University of Padova

Department of Mathematics "Tullio Levi-Civita"

Master Thesis in Data Science

Graph Neural Networks for learning domain dependent heuristics in automated planning; enhacing the GOOSE framework

Supervisor
Prof. Nicolò Navarin
University of Padova

Co-supervisor

Master Candidate
Maren Hoschek

Student ID
2105030

Academic Year
2023-2024

Abstract

In automated planning, heuristics are used to estimate the cost or effort required to reach a goal from a given state within a planning problem. This thesis focuses on enhancing GOOSE, a learning-based planning framework that provides various graph representations and heuristics for heuristic search with a planner. GOOSE models the heuristic function using different graph learning approaches. The goal of this thesis is to improve GOOSE's performance on the IPC 2023 benchmarking dataset by refining existing graph learning methods and exploring new ones. Specifically, the improvements target better handling of domain drift as problems transition from easy to medium, and medium to hard. This work emphasizes modeling domain-dependent heuristics. To enhance existing models, we fine-tuned current GNNs through retraining and explored a multi-heuristic search approach that combines these GNNs with the h^{ff} heuristic. Additionally, we combined retraining with multi-heuristic search. Regarding changes to the graph learning approach, we incorporated attention mechanisms into the framework by using relational graph attention layers in the GNNs that model the heuristic function. Furthermore, we experimented with a novel approach called masked attention for graphs, which does not rely on message passing for heuristic modeling. We evaluated our models based on the number of solved problems, the number of nodes expanded by the planner, and the cost of the resulting plans. The results indicate that multi-heuristic search combined with retraining is most effective for managing domain drift from easy to medium problems, while the existing WL-GPR kernel models in GOOSE remain the best for hard problems. For the attention-based approaches, we encountered challenges related to limited training data and longer inference times compared to the GNN models due to their increased complexity. Overall, this thesis aims to establish a strong foundation for enhancing domain-dependent search within the GOOSE framework and suggests directions for future research.

Contents

Listing of figures

Listing of tables

Listing of acronyms

CNN	Convolutional Neural Network
FDR	Finite Domain Representation
FLG	FDR learning graph
GAT	Graph Attention Network
GBFS	Greedy Best-First Search
GCNN	Graph Convolutional Neural Network
GNN	Graph Neural Network
GOOSE	Graphs Optimised for Search Evaluation
GPR	Gaussian Process Regression
GRNN	Graph Recurrent Neural Network
ILG	Instance learning graph
IPC	International Planning Competition
LLG	Lifted learning graph
MAG	Masked attention for graphs
MPNN	Message Passing Neural Network
MSE	Mean Squared Error
PDDL	Planning Domain Definition Language
ReLU	Rectified Linear Unit
RGAT	Relation Graph Attention Network
RGNN	Relational Graph Neural Network
SLG	STRIPS learning graph
STRIPS	Stanford Research Institute Problem Solver
SVR	Support Vector Regression
WL	Weisfeiler Leman algorithm

1
Introduction

1.1 Background and Need

Planning is an important activity in various domains, ranging from everyday life to complex industrial processes. It involves deciding on a sequence of actions to achieve specific goals, ensuring efficient and effective use of resources. In organizational contexts, planning is crucial for project management, strategic decision-making, and operational efficiency. It helps in anticipating potential challenges, optimizing resource allocation, and aligning efforts towards achieving desired outcomes. The need for planning arises in environments where multiple variables and uncertainties must be managed to achieve specific objectives. This necessity spans across fields such as logistics, manufacturing, healthcare, and robotics [10].

Automated planning, a subfield of artificial intelligence, addresses the complexities associated with manual planning by utilizing computational techniques to generate plans [11]. This approach is particularly valuable in scenarios where the planning tasks are too intricate or time-consuming for humans to handle them efficiently. Automated planning systems can process large amounts of data, evaluate numerous possible action sequences, and select the most optimal plan based on predefined criteria. This capability is essential in applications such as autonomous vehicles, robotic control, space missions, and complex scheduling problems. The automation of planning tasks not only enhances accuracy and efficiency but also frees up human resources to focus on more strategic and creative activities. In this context, a planner is a software tool that takes a formal description of a planning problem as input and generates a sequence of actions that achieve a specified goal. Planners use various algorithms and heuristics to efficiently search through the space of possible actions, identifying the most effective sequence to reach the desired outcome. They are integral components of automated planning systems, enabling them to solve complex problems that require detailed and coordinated action plans [12].

Heuristics play a pivotal role in automated planning by guiding the search process towards optimal solutions. A heuristic is a problem-solving approach that employs practical methods to reach decisions and solve problems more quickly when optimal solutions are not feasible [10]. In the context of automated planning, heuristics are used to evaluate the desirability of different states or actions, helping the system prioritize which paths to explore. They are essential for reducing the search space and computational effort required to find a solution, especially in complex and dynamic environments. By estimating the cost or distance to the goal, heuristics enable the planning algorithm to make informed decisions about the most promising actions to pursue [12]. Due to the advantages of using heuristics, there are various planners that support heuristic search. Examples include the Fast Downward planner [1], LAMA [13], and Metric-FF [14].

Heuristics in automated planning are approached through various strategies to enhance their effectiveness. One common approach is the use of domain-independent heuristics, which do not rely on specific knowledge about the problem domain but rather utilize general principles applicable to a wide range of problems. Another strategy involves domain-specific heuristics, which leverage detailed knowledge about the specific characteristics of the problems to provide more accurate and efficient guidance [12]. Additionally, heuristic functions can be derived through machine learning techniques, where the system learns from past experiences and data to improve its heuristic evaluations. The development and refinement of heuristics are ongoing areas of research, aiming to balance the trade-off between computational efficiency and the quality of the solutions generated [15] [16] [17].

One notable framework for automated planning that utilizes machine learning to learn heuristic functions is GOOSE (Graphs Optimized for Search Evaluation) [6] [18]. GOOSE provides both graph neural networks and kernel-based methods for learning heuristic functions, which can then be employed by a planner within a heuristic search algorithm. The GOOOSE framework has demonstrated the capability to learn both domain-dependent and domain-independent heuristics. However, GOOSE's performance varies depending on the type of planning problem it is applied to. Despite the substantial research and development that has already been conducted to enhance the GOOSE framework, this thesis aims to further improve its performance on a commonly used planning benchmarking dataset [19]. This involves exploring new techniques and strategies to refine the learning of the heuristic functions, thereby enhancing the overall efficiency and effectiveness of the GOOSE framework in solving complex planning problems.

1.2 Statement of the Problem

The primary aim of this study is to enhance the performance of GOOSE, a learning-based framework designed for classical automated planning. GOOSE leverages Graph Neural Networks (GNNs) to model heuristic functions, which are crucial in guiding the planner towards promising directions when solving a planning problem. A more effective heuristic function improves the efficiency of the search process of the planner and by this enhances its overall performance. Consequently, this research focuses on optimizing the GNN models used as heuristic functions within the GOOSE framework.

The GNNs in GOOSE perform graph-level predictions and are frequently called upon by the planner during the search process. Depending on the complexity and size of the planning problem, the planner may invoke the heuristic function millions of times. This high frequency underscores the importance of both the accuracy and the evaluation speed of the heuristic function for the planner's performance. One of the main challenges is the domain drift between the training and testing data concerning the difficulty of the problems. In past studies involving GOOSE, this drift has led to suboptimal performance on previously unseen difficulties during training. Additionally, there are specific domains of planning problems where the expressive capacity of GNNs is limited in learning effective heuristic functions.

These challenges can hinder the overall effectiveness of the planning process. This study aims to address these issues, with a primary focus on the domain drift challenge, thereby improving the overall efficiency and effectiveness of the GOOSE framework.

1.3 Methodology

This study utilizes the dataset from the learning track of the 2023 International Planning Competition (IPC) [19], which was also employed in a previous study involving GOOSE. The dataset comprises 10 distinct planning domains, each providing up to 99 planning problems for training and 90 problems for testing. Training problems are categorized as easy, while the test dataset is divided into 30 easy, 30 medium, and 30 hard problems. The difficulty of a planning problem is roughly indicated by the number of different objects involved, with easy problems having fewer objects than hard problems. All planning problems are defined in the Planning Domain Definition Language (PDDL) [20]. The transformation of these planning problems from PDDL to graphs for graph learning is performed by the existing graph transformations in the GOOSE framework.

The models are trained on the training dataset, which is further split into training and validation subsets during the training process, using an MSE loss function to fit the models. In this study we train all models in a domain dependent setting, meaning the models are trained on planning problems of the same problem domain as they are indented to be tested on. Once trained, the models are saved and integrated with the FastDownward planner [1]. FastDownward is a heuristic planner capable of utilizing custom search algorithms and heuristic functions, with the GOOSE framework already providing integration for its trained models. During the testing phase, the FastDownward planner uses the previously trained models as heuristics to solve the testing problems. The performance of the different models as heuristics is evaluated based on the following metrics:

- **Number of solved problems in the testing dataset:** This metric indicates the number of problems for which the FastDownward planner finds a plan using a given model as a heuristic. Solving problems is the primary goal in automated planning, making this the most important metric.
- **Number of expanded nodes when solving a problem:** This metric reflects how many different states the planner explored during the search using the heuristic. Fewer expanded nodes suggest a more effective heuristic.
- **Plan cost of the found solution:** This metric measures the cost of the plan returned by the planner, as the planner does not always guarantee finding the optimal solution. Heuristics that result in lower costs are preferred.

The experiments are divided into two main categories. The first set of experiments aims to enhance the performance of the currently used GNNs [3] in GOOSE through fine-tuning and multi-heuristic search. The second set involves architectural changes to the models, including the use of RGAT models [21] and a novel graph learning approach called Masked Attention for Graphs (MAG) [5]. Additionally, we employ a set of baseline heuristics, consisting of common heuristics in automated planning and the current GOOSE heuristics, against which all models are compared. Finally, we evaluate all the experiments to determine which approach yields the best results. In the evaluation, we specifically focus on the performance of the medium and hard testing problems, as these difficulties present the greatest opportunities for improvement based on previous GOOSE studies.

The complete thesis is structured as follows. chapter 2 presents the foundational theories of automated planning and graph neural networks. It also introduces the GOOSE framework and reviews related works. chapter 3 details the experiments conducted in this thesis and describes the dataset used for these experiments. chapter 4 presents the results of the experiments described in chapter 3, along with a discussion of these findings. Finally chapter 5 provides concluding remarks and suggests possible directions for future research. The source code for the enhanced version of GOOSE, along with instructions on how to train and run the various models to replicate the experiments in this thesis, as well as the trained models and the results discussed herein, are available on GitHub [*].

[*] `https://github.com/theMaren/goose_thesis.git`

2 Background

2.1 Automated Planning

Automated planning is a branch of artificial intelligence that deals with creating step-by-step plans to achieve specific goals, taking into account the current situation and the effects of potential actions. These actions are carried out by an agent, which could be software or a robot. Automated planning is particularly useful when the agent operates in a dynamic environment that changes unpredictably [11]. For instance, in a rescue operation after a natural disaster, there are many people involved and complex transportation networks to consider. The plan must be flexible and adapt to the current state of the disaster. Software systems that automatically generate such plans are called planners or planning systems. There are three main types of planners discussed in the literature:

- **Domain-specific planners**: These planners are designed or adjusted for a specific planning domain. They do not work well, if at all, in other domains.
- **Domain-independent planners**: These planners are intended to work in any planning domain. However, in practice, they need some restrictions on the type of planning domain they can handle.
- **Configurable planners**: These are domain-independent planning engines. They require input on how to solve problems in certain domains [12].

