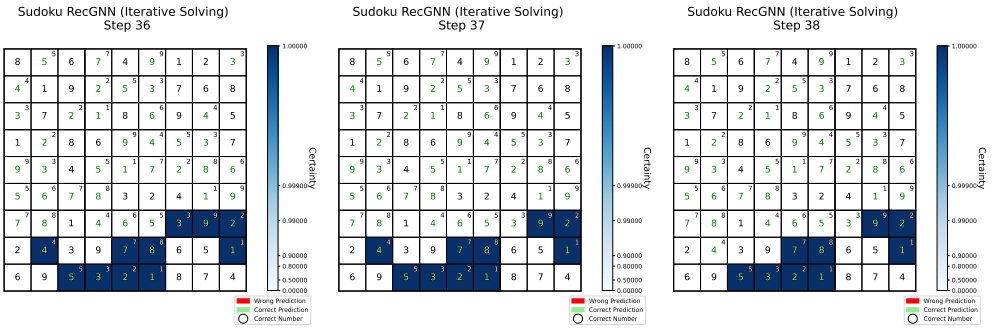

Sudoku RecGNN (Iterative Solving)
Step 26
Sudoku RecGNN (Iterative Solving)
Step 27
Sudoku RecGNN (Iterative Solving)
Step 28
Certainty
Wrong Prediction
Correct Prediction
Correct Number

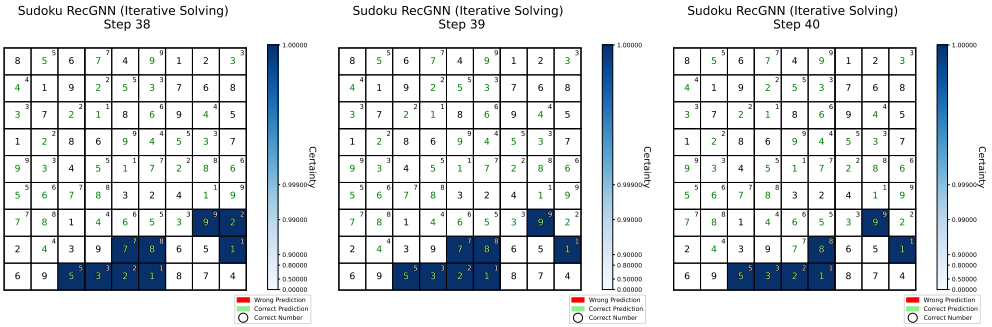

Sudoku RecGNN (Iterative Solving)
Step 28
Sudoku RecGNN (Iterative Solving)
Step 29
Sudoku RecGNN (Iterative Solving)
Step 30
Certainty
Wrong Prediction
Correct Prediction
Correct Number

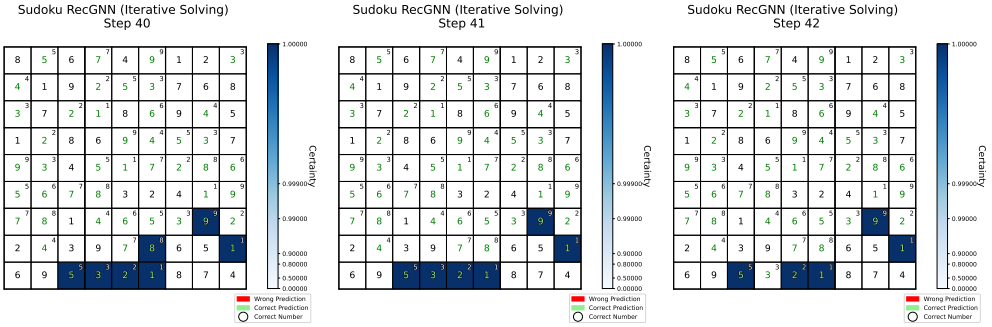

Sudoku RecGNN (Iterative Solving)
Step 30
Sudoku RecGNN (Iterative Solving)
Step 31
Sudoku RecGNN (Iterative Solving)
Step 32
Certainty
Wrong Prediction
Correct Prediction
Correct Number

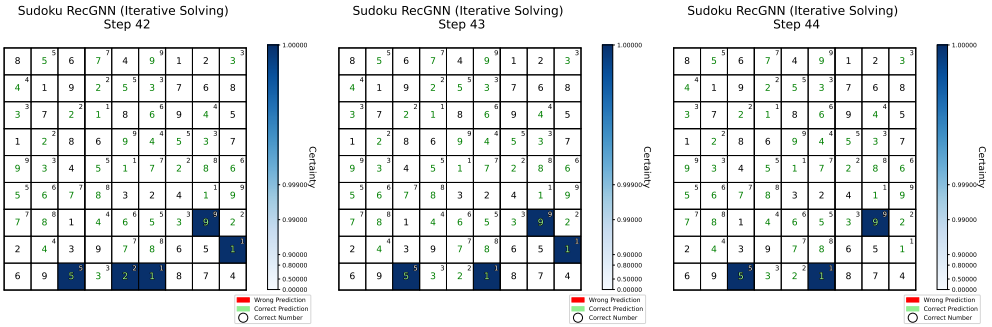

Sudoku RecGNN (Iterative Solving)
Step 32
Sudoku RecGNN (Iterative Solving)
Step 33
Sudoku RecGNN (Iterative Solving)
Step 34
Certainty
Wrong Prediction
Correct Prediction
Correct Number