Creating a plan can be very complex and requires advanced reasoning, which planning systems need to simulate. Therefore, automated planning systems use AI to help with this task [11]. Automated planning has been a research topic for over 40 years. Early work, such as the Stanford Research Institute Problem Solver (STRIPS), focused on representing planning problems in a formal way that computers could process [22]. Today, the focus is on planners that exploit deep neural networks or reinforcement learning, like AlphaGo [23] and AlphaZero [24], which master games like Go and Chess.

2.1.1 Planning Tasks

A classical planning task [25] is a state transition model $\Pi = \langle S, A, s_0, G \rangle$ where:

- S is a finite set of states
- A is a finite set of actions
- s_0 is the known initial state
- G is a non-empty set of goal states
- Each action $a \in A$ is a function $a : S \to S \cup \perp$ mapping a state s in which the action is applicable to its successor $a(s)$, and states in which it is not applicable to $\perp$

Each action has a cost denoted as $c(a) \in \mathbb{N}$. In this context, a solution or plan is a sequence of actions $\pi = \langle a_1, \dots, a_n \rangle$ such that for every $i \in \{1, \dots, n\}$, $s_i = a_i(s_{i-1}) \neq \perp$ and $s_n \in G$. Essentially, a plan consists of a series of applicable actions that transition the initial state to a goal state. The total cost of the plan π is calculated as $c(\pi) = \sum_{i=1}^{n} c(a_i)$. A planning task is considered solvable if there is at least one viable plan. An optimal plan is defined as one with the lowest possible cost, and the model's cost is that of the optimal plan. Often, a simple cost structure is used where every action cost $c(a, s)$ equals 1. In this scenario, the plan's cost corresponds to its length, making the shortest plans the optimal ones [25].

To illustrate the concept of a classical planning task, consider the following simple scenario involving three blocks (A, B, and C) which can be placed on a table or on top of each other. In this scenario, the set of states S includes all possible configurations of the blocks on the table and on each other. The set of actions A consists of $\{\text{Stack}(x, y), \text{Unstack}(x, y)\}$, where x and y are blocks.

The initial state s_0 describes the current position of the blocks, which in our example is defined as: $s_0 = \{$on_table(B), on_table(C), on(B,A)$\}$. The goal state G describes the block ordering we would like to achieve and is defined as $G = \{$on_table(A), on_table(B), on(C,B)$\}$. To transition from the initial state to the goal state, we apply a sequence of actions. The required actions are as follows: First, we unstack Block B from Block A and place it on the table. Next, we stack Block C on Block B. Figure 2.1 illustrates the transition from the initial state to the goal state.The sequence of actions forms the plan $\pi = \langle \text{Unstack}(B, A), \text{Stack}(C, B) \rangle$. Assuming each action has a cost of 1, the total cost of the plan is $c(\pi) = 1 + 1 = 2$, meaning plan π successfully achieves the goal with minimal cost.

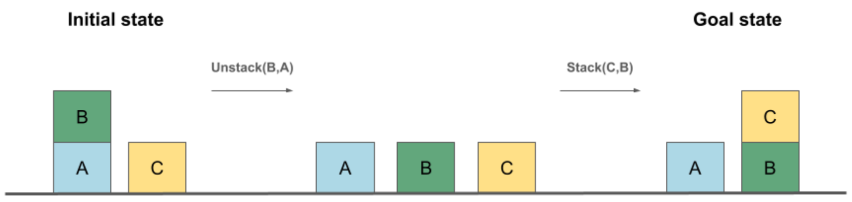


Figure 2.1: Example of a planning task in which blocks need to be stacked in a specific order. By unstacking Block B from Block A and then stacking Block C onto Block B, the goal state is reached

2.1.1.1 STRIPS Planning Task

STRIPS (Stanford Research Institute Problem Solver) is a formal language for planning in artificial intelligence. STRIPS was developed to address the need for a systematic and automated way to generate plans for autonomous systems. A STRIPS planning task [22] is formally defined as a tuple $\Pi = \langle P, A, s_0, G\rangle$ where:

- P is a finite set of propositions (or facts)
- A is a finite set of actions
- $s_0 \subseteq P$ is the known initial state
- $G \subseteq P$ is the goal condition
- A state s is a subset of P and is a goal state if $G \subseteq s$
- An action $a \in A$ is a tuple $\langle \text{pre}(a), \text{add}(a), \text{del}(a)\rangle$ with $\text{pre}(a), \text{add}(a), \text{del}(a) \subseteq P$ and $\text{add}(a) \cap \text{del}(a) = \emptyset$

In comparison to the classical planning task defined in subsection 2.1.1 the set of states S is replaced by the set of propositions P in the STRIPS planning task definition. Propositions (or facts) are individual statements about the world. They represent specific conditions and can be true or false independently of other propositions. States are collections of propositions. A state is a snapshot of the world at a particular moment, described by the set of all propositions that are true at that moment [12]. In a STRIPS planning task, actions have preconditions (pre), positive effects (add), and delete effects (del). The preconditions of an action a are the propositions that must be true for the action to be executed. The add effects of an action are the propositions that become true after the action is executed, while the delete effects are the propositions that become false. Each action has an associated cost $c(a) \in \mathbb{R}$. An action is applicable in a state s if $\text{pre}(a) \subseteq s$, and it results in the successor state $s' = (s \setminus \text{del}(a)) \cup \text{add}(a)$ [22].

2.1.1.2 FDR Planning Task

The Finite Domain Representation (FDR) was developed to efficiently handle a broader range of planning problems. It offers increased expressiveness, enables more efficient planning algorithms by leveraging the structure of finite domains, and provides greater flexibility by allowing partial variable assignments. A FDR planning task [26] is formally defined as the tuple $\Pi = \langle \mathcal{V}, A, s_0, s_\star \rangle$ where:

- $\mathcal{V}$ is a finite set of state variables v, each with a finite domain D_v
- A is a finite set of actions of the form $a = \langle \mathrm{pre}(a), \mathrm{eff}(a) \rangle$ where $\mathrm{pre}(a)$ and $\mathrm{eff}(a)$ are partial variable assignments
- the intial state s_0 is a total variable assignment
- the goal condition $s_\star$ is a partial variable assignment

A partial variable assignment consists of a set of facts, with each variable appearing no more than once. Each fact is expressed as a pair $\langle v, d \rangle$, where $v \in \mathcal{V}$ and $d \in D_v$. When each variable is included exactly once, the assignment is referred to as a total variable assignment. An action a can be applied in a state s if $\mathrm{pre}(a) \subseteq s$, resulting in the successor state $s' = (s \cup \mathrm{eff}(a)) \setminus \{\langle v, d \rangle \in s \mid \exists d' \in D_v, \langle v, d' \rangle \in \mathrm{eff}(a) \land d \neq d'\}$ [6].

The primary difference between an FDR planning task and standard or STRIPS planning tasks is the use of variable assignments, where each variable can take values from a finite domain. In FDR planning, actions are defined by preconditions and effects using these variable assignments, unlike the propositions used in STRIPS. Considering the block stacking example from section subsection 2.1.1, we define a variable for the position of each block: $\mathcal{V} = \{\mathrm{pos}_A, \mathrm{pos}_B, \mathrm{pos}_C\}$. Each of these variables has a finite domain. For instance, $D_{\mathrm{pos}_A} = \{\mathrm{table}, B, C\}$, $D_{\mathrm{pos}_B} = \{\mathrm{table}, A, C\}$, and $D_{\mathrm{pos}_C} = \{\mathrm{table}, A, B\}$. These variables are then used to define actions and states. For example, the unstack action from Figure 2.1 can be defined as follows:

$$\begin{aligned} \mathrm{Unstack}(B, A) : \mathrm{pre}(\mathrm{Unstack}(B, A)) &= \{\langle \mathrm{pos}_B, A \rangle, \langle \mathrm{pos}_A, \mathrm{table} \rangle\}, \\ \mathrm{eff}(\mathrm{Unstack}(B, A)) &= \{\langle \mathrm{pos}_B, \mathrm{table} \rangle\} \end{aligned}$$

States are also described using variable assignments. The initial state shown in Figure 2.1 would be described as follows:

$$s_0 = \{\langle \mathrm{pos}_A, \mathrm{table} \rangle, \langle \mathrm{pos}_B, A \rangle, \langle \mathrm{pos}_C, \mathrm{table} \rangle\}$$

The use of variable assignments allows for a more structured and compact representation, particularly when dealing with domains that naturally fit into variable-value pairs.

2.1.1.3 Lifted Planning Task

Traditional grounded planning approaches enumerate all possible states and actions explicitly, which can be computationally expensive and infeasible for large problems. Lifted planning, on the other hand, works at a higher level of abstraction making the planning process more scalable and flexible. A lifted planning task [27] is formally defined as the tuple $\Pi = \langle \mathcal{P}, \mathcal{O}, \mathcal{A}, s_0, G \rangle$ where

- $\mathcal{P}$ is a set of first-order predicates
- $\mathcal{O}$ is a set of objects
- s_0 is the initial state
- G is the goal condition

A predicate $P \in \mathcal{P}$ can have parameters $x_1, \ldots, x_{n_P}$, where $n_P \in \mathbb{N}$ is the number of parameters, and n_P depends on the predicate. Some predicates might not have any parameters. A predicate with n parameters is called an n-ary predicate. When we assign objects from $\mathcal{O}$ or other variables to the parameters x_i, we instantiate the predicate. A predicate where all parameters are assigned objects is called a ground proposition. The initial state and goal condition are both sets of these ground propositions [6].

An action schema $a \in \mathcal{A}$ is a tuple $\langle \Delta(a), \text{pre}(a), \text{add}(a), \text{del}(a) \rangle$. Here, $\Delta(a)$ is a set of parameter variables, and $\text{pre}(a)$, $\text{add}(a)$, and $\text{del}(a)$ are sets of predicates from $\mathcal{P}$, instantiated with either parameter variables or objects from $\Delta(a) \cup \mathcal{O}$. Just like predicates, an action schema with n parameter variables is called an n-ary action schema. When all variables in an action schema are assigned objects, it becomes an action.

2.1.2 Heuristic Functions

Automated planning systems utilize heuristic functions to efficiently guide the search process towards goal states, enhancing computational efficiency by reducing the number of states evaluated. A heuristic function, represented as h, provides an estimate $h(s)$ of the minimum cost $h^*(s)$ required to reach a goal state from a given state s. The heuristic h is admissible if $0 \leq h(s) \leq h^*(s)$ for all states s, indicating that $h(s) = 0$ when s is a goal state. When a heuristic function can be computed in polynomial time and decreases the number of nodes explored by the planning algorithm, the computational effort is considered worthwhile [10].

One of the most prominent techniques for developing heuristic functions is known as *relaxation*. In a planning domain $\Sigma = (S, A, \gamma)$, where S is the finite set of states, A is the finite set of actions, and $\gamma(s, a)$ is a transition function mapping each state s and action a to a set of states, and with a planning problem $\mathcal{P} = (\Sigma, s_0, g)$, relaxation involves easing certain constraints that define states, actions, and plans. This entails modifying limitations on when actions or plans can be applied, what goals they achieve, and increasing the costs of actions and plans. This process results in a *relaxed domain* $\Sigma' = (S', A', \gamma')$ and a *relaxed problem* $\mathcal{P}' = (\Sigma', s'_0, g')$, maintaining the property that for every solution π for $\mathcal{P}$, there exists a solution π' for $\mathcal{P}'$ such that $\text{cost}(\pi') \leq \text{cost}(\pi)$. By solving planning problems in Σ' using an algorithm, we can create a heuristic function for $\mathcal{P}$ as follows: for a given state $s \in S$, solve (Σ', s, g') and return the cost of the solution. If the algorithm consistently identifies optimal solutions, the heuristic function will be admissible [12].