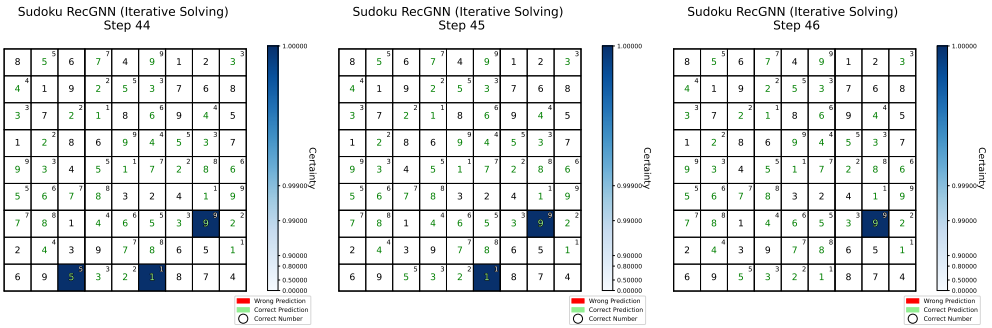

Sudoku RecGNN (Iterative Solving)
Step 34
Sudoku RecGNN (Iterative Solving)
Step 35
Sudoku RecGNN (Iterative Solving)
Step 36
Certainty
Wrong Prediction
Correct Prediction
Correct Number

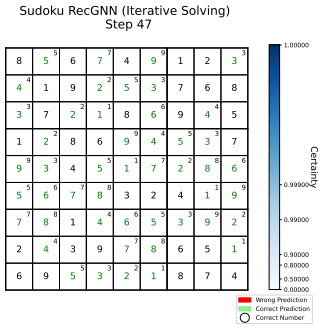

Sudoku RecGNN (Iterative Solving)
Step 36
Sudoku RecGNN (Iterative Solving)
Step 37
Sudoku RecGNN (Iterative Solving)
Step 38
Certainty
Wrong Prediction
Correct Prediction
Correct Number


Sudoku RecGNN (Iterative Solving)
Step 38
Sudoku RecGNN (Iterative Solving)
Step 39
Sudoku RecGNN (Iterative Solving)
Step 40
Certainty
Wrong Prediction
Correct Prediction
Correct Number

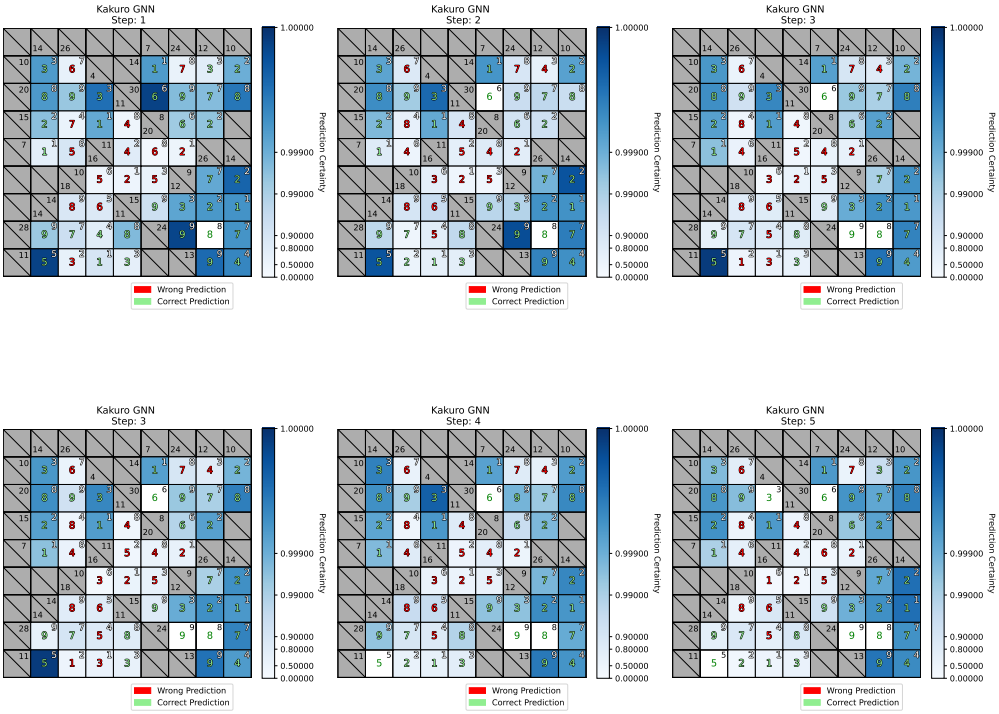

Sudoku RecGNN (Iterative Solving)
Step 40
Sudoku RecGNN (Iterative Solving)
Step 41
Sudoku RecGNN (Iterative Solving)
Step 42
Certainty
Wrong Prediction
Correct Prediction
Correct Number