Similar to domain representations, heuristic functions can be either domain-specific or domain-independent. Besides the optimal heuristic h^*, there exist other heuristics like the max-cost heuristic h_{max} [28], which estimates the cost to reach the goal by considering the most expensive sub-goal, and the additive cost heuristic h_{add} [28], which sums the costs of achieving all individual sub-goals. These alternative heuristics will not be discussed further, as the focus of the later experiments is to model the optimal heuristic h^*.

2.1.3 Fast Downward Planning System

The Fast Downward planning system is a classical planning system that utilizes heuristic forward search and hierarchical problem decomposition. Beyond supporting STRIPS planning, it can handle arbitrary formulae in operator preconditions and goal conditions. It also manages conditional and universally quantified effects as well as derived predicates (axioms). Fast Downward is a heuristic progression planner that computes plans by performing heuristic searches in the space of world states that are reachable from the initial state. The heuristic evaluator works in a hierarchical manner, solving planning tasks by recursively breaking them down. Starting from the top-level goals, the algorithm delves deeper into the causal graph until all remaining subproblems become basic graph search tasks. As shown in Figure 2.2 the planner addresses a planning task in three phases: translation, knowledge compilation, and search [1].

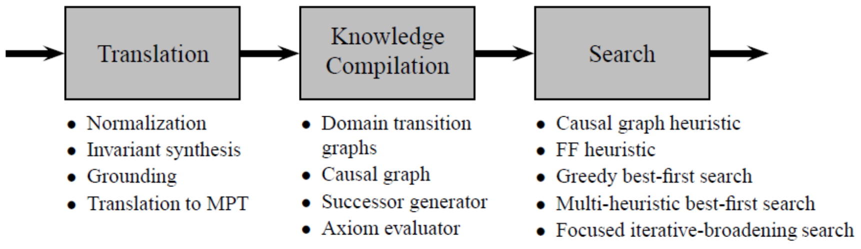


Figure 2.2: The three phases of Fast Downward's execution [1]

The translation phase takes planning domain and problem files in PDDL format and converts them into a nonbinary format, which is more suitable for hierarchical planning. It also normalizes and grounds axioms and operators. Most importantly, it uses invariant synthesis methods to find groups of related propositions that can be encoded as a single multi-valued variable. The output of this phase is a multi-valued planning task [1].

During the knowledge compilation phase, four essential data structures are created for the search process. Domain transition graphs illustrate how and when state variables can change their values. The causal graph depicts the hierarchical relationships between different state variables. The successor generator identifies which operators can be applied in a given state and lastly the axiom evaluator efficiently computes the values of derived variables [1].

In the search phase search algorithms for the actual planning are implemented. These search algorithms can use heuristic functions to enhance the planning process. When the planner was first released in 2006, it included two search algorithms that used heuristic evaluation functions. The first was the greedy best-first search algorithm using the causal graph heuristic. This heuristic estimates the cost of reaching a goal from a given search state by solving several subproblems of the planning task derived from the causal graph [1]. The second was the multiheuristic best-first search, a variant of the greedy best-first search that combined the causal graph heuristic with the FF heuristic a heuristic from the Hoffmanns's planning algorithm [29], improving the search efficiency by using multiple heuristics.

The Fast Downward planner is designed to allow the incorporation of new and custom search algorithms and heuristic functions. Over the years, more algorithms and heuristics have been integrated. Table 2.1 provides an overview of the currently available search algorithms and heuristic functions in the Fast Downward planner. It is important to note that not all search algorithms are heuristic-based, so not all of them can be combined with the listed heuristic functions.

	Fast Downward Planning System
Search Algorithms	A* search (eager), Eager best-first search, Greedy search (eager), Eager weighted A* search, Lazy enforced hill-climbing, Iterated search, Lazy best-first search, Greedy search (lazy), (Weighted) A* search (lazy)
Heuristics	Additive heuristic, Blind heuristic, Context-enhanced additive heuristic, Additive Cartesian CEGAR heuristic, Causal graph heuristic, FF heuristic, Goal count heuristic, h^m heuristic, Max heuristic, Landmark cost partitioning heuristic, Landmark sum heuristic, Landmark-cut heuristic, Merge-and-shrink heuristic, Operator-counting heuristic

Table 2.1: Search algorithms and heuristics used in the Fast Downward Planning System [7][8]

The Fast Downward planning system has demonstrated great impact within the field of automated planning, notably winning the classical track of the 4th International Planning Competition at ICAPS 2004. Over the years, numerous planners have been developed that extend or build upon the Fast Downward framework, many of which have received accolades themselves. For instance, the LAMA planner [13] enhances Fast Downward by employing finite-domain variables and multi-heuristic search, showcasing its robust capabilities. Delfi [30], a machine learning-based planner, selects from a portfolio of cost-optimal planners derived from Fast Downward, winning the optimal track at the 2018 International Planning Competition. Additionally, the merge-and-shrink framework [31] was integrated into

Fast Downward to compute abstraction heuristics for large transition systems, further expanding its utility. The Scorpion planning system [32] is another notable extension, incorporating additional search algorithms and utilities. The 2023 version of Scorpion was recognized as the runner-up in the optimal track at the 2023 International Planning Competition.

2.1.3.1 Fast Forward Heuristic

The Fast Forward (FF) heuristic h^{ff} [29] is a prominent heuristic used in planning algorithms, including the Fast Downward planner. The primary idea behind the FF heuristic is to estimate the cost to reach the goal from a given state by simplifying the problem, specifically by ignoring the delete effects of actions. In other words, it assumes that once an action adds a fact to the state, that fact remains true, regardless of any subsequent actions that might delete it. This simplification turns the planning problem into a relaxed version, making it easier to solve. The FF heuristic operates by constructing a relaxed planning graph, which is a layered graph representing the sequence of actions and states without considering the delete effects. From this graph, the heuristic extracts a relaxed plan, which serves as an approximation of the optimal solution. The length of this relaxed plan is used as the heuristic estimate of the cost to reach the goal from the current state.

By ignoring delete effects, the heuristic can sometimes overestimate the feasibility of reaching the goal. This can lead to suboptimal plans or longer computation times in cases where the relaxed plan significantly diverges from a real executable plan. Additionally, in domains with many negative interactions between actions, the heuristic might be less accurate, potentially misleading the planner [29]. Despite these limitations, the FF heuristic has proven to be robust across a wide range of planning domains, contributing to its popularity and widespread use in automated planning.

Because it is relatively quick in generating estimates, the FF heuristic allows planners to efficiently explore and navigate large state spaces. Its balance between accuracy and computational efficiency makes it a valuable heuristic for planners [1]. Consequently, not only does the Fast Downward planner incorporate the FF heuristic, but other planners also utilize it. Examples of such planners include LAMA [13], Metric-FF [14], and Marvin [33]. These planners leverage the FF heuristic to guide their search processes effectively, benefiting from its ability to provide quick and informative estimates.

2.2 Graph Learning and Graph Neural Networks

The representation of data in the form of graphs has gained significant popularity in recent years due to its flexible structure and ability to capture complex relationships and interactions. Graph data is utilized in various domains, including the modeling of molecular structures in bioinformatics [34], representing connections in social networks [35], and product recommendations in e-commerce systems [36].

Graph-structured data may consist of an arbitrary number of unordered nodes, each of which can have an arbitrary number of edges (connections) between them. Capturing information and relationships within these structures is challenging because most deep learning algorithms are designed to capture patterns in Euclidean data. Additionally, one core assumption of many deep learning algorithms—the independence of instances—does not hold for graphs, as the instances (nodes) can be linked with each other by edges [37]. Graph Neural Networks (GNNs) aim to address these challenges to be able to capture complex relationships in graph-structured data. The most common graph analytics tasks that GNNs tackle are:

- **Node Classification**: Aims to learn a model that, given a graph and a set of labels, predicts the labels of the unlabeled nodes in the graph.
- **Link Prediction**: Aims to learn a model that can predict unobserved links (edges) in a graph.
- **Graph Classification**: Operates at the level of the entire graph and aims to learn a model that, based on a given dataset of graphs, can predict the labels of unseen test graphs [38].

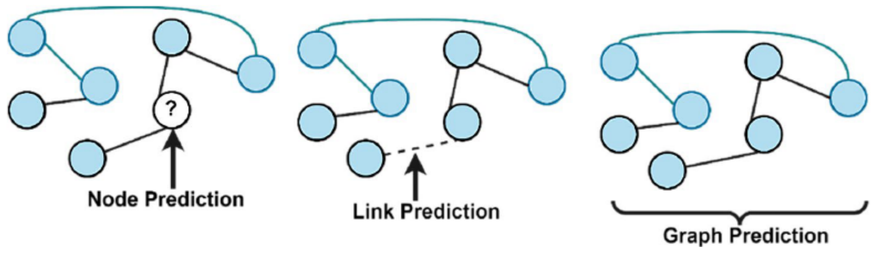


Figure 2.3: Node Level Prediction (left), Edge Level Prediction (middle) and Graph Level Prediction (right) [2]

The concept of graph neural networks was first introduced by Gori et al. [39], where node representations are learned by exchanging information with neighboring nodes iteratively until a stable equilibrium is reached. This initial graph neural network can be classified as a Graph Recurrent Neural Network (GRNN). GRNNs apply the same set of parameters recurrently over nodes in a graph to extract high-level node representations, and they recurrently exchange information with their neighbors until a stable equilibrium is reached [37]. GRNNs are capable of handling various graph types, including acyclic, cyclic, directed, and undirected graphs [40].

With the increasing success of convolutional neural networks (CNNs) in the computer vision domain, the concept of convolution has been applied to graph data, leading to the development of graph convolutional neural networks (GCNNs). The literature distinguishes two main classes of GCNNs: spectral-based GCNNs and spatial-based GCNNs.

Spectral-based GCNNs have their foundation in graph signal processing [41] [42]. These models operate by performing convolutions in the spectral domain, as the convolution between two signals can be achieved through the multiplication of their Fourier transforms. However, spectral-based GCNNs have some limitations. They can be applied exclusively to undirected graphs, generalize poorly to new graphs, and face scalability issues due to their reliance on the eigen-decomposition of the Laplacian matrix, whose dimensions increase with the number of nodes within the graph. Therefore, spectral GCNNs are not suitable for large-scale graphs [37].

Spatial GCNNs attempt to extend the conventional convolutional operator of CNNs to work with graph data. They use an approach inherited from GRNNs, aggregating information from neighboring nodes to update the current state of the nodes. Because of this exchange of information between nodes (messages), they are often referred to as message-passing neural networks (MPNNs). Spatial GCNNs are more flexible than spectral-based GCNNs, capable of handling heterogeneous graph types as inputs, and they scale better than spectral-based GCNNs [37]. Spatial based GCNNs are described in more detail in subsection 2.2.1.

Apart from GRNNs and GCNNs advanced forms such as graph autoencoders, spatial-temporal graph neural networks, and Graph Attention Networks have emerged. Graph autoencoders learn efficient representations of graph data [43], while spatial-temporal GNNs handle data that changes over time and space, making them useful for tasks like traffic prediction [44]. Graph Attention Networks introduce an attention mechanism, allowing the network to weigh the importance of different nodes and edges, improving performance in various tasks. [45] These advancements demonstrate the versatility and potential of GNNs in handling diverse and complex graph-based data.

2.2.1 Spatial based Graph Convolutional Neural Networks

Convolution is a specialized kind of linear operation which is used instead of matrix multiplication in at least one layer when considering standard CNNs. In a convolution operation, a small matrix called a filter or kernel slides over the input data (such as an image) to produce feature maps. This operation allows CNNs to detect patterns such as edges, textures, and shapes [46].

Spatial-based methods define graph convolutions based on a node's spatial relations, meaning they update a central node's representation by combining it with the representations of its neighboring nodes. The information from neighboring nodes is propagated along the edges of the graph, a concept shared with GRNNs. However, the key difference is that GRNNs use shared parameters across all layers to capture dynamic and sequential information in graphs, while spatial GCNNs utilize independent parameters at each layer to perform localized aggregation of node features based on the graph structure. The neighborhood of a node can be extended through incremental construction of the architecture [37]. A. Micheli proposed Neural Networks for Graphs (NN4G), the first work on spatial GCNNs [47]. NN4G performs graph convolutions by directly summing up a node's neighborhood information and also applies residual and skip connections to retain information across layers. Consequently, NN4G derives its next layer node states by Equation 2.1.

$$\mathbf{h}_v^{(k)} = f\left(\mathbf{W}^{(k)^T}\mathbf{x}_v + \sum_{i=1}^{k-1}\sum_{u\in N(v)} \Theta^{(k)^T}\mathbf{h}_u^{(k-1)}\right) \tag{2.1}$$

In this equation, k represents the layer index and $f(\cdot)$ denotes an activation function. The matrices $\mathbf{W}$ and Θ are learnable model parameters. The term $\mathbf{x}_v$ refers to the feature vector of node v, while $\mathbf{h}_u^{(k-1)}$ is the hidden feature vector of node u from the previous layer. Note that $\mathbf{h}_v^0 = 0$.

To further expand on these concept, In 2017, Gilmer et al. introduced the concept of Message Passing Neural Networks (MPNNs) [48], which serve as a general framework for spatial-based GCNNs. This approach treats graph convolutions as a message passing process, where information is transferred directly from one node to another along edges. MPNNs perform K-step message passing iterations to allow information to propagate further across the graph. The message passing function in an MPNN is defined as:

$$\mathbf{h}_v^{(k)} = U_k\left(\mathbf{h}_v^{(k-1)}, \sum_{u\in N(v)} M_k\left(\mathbf{h}_v^{(k-1)}, \mathbf{h}_u^{(k-1)}, \mathbf{x}_{vu}^e\right)\right) \tag{2.2}$$

In this equation, $U_k(\cdot)$ and $M_k(\cdot)$ are functions with learnable parameters. The variable $\mathbf{h}_v^{(k)}$ represents the hidden state of a node v in the k-th layer of the neural network, with the initial hidden state $\mathbf{h}_v^{(0)}$ equal to the feature vector of node v, denoted as x_v. The term $\mathbf{x}_{vu}^e$ refers to the edge feature vector of the edge (v, u). Figure 2.4 provides a graphical representation of how a single node aggregates messages from its neighboring nodes in a MPNN.

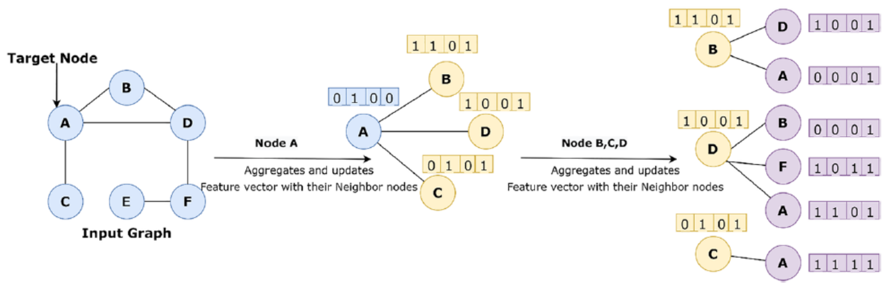


Figure 2.4: Concept of message aggregation for a single node from its adjacent neighbor nodes [2]

After obtaining the hidden representations of each node, $h_v^{(K)}$ can be used in two ways: it can be sent to an output layer for node-level prediction tasks, or it can be passed to a readout function for graph-level prediction tasks. The readout function creates a representation of the entire graph using the hidden representations of the nodes and is defined as

$$h_G = R(h_v^{(K)} | v \in G), \tag{2.3}$$

where $R(\cdot)$ represents the readout function with learnable parameters [48]. MPNNs have been progressively extended to manage edge attributes [48] and graph-level attributes [49]. These advancements have led to their widespread adoption across various domains of graph data, including bioinformatics [50, 51], combinatorial optimization [52, 53, 54], and recommender systems [36].

2.2.2 Relational Graph Neural Networks

A Relational Graph Neural Network (RGNN) is a type of GNN specifically designed for modeling relational data. It can be used for tasks such as entity classification (both graph and node classification) and link prediction. RGNNs are intended for use on directed and labeled multi-graphs [3]. They are based on the concept of message passing and can be seen as a specific example of the standard message passing framework [48] introduced by Gilmer et al. and already described in more detail in subsection 2.2.1. In an RGNN model, the forward update calculation for an entity is represented by:

$$\mathbf{h}_v^{(k+1)} = f\left(\sum_{r\in\mathcal{R}} \sum_{u\in\mathcal{N}_v^r} \frac{1}{c_{v,r}} \mathbf{W}_r^{(k)} \mathbf{h}_u^{(k)} + \mathbf{W}_0^{(k)} \mathbf{h}_v^{(k)}\right) \tag{2.4}$$

Where $f(\cdot)$ represents the activation function, which introduces non-linearity into the model. The term $\mathbf{W}$ denotes a matrix of learnable parameters, which are optimized during the training process. The variable $\mathbf{h}_v^{(k)}$ refers to the hidden state of node v in the k-th layer of the neural network, with $d^{(k)}$ being the dimensionality of this layer. The constant $c_{v,r}$ is a problem-specific normalization factor that can either be learned during training or chosen in advance. Finally, $\mathcal{N}_v^r$ indicates the set of neighbor indices of node v under relation $r \in \mathcal{R}$, which defines the relational structure of the graph [3]. The main difference of Equation 2.4 from the standard message passing equation is the inclusion of relation-specific transformations that depend on the type and direction of an edge. This means that for each edge type and direction, the model learns different weights in each layer. To ensure that the representation of a node at layer $k+1$ also relies on the hidden state of the node from the previous layer k, and not just on the states of neighboring nodes, a single self-connection is added to each node in the data. A neural network layer update involves evaluating Equation 2.4 in parallel for every node in the graph. Multiple layers can be stacked to capture dependencies across several relational steps. Figure 2.5 shows the computation of a single node update in the RGNN model.

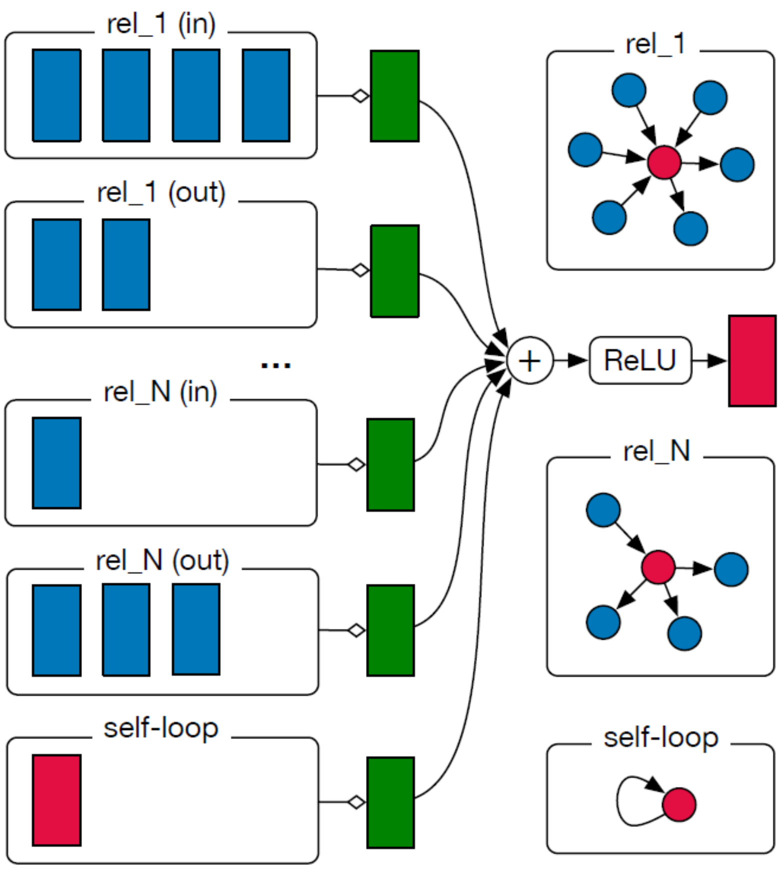


Figure 2.5: Diagram for computing the update of a single graph entity [3]

Since their publication, RGNN models have been applied in various domains. For instance, they have been used to predict temporal relationships in clinical data [55]. Additionally, RGNN models serve as encoders for topic modeling tasks [56], and are employed for event causality identification [9]. These applications demonstrate the versatility and effectiveness of RGNN models in handling complex relational data across different fields.

2.2.3 Graph Attention Networks

In recent years attention mechanisms have shown to be very effective in natural language processing and computer vision. Therefore the attention mechanism has been adopted for GNNs to selectively focus on key features and filter out irrelevant information. Traditional GNNs tend to treat all neighboring nodes equally when aggregating and disseminating information, which can be problematic since real-world graphs often contain noisy connections between unrelated nodes. This noise can result in less effective node representations. To counter this, the Graph Attention Network (GAT)[45] introduces an attention mechanism that helps the network learn which neighbors are more important.

The attention mechanism is inspired by the human visual system's process of selectively focusing on important details. It mimics human cognitive processes to highlight crucial parts of the data [57]. The general formulation of attention mechanisms [58] can be expressed as:

$$\text{Attention} = f(g(\mathbf{X}), \mathbf{X}) \tag{2.5}$$

In this equation, $g(\cdot)$ represents an attention function that identifies significant features, while $f(\cdot)$ processes the input data X to extract important information using $g(\cdot)$. The attention function $g(\cdot)$ consists of two main parts: an alignment function and a distribution function, with $f(\cdot)$ acting as a weighted sum to produce the final attention value. Essentially, the attention mechanism maps a sequence of keys K to an attention distribution α based on queries Q, with each key having a corresponding value V [59]. The alignment function, which is crucial to the attention mechanism, computes the attention alignment score and is defined as:

$$\text{scores} = \text{Sim}(\mathbf{Q}, \mathbf{K}) \tag{2.6}$$

Popular alignment functions include Cosine Similarity, Dot Product, Scaled Dot Product, Additive, and Concat. The distribution function then transforms these attention scores into attention coefficients α. This process can be described by:

$$\alpha = \text{Norm}(\text{scores}) \tag{2.7}$$

Here, $\text{Norm}(\cdot)$ refers to a distribution function, with the softmax function being the most commonly used method [58].

GATs use local attention to assign different weights to each neighboring node's representation [45]. By using normalized attention coefficients as relative weights, attention-based GNNs can aggregate and update node representations through a weighted sum function, which is then passed to subsequent layers [60]. In terms of node representation, let h_v^k and h_u^k denote the representations of nodes v and u in the k^{th} layer, with $h_v^0 = x_v^0$ representing the input features of node v. Initially, a shared linear transformation using a weight matrix Θ^k is applied to each node in the graph. This transformation is followed by an alignment function that computes the attention alignment score, indicating the significance of neighboring nodes. The attention scores are then converted into attention coefficients through a distribution function, allowing for comparison across different nodes. The node representations in the graph are updated and aggregated using a weighted sum function. The local attention layer can be mathematically expressed as follows:

$$h_v^k = \Theta^k x_v^k, \quad h_u^k = \Theta^k x_u^k \tag{2.8}$$

$$\text{scores}_{vu}^{k} = \text{Sim}(h_v^k, h_u^k) \tag{2.9}$$

$$\alpha_{vu}^{k} = \text{Norm}(\text{scores}^k) \tag{2.10}$$

$$h_v^{k+1} = \sigma\left(\sum_{u \in_v} \alpha_{vu}^k h_u^k\right) \tag{2.11}$$

In these equations, σ is the non-linear activation function, and Γ_v refers to the local neighborhood of node v, which corresponds to the first-order neighbors in GAT. $\text{Sim}(\cdot)$ denotes the alignment function, and $\text{Norm}(\cdot)$ represents the distribution functions.