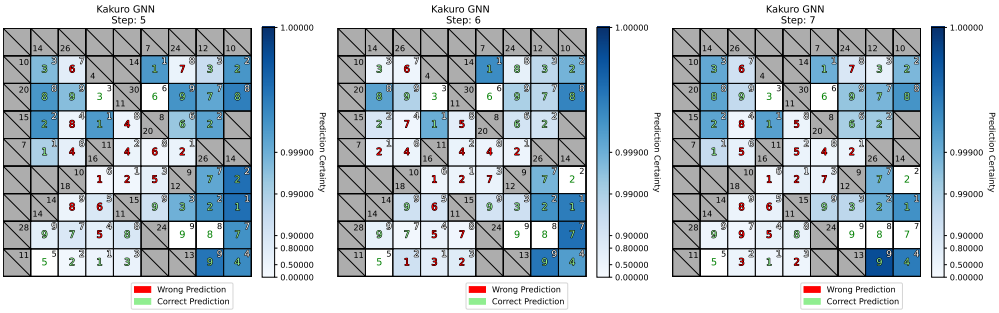

Sudoku RecGNN (Iterative Solving)
Step 42
Sudoku RecGNN (Iterative Solving)
Step 43
Sudoku RecGNN (Iterative Solving)
Step 44
Certainty
Wrong Prediction
Correct Prediction
Correct Number

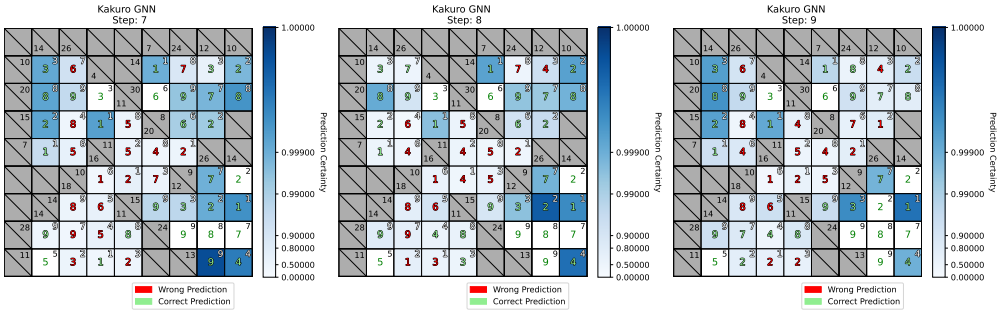

Sudoku RecGNN (Iterative Solving)
Step 44
Sudoku RecGNN (Iterative Solving)
Step 45
Sudoku RecGNN (Iterative Solving)
Step 46
Certainty
Wrong Prediction
Correct Prediction
Correct Number