2.2.4 Relational Graph Attention Networks

Relational Graph Attention Networks (RGAT) [21] are an extension of traditional graph attention models that include relational data. These models operate on graph structures using a self-attention mechanism that takes into account both the local relational structure and the node features. This approach allows for dynamically assigning importance to nodes and their properties under specific relations for different nodes in the graph, making RGATs applicable to a broader range of problems. Their spectral counterpart is the RGNN described in subsection 2.2.2. RGATs are build on the design of the GAT layer [45] and extend it to handle relational settings using concepts from RGNNs [3].

In an RGNN each relation provides distinct information. The update rule of the RGNN (Equation 2.4) does this by giving each node a unique intermediate representation under relation r. RGATs assume that the attention coefficient between two nodes depends only on the features of those nodes, up to a neighborhood-level normalization. Given linear transformations $\mathbf{W}^{(r)}$, the logits $E_{v,u}^{(r)}$ for each relation r are independent and defined as [21]:

$$E_{v,u}^{(r)} = a\left(g_v^{(r)}, g_u^{(r)}\right) \tag{2.12}$$

Logits are the raw, unnormalized scores that the model produces before applying any activation function. In Equation 2.12 $g_v^{(r)}$ and $g_u^{(r)}$ are the distinct intermediate representations of the nodes v and u under the relation r and a is the attention mechanism. In RGAT different forms of attention mechanisms can be used. The additive attention logits [45] are computed by summing the query and key features, followed by a LeakyReLU activation. This method enhances the interaction between the node features in an additive manner. On the other hand, the multiplicative attention logits [61] use a simple multiplication of the query and key scalar values to determine the interaction strength between nodes.

Busbridge et al. introduced two types of RGATs: Within-Relational Graph Attention (WIRGAT) and Across-Relational Graph Attention (ARGAT) [21]. WIRGAT operates on the assumption that the significance of relations is a global feature of the graph. It achieves this by assigning an independent probability distribution for each relation r over the nodes neighboring node v. Conversely, ARGAT operates under the assumption that the significance of relations is a local feature of the graph. It does so by using a single probability distribution across the different representations $g_u^{(r)}$ for nodes u neighboring node v. In Equation 2.13, the attention coefficients $\alpha_{v,u}^{(r)}$ are defined for the WIRGAT model under the relation r. Similarly, Equation 2.14 defines these coefficients for the ARGAT model.

$$\alpha_{v,u}^{(r)} = \text{softmax}_u \left(E_{v,u}^{(r)}\right) = \frac{\exp\left(E_{v,u}^{(r)}\right)}{\sum_{k \in \mathcal{N}_v^{(r)}} \exp\left(E_{v,k}^{(r)}\right)}, \quad \forall i, r : \sum_{u \in \mathcal{N}_v^{(r)}} \alpha_{v,u}^{(r)} = 1. \tag{2.13}$$

$$\alpha_{v,u}^{(r)} = \text{softmax}_{u,r} \left(E_{v,u}^{(r)}\right) = \frac{\exp\left(E_{v,j}^{(r)}\right)}{\sum_{r' \in \mathcal{R}} \sum_{k \in \mathcal{N}_v^{(r')}} \exp\left(E_{v,k}^{(r')}\right)}, \quad \forall v : \sum_{r \in \mathcal{R}} \sum_{u \in \mathcal{N}_v^{(r)}} \alpha_{v,u}^{(r)} = 1. \tag{2.14}$$

Where $E_{v,u}^{(r)}$ is the logit described in Equation 2.12 and $\mathcal{N}_v^{(r)}$ is the set of neighboring indices of node v under relation r. By combining these attention mechanisms from either the ARGAT or the WIRGAT with the neighborhood aggregation step of the RGNN the following equation for the update of the hidden state h of the node v is obtained [21]:

$$\mathbf{h}'_v = \sigma\left(\sum_{r \in \mathcal{R}} \sum_{u \in \mathcal{N}_v^{(r)}} \alpha_{v,u}^{(r)} \mathbf{g}_u^{(r)}\right) \in \mathbb{R}^{N \times F'}, \tag{2.15}$$

where σ represents an optional non-linear activation function, $\alpha_{v,u}^{(r)}$ are the attention coefficients from equation Equation 2.13 or Equation 2.14, and $\mathbf{g}_u^{(r)}$ are the intermediate representations of nodes u under relation r.

Busbridge et al. evaluated the WIRGAT and ARGAT models using both additive and multiplicative attention for transductive and inductive tasks. For the transductive tasks, they used the AIFB and MUTAG datasets [3] from the Resource Description Framework. For the inductive task, they used the molecular dataset Tox21 [62]. The performance of RGATs on the tested datasets varied, depending largely on the type of task. For relational inductive tasks, such as graph classification, the multiplicative ARGAT model was more effective. In contrast, for transductive relational tasks like knowledge base completion, especially when node features were absent, spectral methods like RGNNs or the Weisfeiler-Lehman graph kernels performed better then the RGAT models. The study concluded that WIRGAT paired with an additive logit mechanism was slightly superior to ARGAT in transductive tasks. Meanwhile, ARGAT combined with a multiplicative logit mechanism performed marginally better on inductive tasks. Importantly, the study did not find any scenario where any version of RGAT consistently outperformed RGNN. Therefore, it is advised to initially test the training set performance using RGNN before considering the use of RGAT [21].

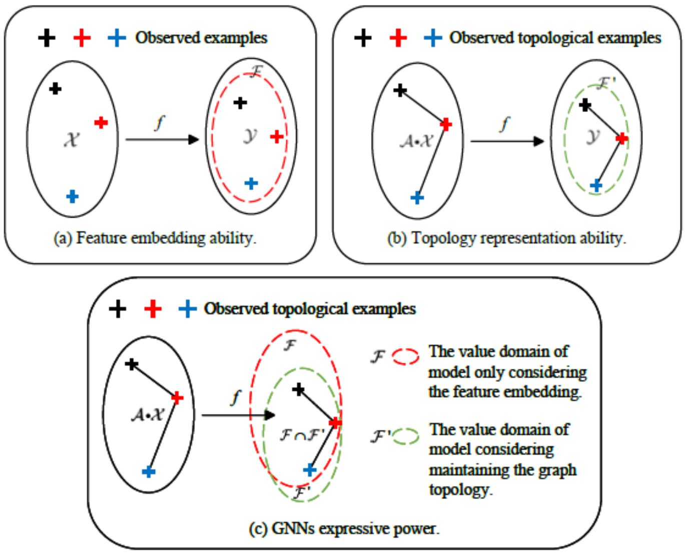
2.2.5 Expressiveness

Machine learning problems can be viewed as learning a mapping f^* from the feature space $\mathcal{X}$ to the target space $\mathcal{Y}$. Typically, f^* is approximated using a model f_θ by optimizing parameters θ. Since f^* is usually unknown beforehand, the aim is for f_θ to cover a wide range of possible functions f^*. This range is known as the expressive power of the model and serves as a key measure of its potential [63]. Neural networks demonstrate significant expressive power, as they can approximate all continuous functions [64].

GNNs extend this concept by adding the inductive bias of permutation invariance [49], which enables them to propagate and aggregate information through the graph's topology [65]. Unlike feed-forward neural networks, whose expressive power is limited by their width, GNNs' expressive power depends not only on their width but also on their ability to utilize graph topology for message propagation and node updates. As a result, the expressiveness of GNNs is determined by their feature embedding ability Figure 2.6 (a), which ensures nodes with different features have distinct embeddings, and their topology representation ability Figure 2.6 (b), which ensures nodes in different topological positions have distinct embeddings. Combining the two abilities, describes the overall expressive power of the GNN Figure 2.6 (c), with the topology representation ability beeing the primary factor in limiting the expressiveness of GNNs. [4].

Determining whether two graphs share the same structure involves solving the graph isomorphism problem, which examines if two graphs are topologically equivalent. This problem is notably challenging and no algorithms for solving it in polynomial time are known [66]. The Weisfeiler-Lehman (WL) test [67] [68] is a widely used method to address graph isomorphism. The WL test uses a graph coloring approach to assign labels to each node. Initially, each node receives a unique color label. In each iteration, the colors of a node and its neighbors are combined into a multiset, which is then hashed into a new unique color, updating the node's label. This iterative process continues until the node colors no longer change, producing a color distribution histogram for the graph. By comparing these histograms, the WL test can identify if two graphs are isomorphic. While the WL test is efficient for many graphs [69], it's important to note that the same color distributions can only suggest but not definitively confirm graph isomorphism [4].

MPNNs work in a way that is comparable to the WL algorithm. Instead of using natural numbers for node colors, MPNNs use real vectors known as node embeddings. These embeddings are generated through parametric aggregation and combination functions, making them learnable. As a result, GNNs have an expressive power on par with the WL test [70] [71]. This means that if the WL test cannot differentiate between two graphs, a GNN will also be unable to distinguish between them.



Observed examples
f
$\mathcal{X}$
$\mathcal{Y}$
$\mathcal{F}$
(a) Feature embedding ability.
Observed topological examples
$A \cdot \mathcal{X}$
$\mathcal{F}'$
(b) Topology representation ability.
Observed topological examples
$\mathcal{F} \cap \mathcal{F}'$
The value domain of model only considering the feature embedding.
The value domain of model considering maintaining the graph topology.
(c) GNNs expressive power.

Figure 2.6: Demonstrating the expressive capabilities of GNNs. a) GNNs share the same feature embedding ability as NNs, mapping examples from the feature space $\mathcal{X}$ to the target space $\mathcal{Y}$ via f b) GNNs' capability to represent topology involves mapping examples from the feature space to the target space using $\hat{f}$, while maintaining the original topology between examples. c) The expressive power of GNNs is a combination of their feature embedding and topology representation abilities, measured by the size of the intersection of $\mathcal{F}$ and $\mathcal{F}'$ when $\mathcal{X}$ is random.[4]

To apply the concept of expressiveness in the context of MPNNs in automated planning, we can leverage the relationship between the WL algorithm and description logic. In planning, general policies and heuristic functions for many classical benchmark domains can be represented using features derived from domain predicates with a description logic grammar [72] [73]. The C_2 fragment of first-order logic allows formulas with only two variables but includes counting quantifiers. This fragment is connected to the WL algorithm because two nodes in a graph are classified the same by the WL test if they satisfy the same unary C_2 formulas [74] [75]. This means that general policies and heuristic functions can be learned from planning domain predicates using MPNNs if these predicates can be expressed in terms of C_2 features.For a better understanding of planning domains expressed as C2 features let us consider the following two examples [76] for a blocksworld and rovers domain:

In Blocksworld problems, the objective is to stack a set of blocks in a specific arrangement. These problems are characterized by goals of the form $ON(x, y)$, where x and y represent blocks. We can define these problems using the following features:

$$\begin{aligned}
\alpha &= \exists xy(ON_G(x,y) \land \neg ON(x,y)),\\
L &= \exists xy(ON_G(x,y) \land (\neg CLEAR(y) \lor \neg HOLDING(x))),\\
X_k &= \exists xy(ON_G(x,y) \land \eta_k(x) \land \neg CONN_N^2[ON_G^2, ON](x)),\\
Y_k &= \exists xy(ON_G(x,y) \land \eta_k(y) \land \neg CONN_N^1[ON_G^1, ON](y))
\end{aligned}$$

Here, L indicates whether x is not being held or cannot be placed on y. The feature X_k (and similarly Y_k) signifies that there are k blocks above x (or y) and that x (or y) is not above y (or x). These features only involve up to two variables (x and y) and the counting quantifier k, therefore fitting into the C_2 category. Consequently, a MPNN can be used to learn a policy for this problem domain.