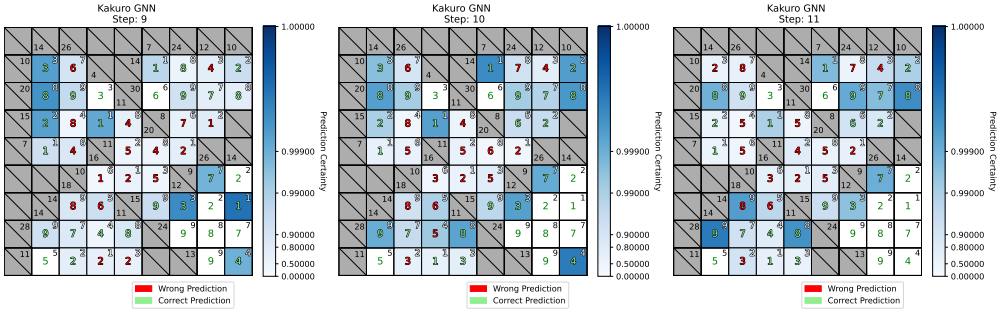


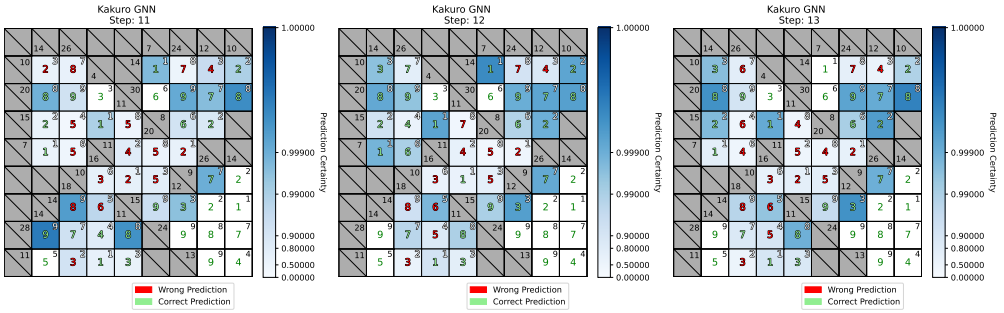
F.2 Kakuro GNN

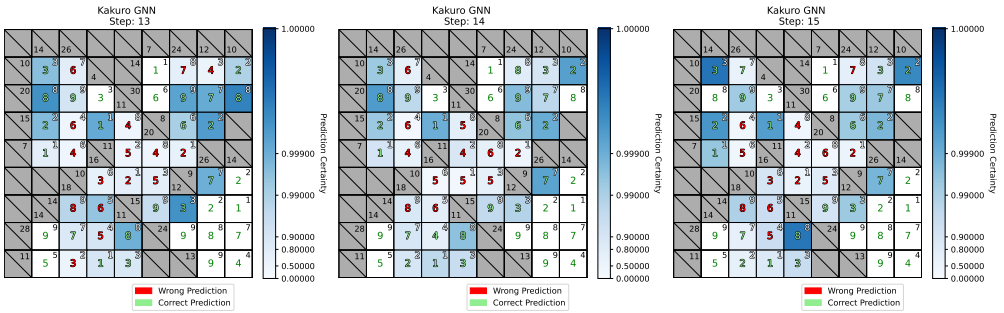


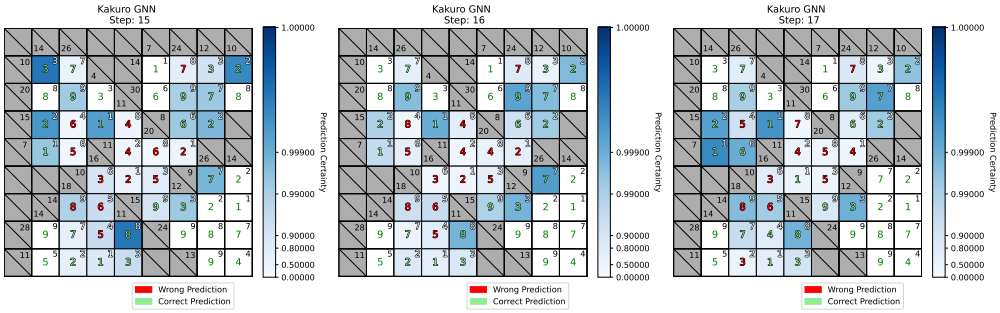

Kakuro GNN
Step: 5
Kakuro GNN
Step: 6
Kakuro GNN
Step: 7
Prediction Certainty
1.00000
0.99900
0.99000
0.90000
0.80000
0.50000
0.00000
Wrong Prediction
Correct Prediction

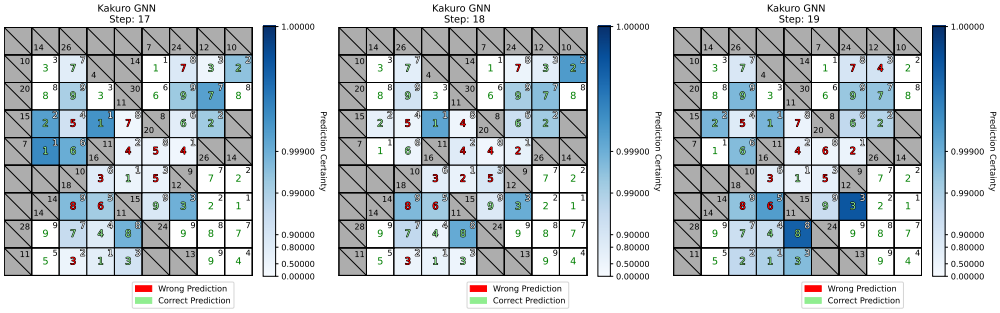

Kakuro GNN
Step: 7
Kakuro GNN
Step: 8
Kakuro GNN
Step: 9
Prediction Certainty
1.00000
0.99900
0.99000
0.90000
0.80000
0.50000
0.00000
Wrong Prediction
Correct Prediction

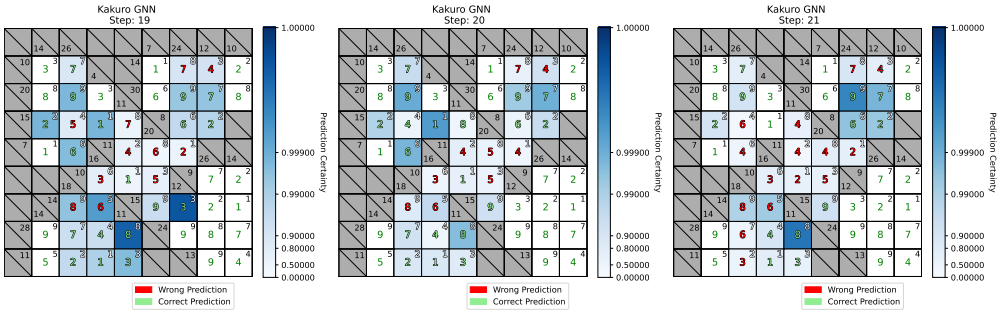

Kakuro GNN
Step: 9
Kakuro GNN
Step: 10
Kakuro GNN
Step: 11
Prediction Certainty
1.00000
0.99900
0.99000
0.90000
0.80000
0.50000
0.00000
Wrong Prediction
Correct Prediction

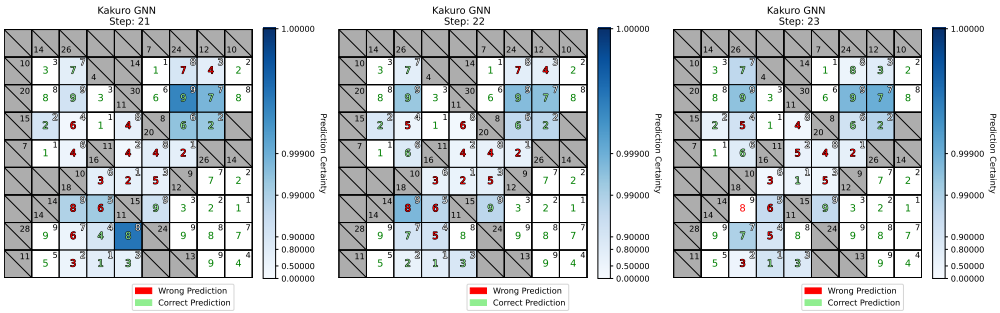

Kakuro GNN
Step: 11
Kakuro GNN
Step: 12
Kakuro GNN
Step: 13
Prediction Certainty
1.00000
0.99900
0.99000
0.90000
0.80000
0.50000
0.00000
Wrong Prediction
Correct Prediction

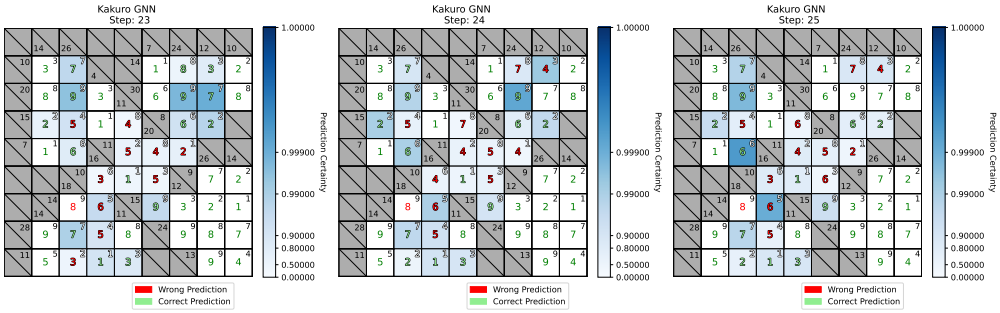

Kakuro GNN
Step: 13
Kakuro GNN
Step: 14
Kakuro GNN
Step: 15
Prediction Certainty
1.00000
0.99900
0.99000
0.90000
0.80000
0.50000
0.00000
Wrong Prediction
Correct Prediction

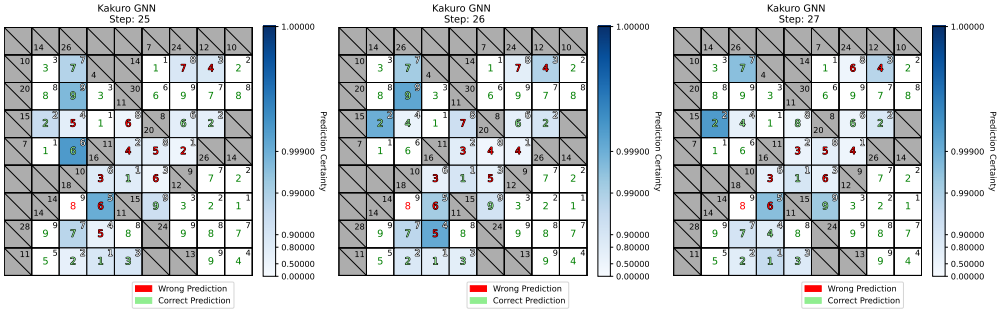

Kakuro GNN
Step: 15
Kakuro GNN
Step: 16
Kakuro GNN
Step: 17
Prediction Certainty
1.00000
0.99900
0.99000
0.90000
0.80000
0.50000
0.00000
Wrong Prediction
Correct Prediction