For Rovers problems, multiple rovers with different capabilities (such as soil analysis) need to carry out experiments and report back to the lander. A simpler scenario involves sampling soil at a specific location. These problems can be characterized by the following features:

$$\begin{aligned}
P_0(r,x) &= \text{AT-SOIL-SAMPLE}(x),\\
P_k(r,x) &= \exists y(\text{CAN-TRAVERSE}(r,x,y) \land P_{k-1}(r,y)),\\
SP_k(r,x) &= \text{AT}(r,x) \land P_k(r,x) \land \neg P_{k-1}(r,x).
\end{aligned}$$

To find the nearest capable rover for soil sampling R_k, we need features to determine if the rover is full F_k, if the soil is not yet sampled S, and if the goal is unmet α. Boolean features L_k are also needed to indicate the distance from the soil or rover with the sample to a location where data can be sent to the lander. The formulas $SP_k(r,x)$ define the features R_k. Since these formulas use three variables, the features used to decompose V^* do not fit into C_2. Hence, it is challenging to use a MPNN to learn a policy in the Rovers domain.

There exist some approaches to overcome the limitations of C_2. One method is to use k-GNNs, specifically with $k = 3$, where groups of three objects are embedded instead of individual ones [70]. 3-GNNs, or 3rd-order Graph Neural Networks, extend the typical GNN architecture by considering triplets of nodes rather than pairs or individual nodes. This means they can capture more complex relationships in the data. It has been shown that 3-GNNs have the expressive power of C_3 logic, unlike the C_2 power of 1- and 2-GNNs [77]. However, 3-GNNs do not scale well because they need a cubic number of embeddings and quartic time for message passing. To adress this issue Ståhlberg et al. proposed an alternative, parameterized version of Relational GNNs R-GNNsthat operates in quadratic time for message passing while still capturing C_3 features [78].

2.2.6 Masked Attention for Graphs

As presented in the previous sections GNNs making use of different variants of the message passing algorithms are predominantly used for graph learning due to their adaptability, speed, and effective performance. However, creating powerful and versatile GNNs requires substantial research and often involves the development of complex and carefully-selected message passing operators. Despite their strengths, as discussed in section subsection 2.2.5, message passing GNNs have limitations in terms of expressiveness. Thus, this section introduces Masked Attention for Graphs (MAG) as an alternative approach to graph learning, which does not rely on message passing layers but rather uses a classical attention mechanism with masking to create customized attention patterns.

MAG [5] treats graph learning as a set-based learning task, where the graph structure (i.e., adjacency matrix) is maintained by masking the pairwise attention weight matrix, focusing only on node or edge features. As MAG is based on the idea of learning on sets, it is inspired by the Set Transformer. Set Transformers [79] are attention-based architectures with encoder-decoder structures, designed for learning on sets. They utilize scaled dot-product and multihead attention mechanisms [61] to form Multihead Attention Blocks (MABs), Self Attention Blocks (SABs), and Pooling by Multihead Attention (PMA) blocks. The original Set Transformer architecture [79] is described as $\mathrm{ST}(X) = \mathrm{Decoder}(\mathrm{Encoder}(X))$ with:

$$\begin{aligned} \mathrm{MAB}(X, Y) &= H + \mathrm{Linear}_{\varphi}(H) \\ H &= X + \mathrm{Multihead}(X, Y, Y) \\ \mathrm{SAB}(X) &= \mathrm{MAB}(X, X) \\ \mathrm{Encoder}(X) &= \mathrm{SAB}^{n}(X) \\ \mathrm{PMA}_{k}(Z) &= \mathrm{MAB}(S_k, \mathrm{Linear}_{\varphi}(Z)) \\ \mathrm{Decoder}(Z) &= \mathrm{Linear}_{\varphi}(\mathrm{SAB}^{n}(\mathrm{PMA}_{k}(Z))) \end{aligned} \tag{2.16}$$

assuming that a function *Multihead(Q, K, V)* [61] for the query, key, and value matrices, is available. The term $\mathrm{Linear}_{\varphi}$ refers to a linear layer followed by an activation function φ. The notation $\mathrm{SAB}^{n}(\cdot)$ signifies n successive applications of a SAB, while S_k denotes a tensor containing k learnable seed vectors, which are initialized randomly (PMA_k produces k vectors).

MAG extends the set transformer by applying a mask to the pairwise attention weight matrix according to node or edge adjacency information. When the node adjacency matrix is utilized, the information propagation relies on the node feature matrix X, and this approach is called MAGN. On the other hand, when the edge adjacency matrix and the edge feature matrix E are used together, the approach is known as MAGE. In both approaches, the MAB and SAB blocks of the set transformer are substituted with their masked counterparts, known as MSAB. This involves providing the new blocks with a mask tensor of shape $B \times N_d \times N_d$ for MAGN and $B \times N_e \times N_e$ for MAGE, where B represents the batch size. For a node $u \in V$, the node-to-node propagation mechanism in MSAB is defined as follows [5]:

$$\mathrm{MSAB}_u(\mathbf{X}, \mathbf{A}) = \sum_{j=1}^{h} \mathbf{W}_O^j \left(\mathbf{W}_V^j \mathbf{X}_{N_u} \right) \sigma \left(\left(\mathbf{W}_K^j \mathbf{X}_{N_u} \right)^\top \left(\mathbf{W}_Q^j \mathbf{x}_u \right) \right) \tag{2.17}$$

where the parameter h describes the number of attention heads. $\mathbf{X}$ denotes the input feature matrix, and $\mathbf{x}_u$ is the row associated with node u. $\mathbf{X}_{N_u}$ refers to the neighborhood of node $u \in \mathcal{V}$ as specified by the adjacency matrix, thereby focusing attention on a subset of $\mathbf{X}$. The matrices $\mathbf{W}_O^j$, $\mathbf{W}_V^j$, $\mathbf{W}_K^j$, and $\mathbf{W}_Q^j$ are the weight matrices for outputs, values, keys, and queries, respectively. Lastly $\mathbf{A}$ is the adjacency matrix and σ is the softmax function. In a basic implementation, masking involves replacing the values of the scaled dot product inside the softmax function with negative infinity (or a very large negative number for stability) before applying the softmax function. The correct mask for each batch is unique and must be computed dynamically. For MAGN, the mask represents connections of adjacent nodes in is therefore equal to the node adjacency matrix. For MAGE, the mask is applied to the set of edges, permitting only edges that share a common node. Although this computation is more complex than for the MAGN mask, both types of masks can be efficiently computed using tensor operations. A complete MAG model is structured with an encoder where MSAB and SAB blocks are alternated as needed, along with a PMA-module decoder. The decoder is only necessary for graph-level tasks, as it functions similarly to the readout fucntion in GNNs. For node-level tasks, no decoder is required. Figure 2.7 displays a possible MAG architecture for a graph level task using either the MAGE or MAGN approach. MAG can utilize either layer or batch normalization and can optionally include MLPs after each multihead attention block [5].

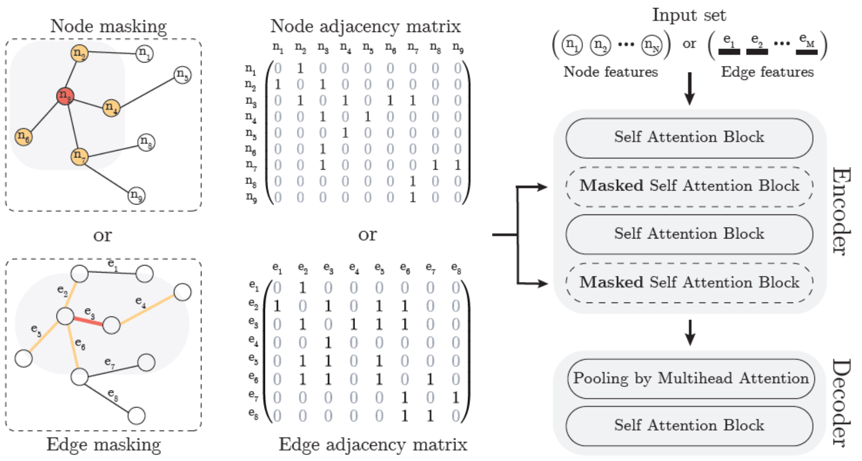


Figure 2.7: The Masked Attention for Graphs (MAG) architecture. An essential component is deciding whether to process the node features (MAGN) or the edge features (MAGE) and applying the correct masking strategy. Standard self-attention blocks (S) can be interchanged with masked blocks (M) as required, here a SMSM configuration is shown. The decoder is required only for tasks at the graph level. [5]

MAG, despite its straightforward design, consistently outperforms traditional message-passing baselines and more intricate Transformer-based methods in various graph benchmarks. In the study by Buterez et al. [5] out of the MAG approaches, MAGN delivered the best performance on node-level tasks, while MAGE excelled in graph-level and long-range tasks. Nevertheless, MAG has certain limitations. The implementation relies on either PyTorch's [80] or xformers' [81] native flash attention, which users can choose between. Although both flash attention methods are memory-efficient, they do not support custom attention masks. As a result, masks need to be generated during training, which can be done efficiently but involves squaring the number of nodes or edges as the mask are of dimensions (N_d, N_d) for nodes and (N_e, N_e) for edges. Additionally, regardless of whether PyTorch or xformers is used, the mask tensor must be replicated for each attention head, which can create a bottleneck when replication occurs in the batch dimension [5].

Overall, MAG represents an innovative and promising approach to graph learning that overcomes the expressiveness limitations of message-passing GNNs. However, its implementation is not yet fully optimized. Therefore, when comparing MAG to message-passing GNNs, these considerations should still be taken into account.

2.2.7 Domain Adaptation

This section delves into the challenge of domain adaptation in GNNs. Despite the strong representation capabilities of GNNs, like other machine learning models, they often struggle with effectiveness and tend to be overly confident when the test data distribution significantly differs from the training data distribution [38].

Domain adaptation addresses the out-of-distribution (OOD) problem between a source and a target domain. In graph learning, OOD scenarios occur when the model encounters graph instances that are substantially different from those seen during training [82]. OOD generalization [83] aims to improve the model's accuracy on instances that deviate from the training data, enabling reliable predictions even on novel, unseen graph instances. Distribution shifts generally include attributive shifts and structural shifts. Attributive shifts pertain to changes in the distribution of node attributes, which can arise from different environments or backgrounds. Structural shifts refer to variations in adjacency matrices due to changes in connectivity or graph size [38].

One straightforward method to address the OOD issue and achieve domain adaptation is parameter adaptation. Parameter adaptation, particularly through simple retraining or fine-tuning, involves leveraging a GNN pre-trained on source domain data and further training it on target domain data. This method allows the model to retain useful knowledge from the source domain while adjusting its parameters to capture the specific characteristics of the target domain. Fine-tuning is especially effective when there is a reasonable amount of labeled data in the target domain, as it enables the GNN to refine its representations and improve performance on target-specific tasks [84].

Another approach is instance re-weighting, which addresses domain adaptation by adjusting the influence of source domain instances based on their relevance to the target domain [85]. This involves assigning weights to nodes or edges in the source domain to align their distribution more closely with that of the target domain. Techniques such as Importance Weighting [86] and Kernel Mean Matching [87] ensure that the training process emphasizes source instances that are more representative of the target domain. By doing so, instance re-weighting enhances the GNN's ability to generalize to the target domain, thus improving its robustness and adaptability in diverse environments.

Adversarial learning is another strategy for handling OOD scenarios. By training the model with adversarial examples—inputs intentionally crafted to challenge its predictions—the model learns to recognize and manage unexpected or unfamiliar data distributions. Adversarial learning has been widely adopted for OOD generalization [88] to reduce domain discrepancy, and it naturally extends to graph data. These approaches use adversarial learning to generate effective perturbations that enhance generalization. Some works also introduce a domain classifier trained adversarially to improve the invariance of graph representations across different domains. For instance, DEAL [88] applies adversarial perturbations to both node attributes and features to adapt source graphs to the target domain.

Subgraph-based methods [89] assume that each graph is composed of a crucial part and a non-crucial part, derived from semantic and environmental information, respectively. To identify subgraphs with essential knowledge, these approaches typically employ causal inference and invariant theory for effective graph representation learning. Causal inference in subgraph-based methods identifies robust, causally relevant subgraph patterns by distinguishing cause-and-effect relationships within the graph. This enables the model to focus on critical, invariant features that remain relevant even when the data distribution changes. Examples of GNNs using causal inference to manage distribution shifts include CAL [83], which integrates graph representations into the structural causal graph and uses an attention mechanism and representation disentanglement to select causal patterns, and StableGNN [90], which uses a differentiable graph pooling operator for subgraph extraction, optimized with a distinguishing regularizer to reduce spurious correlations. Invariant theory aims to extend empirical risk minimization to invariant risk minimization to increase robustness to distribution shifts. For example, SizeShiftReg [91] simulates size shifts using graph coarsening and proposes a simple regularization loss for consistency learning after coarsening, while GIL [92] learns a mask matrix for subgraph generation and then enforces model invariance to environment inference using an invariance regularizer.

In conclusion, addressing domain adaptation in graph neural networks is crucial due to the challenges posed by out-of-distribution data. Several approaches exist beyond those discussed in this section. The approaches presented here—parameter adaptation, instance re-weighting, adversarial learning, and subgraph-based methods—offer promising solutions. Each focuses on enhancing the model's ability to generalize to new, unseen data by employing various techniques to manage distribution shifts. By improving the robustness and adaptability of GNNs, these methods contribute to more reliable and accurate predictions in diverse and dynamic environments.

2.3 Graphs Optimised for Search Evaluation

Goose (Graphs Optimized for Search Evaluation) is a learning-based framework for classical planning. It was first introduced by D. Z. Chen, S. Thiébaux, and F. Trevizan in [6] and further extended by them in [18]. Goose effectively represents planning tasks using various graph structures. By leveraging these graph representations, a GNN is employed to generate heuristic functions that facilitate the resolution of planning problems. GNNs are used because they exhibit great generalization potential; once trained, they offer outputs for any graph regardless of size or structure. Goose is specifically designed for integration as a custom heuristic within the Fast Downward planning system. Figure 2.8 illustrates the interaction between Goose and the Fast Downward planner for solving a planning problem.

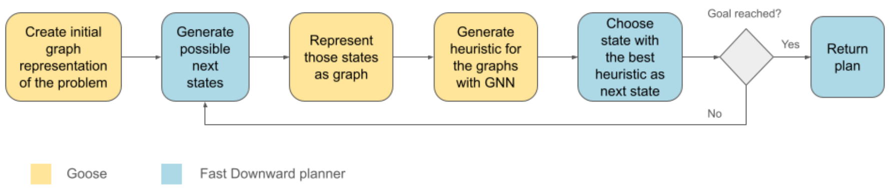

Create initial graph representation of the problem
Generate possible next states
Represent those states as graph
Generate heuristic for the graphs with GNN
Choose state with the best heuristic as next state
Goal reached?
Yes
No
Return plan
Goose
Fast Downward planner

Figure 2.8: Solving a planning problem with Goose and the Fast Downward planning system

Goose can learn both domain-dependent and domain-independent heuristics. Domain-dependent heuristics are tailored specifically to a particular problem domain, meaning that the GNN was trained on problems from the same domain it is applied to. Domain-independent heuristics are general-purpose strategies that can be applied across various problem domains without any customization, meaning that the GNN is trained on problems from various domains and not only on those it is applied to. Goose defines novel grounded and lifted graph representations of planning tasks and presents the first method for learning domain-independent heuristics with only the lifted representation of a planning task, while also been optimized for runtime with the use of GPU batch evaluation [6]. In addition to heuristics based on GNN, the 2024 extension of Goose called WL-Goose [18] offers an approach to build heuristic functions based on graph kernels. WL-Goose introduces another lifted graph representation. Based on this representation and using the Weisfeiler Leman (WL) algorithm, features are generated that are used with classical machine learning methods (SVMs and Gaussian Processes) to generate heuristics. WL-Goose focuses on learning domain-specific heuristics and demonstrates that the learned heuristics outperform current state-of-the-art heuristics while being faster and using fewer parameters.

2.3.1 Graphs of GOOSE

Goose offers 4 different graph representations. Two grounded graph representations and two lifted graph representations. Grounded graph representations, also known as propositional or instantiated representations, explicitly specify all variables, actions, and states in the planning problem. This detailed representation enumerates every possible state and action, leading to large graph sizes, especially for problems with many variables and objects. While grounded representations provide a thorough description, they can be computationally expensive due to their size, causing performance issues in planning algorithms [93]. For example, in a logistics planning problem, a grounded representation would list all possible locations for each package and truck and every potential action.

The first grounded representation available in Goose is the STRIPS Learning Graph (SLG) [6]. It is based on a STRIPS problem $\langle P, A, s_0, G\rangle$, already described in more detail in subsubsection 2.1.1.1, and is defined as the graph $\langle V, E, \mathbf{X}\rangle$ with:

- $V = A \cup P$,
- $E = E_{\text{pre}} \cup E_{\text{add}} \cup E_{\text{del}}$ where for $\iota \in \{\text{pre}, \text{add}, \text{del}\}$,

  $E_\iota = \{(a,p)_\iota \mid p \in \iota(a), a \in A\}$,
- $\mathbf{X} : V \to \mathbb{R}^3$ defined by $u \mapsto [u \in P; u \in s_0; u \in G]$.

In this representation all possible action and object combinations and all possible propositions become a node. Edges with different labels define the preconditions, add effects and delete effects of the actions. In Figure 2.9 (a), this concept of edge labels is shown by the yellow (preconditions), green (add effects), and red (delete effects) colored edges. The propositions nodes are assigned different node features depending if they are part of the goal (green nodes) or not (blue nodes). By allowing for edge labels, only one node for each proposition is required to encode the semantics of action effects. Additionally, three-dimensional node features are sufficient for encoding whether a node corresponds to an action or proposition, and in the latter case, whether it is true in the initial state and present in the goal state [6].

The second grounded representation is the FDR Learning Graph (FLG) [6], which is based on an FDR problem $\langle \mathcal{V}, \mathcal{A}, s_0, s_\star\rangle$, already described in more detail in subsubsection 2.1.1.2. There already exists a graph representation called the FDR Problem Description Graph (PDG) [94] that can identify symmetrical states during the search for FDR problems, but as this graph representation was not designed for learning with GNNs, the FLG extends the PDG representation by adding node features and edge labels. An FLG is defined as the graph $\langle V, E, \mathbf{X}\rangle$ with:

- $V = \mathcal{V} \cup \bigcup_{v \in \mathcal{V}} D_v \cup \mathcal{A}$
- $E = E_{\text{var:val}} \cup E_{\text{pre}} \cup E_{\text{eff}}$ with

$$E_{\text{var:val}} = \bigcup_{v\in\mathcal{V}} \{(v,d)_{\text{var:val}} \mid d \in D_v\}$$
$$E_{\text{pre}} = \bigcup_{a\in\mathcal{A}} \{(d,a)_{\text{pre}} \mid (v,d) \in \text{pre}(a)\},$$
$$E_{\text{eff}} = \bigcup_{a\in\mathcal{A}} \{(d,a)_{\text{eff}} \mid (v,d) \in \text{eff}(a)\},$$

- $\mathbf{X} : V \to \mathbb{R}^5$ defined by
  $u \mapsto [u \in \mathcal{V}; u \in A; \text{val}(u); \text{true}(u); \text{goal}(u)]$ *where*
  $\text{val}(u) = \exists v \in \mathcal{V}, u \in D_v, \text{true}(u) = \exists v \in \mathcal{V}, (v,u) \in s_0, \text{goal}(u) = \exists v \in \mathcal{V}, (v,u) \in s_\star$

An FLG graph introduces the concepts of variable and value nodes. In Figure 2.9 (b) variable nodes are represented by the grey nodes and facts are considered concrete values of these variables. The value nodes are again assigned different node features depending wether we deal with an non goal value, unachieved or achieved goal value, represented by the different node colors in Figure 2.9 (b). In the FLG representation, all actions and facts are depicted as nodes. Edges describe preconditions and positive effects of actions and link the variable and value nodes. In contrast to the SLG representation the delete effects of actions are not modeled in the FLG representation.

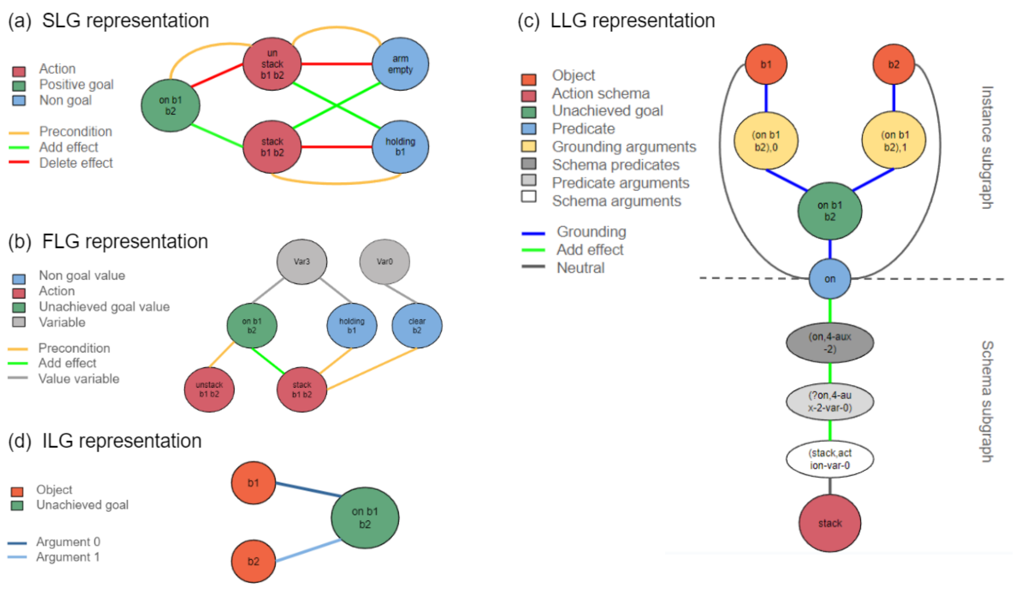


Figure 2.9: Representation of the fact (on b1 b2) as part of the goal in a blocksworld problem, shown as an SLG graph (a), FLG graph (b), LLG graph (c), and ILG graph (d). The depicted graphs are all subgraphs focusing solely on the fact (on b1 b2). Complete graphs for a planning problem are too large for a clear graphical representation.

The FLG representation is the only representation that uses the Fast Downward planner during the graph generation phase, as the intermediate planner output from the translation phase facilitates the representation of the planning problem as an FDR problem.