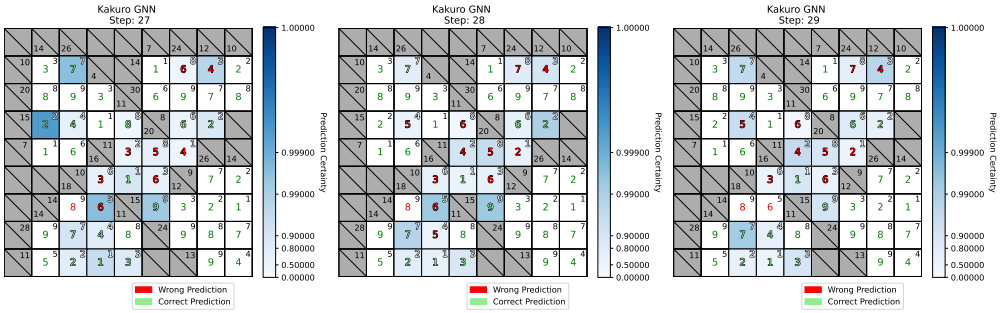

Kakuro GNN
Step: 17
Kakuro GNN
Step: 18
Kakuro GNN
Step: 19
Prediction Certainty
1.00000
0.99900
0.99000
0.90000
0.80000
0.50000
0.00000
Wrong Prediction
Correct Prediction

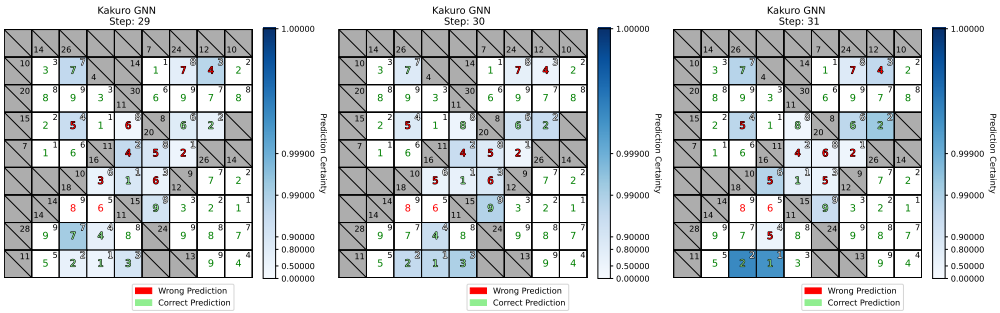

Kakuro GNN
Step: 19
Kakuro GNN
Step: 20
Kakuro GNN
Step: 21
Prediction Certainty
1.00000
0.99900
0.99000
0.90000
0.80000
0.50000
0.00000
Wrong Prediction
Correct Prediction

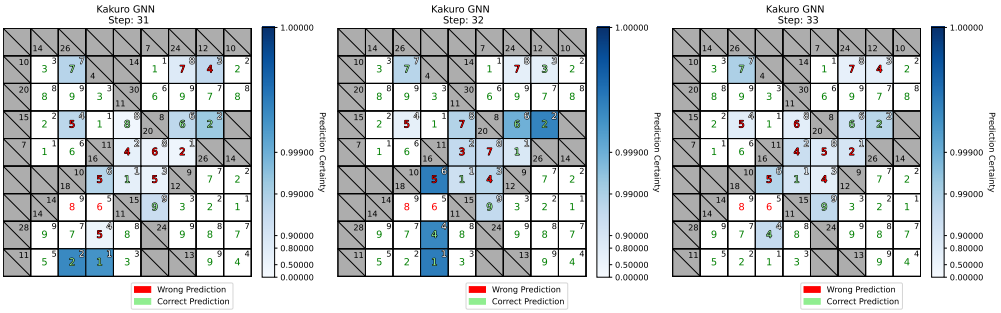

Kakuro GNN
Step: 21
Kakuro GNN
Step: 22
Kakuro GNN
Step: 23
Prediction Certainty
1.00000
0.99900
0.99000
0.90000
0.80000
0.50000
0.00000
Wrong Prediction
Correct Prediction

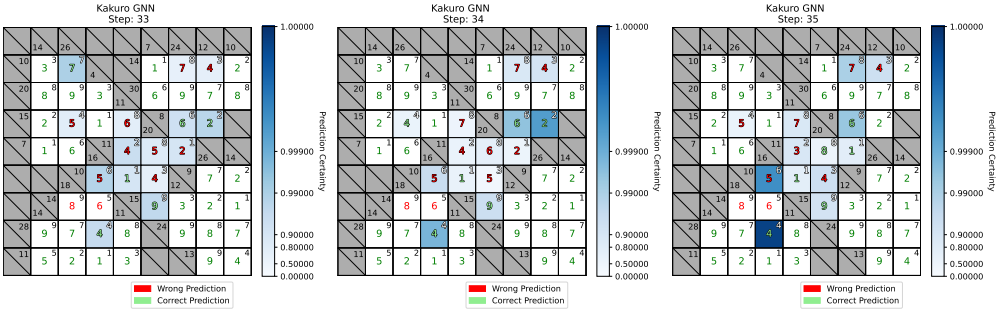

Kakuro GNN
Step: 23
Kakuro GNN
Step: 24
Kakuro GNN
Step: 25
Prediction Certainty
1.00000
0.99900
0.99000
0.90000
0.80000
0.50000
0.00000
Wrong Prediction
Correct Prediction

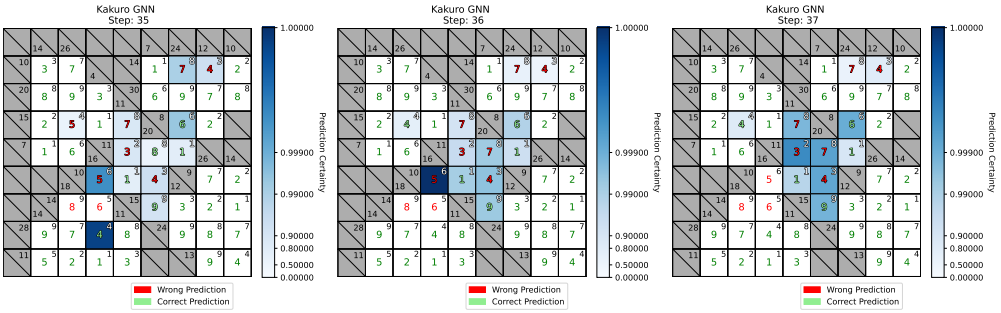

Kakuro GNN
Step: 25
Kakuro GNN
Step: 26
Kakuro GNN
Step: 27
Prediction Certainty
1.00000
0.99900
0.99000
0.90000
0.80000
0.50000
0.00000
Wrong Prediction
Correct Prediction

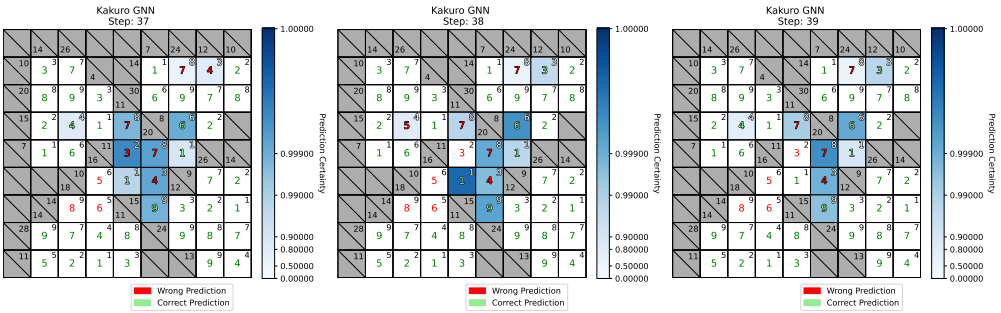

Kakuro GNN
Step: 27
Kakuro GNN
Step: 28
Kakuro GNN
Step: 29
Prediction Certainty
1.00000
0.99900
0.99000
0.90000
0.80000
0.50000
0.00000
Wrong Prediction
Correct Prediction

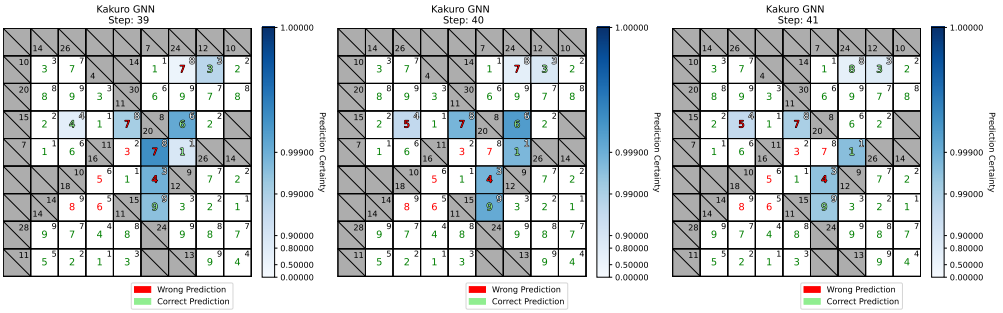

Kakuro GNN
Step: 29
Kakuro GNN
Step: 30
Kakuro GNN
Step: 31
Prediction Certainty
1.00000
0.99900
0.99000
0.90000
0.80000
0.50000
0.00000
Wrong Prediction
Correct Prediction

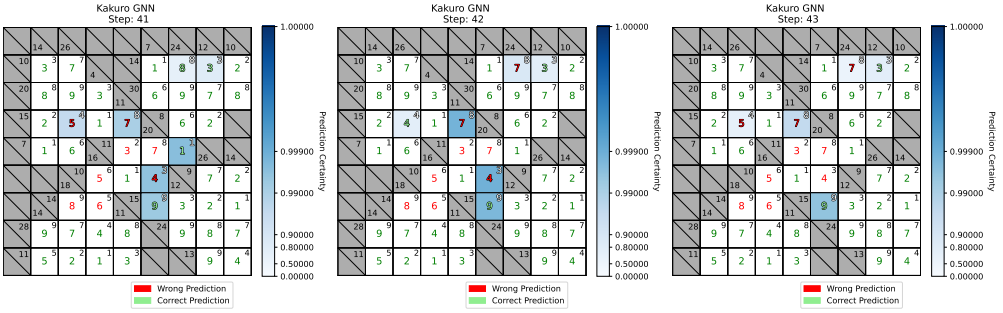

Kakuro GNN
Step: 31
Kakuro GNN
Step: 32
Kakuro GNN
Step: 33
Prediction Certainty
1.00000
0.99900
0.99000
0.90000
0.80000
0.50000
0.00000
Wrong Prediction
Correct Prediction

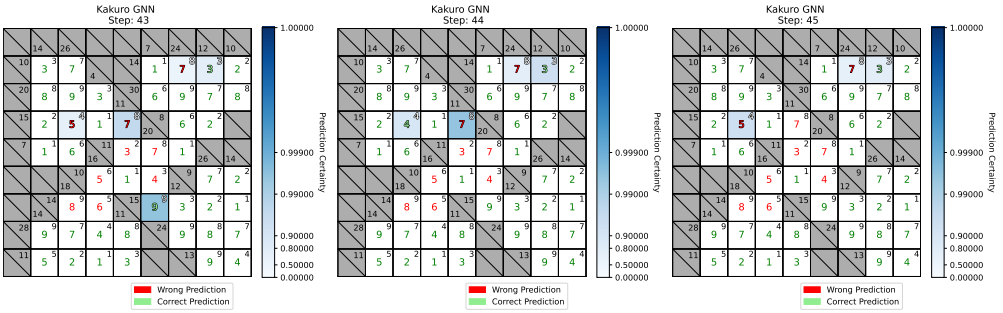

Kakuro GNN
Step: 33
Kakuro GNN
Step: 34
Kakuro GNN
Step: 35
Prediction Certainty
1.00000
0.99900
0.99000
0.90000
0.80000
0.50000
0.00000
Wrong Prediction
Correct Prediction

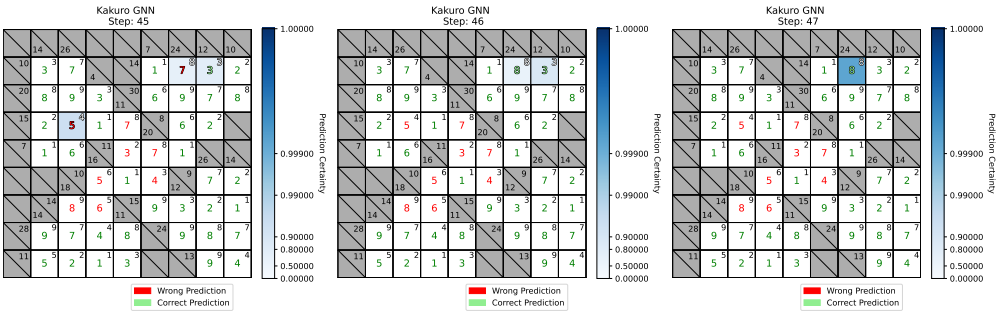

Kakuro GNN
Step: 35
Kakuro GNN
Step: 36
Kakuro GNN
Step: 37
Prediction Certainty
1.00000
0.99900
0.99000
0.90000
0.80000
0.50000
0.00000
Wrong Prediction
Correct Prediction

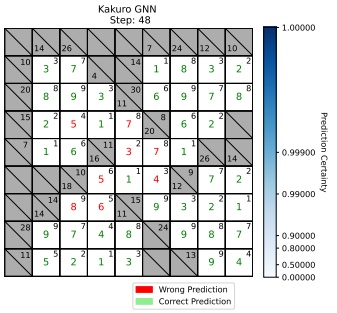

Kakuro GNN
Step: 37
Kakuro GNN
Step: 38
Kakuro GNN
Step: 39
Prediction Certainty
1.00000
0.99900
0.99000
0.90000
0.80000
0.50000
0.00000
Wrong Prediction
Correct Prediction


Kakuro GNN
Step: 39
Kakuro GNN
Step: 40
Kakuro GNN
Step: 41
Prediction Certainty
1.00000
0.99900
0.99000
0.90000
0.80000
0.50000
0.00000
Wrong Prediction
Correct Prediction


Kakuro GNN
Step: 41
Kakuro GNN
Step: 42
Kakuro GNN
Step: 43
Prediction Certainty
1.00000
0.99900
0.99000
0.90000
0.80000
0.50000
0.00000
Wrong Prediction
Correct Prediction


Kakuro GNN
Step: 43
Kakuro GNN
Step: 44
Kakuro GNN
Step: 45
Prediction Certainty
1.00000
0.99900
0.99000
0.90000
0.80000
0.50000
0.00000
Wrong Prediction
Correct Prediction


Kakuro GNN
Step: 45
Kakuro GNN
Step: 46
Kakuro GNN
Step: 47
Prediction Certainty
1.00000
0.99900
0.99000
0.90000
0.80000
0.50000
0.00000
Wrong Prediction
Correct Prediction


Kakuro GNN
Step: 48
Prediction Certainty
1.00000
0.99900
0.99000
0.90000
0.80000
0.50000
0.00000
Wrong Prediction
Correct Prediction