Lifted graph representations, also known as first-order or relational representations, describe a planning problem in a more abstract form using relations and quantifiers, without explicitly enumerating all possible states and actions. This compact representation avoids the combinatorial explosion of grounded graphs, making it more scalable and flexible for large and complex problems [27]. In the logistics planning example, a lifted representation would describe generic actions like "move any truck to any location" or "load any package onto any truck," with specific instances instantiated during the planning process. The first lifted graph representation offered by Goose is the lifted learning graph (LLG) [6]. It is based on a lifted problem $\Pi = \langle \mathcal{P}, \mathcal{O}, \mathcal{A}, s_0, G \rangle$ already described in more detail in subsubsection 2.1.1.3 and is defined as the graph $G = \langle V, E, \mathbf{X} \rangle$ with:

- $V = \mathcal{P} \cup \mathcal{O} \cup N(\mathcal{A}) \cup N(s_0 \cup G)$ with

  $$N(s_0 \cup G) = \bigcup_{p=P(o_1,\ldots,o_{np}) \in s_0 \cup G} \{p, p_1, \ldots, p_{np}\}$$

  $$N(\mathcal{A}) = \bigcup_{a \in \mathcal{A}} (\{a\} \cup \{a_\delta \mid \delta \in \Delta(a)\}) \cup \bigcup_{f \in \{\text{pre,add,del}\}} \bigcup_{p=P(\delta_1,\ldots,\delta_{n_p}) \in f(a)} \{p_{a,f}, p_{a,f,1}, \ldots, p_{a,f,np}\}$$

  The nodes in $N(s_0 \cup G)$ represent the state and goal, along with the ground arguments, shown as green and yellow nodes in Figure 2.9 (c). $N(A)$ includes all nodes associated with the action schema, schema argument, predicate argument, and schema predicate nodes, depicted as red and grey nodes in Figure 2.9 (c).

- $E = E_\nu \cup E_\gamma \cup \bigcup_{f \in \{\text{pre,add,del}\}} E_f$ where

  $$E_\nu = \{\langle o, P \rangle_\nu \mid o \in \mathcal{O}, \mathbf{P} \in \mathcal{P}\} \cup \{\langle a, a_\delta \rangle_\nu \mid \delta \in \Delta(a), a \in \mathcal{A}\}$$

  $$E_\gamma = \bigcup_{p=P(o_1,\ldots,o_{np}) \in s_0 \cup G} \left(\{\langle p, p_i \rangle_\gamma \mid i \in [np]\} \cup \{\langle p_i, o_i \rangle_\gamma \mid i \in [np]\} \cup \{\langle p, P \rangle_\gamma\}\right)$$

  $$E_f = \bigcup_{p=P() \in f(a)} (\{\langle P, p_{a,f} \rangle_f, \langle p_{a,f}, a \rangle_f\} \cup$$

  $$\bigcup_{p=P(\delta_1,\ldots,\delta_{np}) \in f(a), n_p \geq 1} \left(\{\langle P, p_{a,f} \rangle_f\} \cup \{\langle p_{a,f}, p_{a,f,i} \rangle_f, \langle p_{a,f,i}, a_{\delta_i} \rangle_f \mid i \in [np]\}\right) \text{for} f \in \{\text{pre, add, del}\}$$

  E_γ represents the grounding edges, illustrated as blue edges in Figure 2.9 (c). These edges connect nodes in $\mathcal{P}$, $\mathcal{O}$, and $N(s_0 \cup G)$ to depict propositions in the goal and those true in the state, where predicates are instantiated with objects in the correct arguments. $\bigcup_{f \in \{\text{pre,add,del}\}} E_f$ connects nodes in $\mathcal{P}$ and $N(\mathcal{A})$ to encode the semantics of action schemas in the graph, exemplified by the green E_{add} edges shown in Figure 2.9 (c). Finally, E_ν connects objects to predicates and schemas to their arguments, indicated by neutral gray edges in Figure 2.9 (c).

- $\mathbf{X} : V \to \mathbb{R}^{5+T}$ defined by
  $u \mapsto [u \in \mathcal{P}; u \in \mathcal{O}; u \in \mathcal{A}; u \in s_0; u \in G] \| \overline{IF}(u)$

  where $\| \cdot \|$ denotes vector concatenation, $\overline{IF}(u) = IF(i)$ for u of the form p_i or $p_{a,f,i}$ with $f \in \{\text{pre}, \text{add}, \text{del}\}$, and $\overline{IF}(u) = \overrightarrow{0}$ otherwise. The function $IF : \mathbb{N} \to \mathbb{R}^T$ is an index function defined by a fixed, randomly chosen injective map from $\mathbb{N}$ to the sphere $\{x \in \mathbb{R}^T \mid \|x\| = 1\}$.

The LLG graph can be devided into two subgraphs: the instance subgraph and the schema subgraph which are connected by the predicate nodes. The instance subgraph encodes the instances specific information which includes the goal state, the current state of a plan, the objects and predicates. With the objects and the predicates forming a fully connected graph between them. The schema subgraph, on the other hand, represents the domain actions along with their preconditions and effects [6].

The second lifted representation is the instance learning graph (ILG) [18]. It is also based on a lifted problem $\Pi = \langle \mathcal{P}, \mathcal{O}, \mathcal{A}, s_0, G \rangle$ already described in more detail in subsubsection 2.1.1.3 and is defined as the graph $G = \langle V, E, c, l \rangle$ with:

- $V = \mathcal{O} \cup s_0 \cup G$
- $E = \bigcup_{p=P(o_1,\ldots,o_{n_p}) \in s_0 \cup G} \{\langle p, o_1 \rangle, \ldots, \langle p, o_{n_p} \rangle\}$
- $c : V \to (\{ap, ug, ag\} \times \mathcal{P}) \cup \{ob\}$ defined by
  $$u \mapsto \begin{cases} ob, & \text{if } u \in \mathcal{O}; \\ (ag, P), & \text{if } u = P(o_1, \ldots, o_{n_p}) \in s_0 \cap G; \\ (ap, P), & \text{if } u = P(o_1, \ldots, o_{n_p}) \in s_0 \setminus G; \\ (ug, P), & \text{if } u = P(o_1, \ldots, o_{n_p}) \in G \setminus s_0; \end{cases}$$
- $l : E \to \mathbb{N}$ with $\langle p, o_i \rangle \mapsto i$.

An ILG features a node for each object and combines propositions that hold true in the initial state s_0 and the goal condition G. Propositions are connected to n object nodes that instantiate them, with edge labels indicating the position of each object in the predicate's argument list. Figure 2.9 (d) shows this instantiation using the example of the proposition (on b1 b2) from a blocksworld problem. Node colors indicate whether a node corresponds to an object (ob), a proposition true in s_0 (ap), a proposition in the goal G (ug), or both (ag), along with its corresponding predicate. Specifically, ug refers to an unachieved goal, ag to an achieved goal, and ap to a non-goal proposition that is achieved. Note that ILGs do not consider the transition system of the planning task, as they omit the concept of actions [18].

In the context of Figure 2.9, it is important not to misinterpret the size of the depicted subgraphs, particularly the LLG representation, which appears to have the most nodes and edges. This larger size is due to the need to include schema and predicate argument nodes for the lifted representation. When considering the complete graph representation of a planning problem, this more abstract representation ultimately results in a smaller graph compared to the two grounded representations. In the grounded SLG and FLG representations, all possible states and actions are instantiated from the start, leading to larger graphs. Conversely, in the lifted representations, nodes are added progressively during the search process, whereas in the grounded representations, only the types of the nodes change but their total number remains fixed because everything is instantiated from the beginning. The ILG representations are typically the most concise because they exclude the concept of actions entirely.

2.3.2 GNN of GOOSE

The GNN currently utilized in Goose for representing the heuristic functions, is a message-passing Relational Graph Neural Network (RGNN). The implementation of the message-passing algorithm follows the method from Schlichtkrull et al. detailed in [3] and is explained in more detail in subsection 2.2.2. The architecture of the RGNN used in Goose is depicted in Figure 2.10.

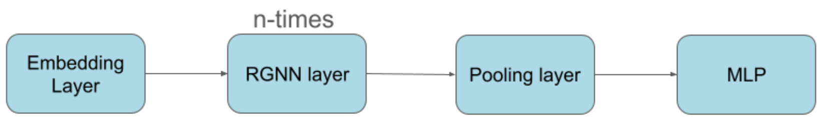


Figure 2.10: Architecture of the GNN used in Goose

The embedding layer initially performs a linear transformation of the node features to a higher dimension, differing from a "classical" embedding layer which typically generates vector representations of the nodes in the graph. The RGNN layer comprises one linear layer applied to the node features and one linear convolutional layer per edge type and direction. These linear convolutional layers perform the message passing. Subsequently, an aggregation function collects the passed messages and forwards them to a ReLU activation function. The possible types of aggregation (e.g., sum, mean, max) and the number of RGNN layers are hyperparameters of the GNN. Following the RGNN layers, a pooling layer reduces the graph to an overall structure and features, enabling graph-level predictions. The type of pooling is also a hyperparameter, with options including sum, mean, and max. The Multi-Layer Perceptron (MLP) layer consists of two linear layers with a ReLU activation function in between. The MLP ensures that the final prediction is in the correct format and scale required for the heuristic function.

Table 2.2 provides an overview of the hyperparameters in the Goose neural networks. In the experiments described in [6] a GNN with hyperparameter combination of 8 layers, 64 hidden units, sum pooling method, and mean aggregation function was used. In contrast in the experiments comparing Goose with the extension WL-Goose a GNN with hyperparameter combination of 4 layers, 64 hidden units, sum pooling method, and mean aggregation function was used [18].

Table 2.2: Overview of Hyperparameters in the Goose Neural Networks

Hyperparameter	Meaning	Possible Values
pool	Pooling function used by the pooling layer for readout	sum, mean, max
nhidden	Number of hidden units in the RGNN layers	Integer values
nlayers	Number of RGNN layers	Integer values
aggr	Aggregation method used in the message passing	sum, mean, max

The RGNNs in Goose are trained using the ADAM optimizer [95], a batch size of 16, and a Mean Squared Error (MSE) loss function. The learning rate is scheduled by extracting 25% of the training data, starting with an initial learning rate of 0.001, and is reduced by a factor of 10 if the loss on this data subset does not decrease over the last 10 epochs. Training is stopped when the learning rate drops below 10^{-5}.

The RGNNs in Goose aim to estimate the heuristic h^*, which describes the cost of the remaining plan and, if all actions have an equal cost of 1, also the length of the remaining plan. In terms of expressiveness, MPNNs have some limitations regarding the prediction of the heuristic h^*. MPNNs are unable to learn h^* with the Goose graph representations and cannot learn a relative or absolute approximation of h^* across all planning problems [6]. However, MPNNs can approximate h^* within certain subclasses of planning tasks. These subclasses include tasks that belong to the C_2 fragments of first-order logic because, as already discussed in subsection 2.2.5, MPNNs can learn C_2 features.

2.3.3 Results of GOOSE

2.3.3.1 Domain Dependent Results

Experiments were conducted using the dataset from the learning track of the 2023 International Planning Competition (IPC) [19]. All actions in the problem domains in the dataset have a unit cost. Each domain featured instances categorized into easy, medium, and hard difficulties based on the number of objects in each instance. In each domain, the training set contained up to 99 easy instances. The test set consisted of 30 instances from each difficulty level — easy, medium, and hard — that were not part of the training set. The models were trained domain dependent exclusively on easy data but were tested across all difficulty levels [9].

GNNs with max and mean aggregation functions were used as heuristic functions, both using four message-passing layers, 64 hidden dimension, and sum pooling function. For the WL-Goose models, which incorporate the WL-algorithm and graph kernels, the regression models considered were support vector regression with the dot product kernel (SVR) and the radial basis function kernel (SVR_∞), as well as Gaussian process regression (GPR) with the dot product kernel. Both the Goose and the WL-Goose models use the ILG graph representation to represent the planning problems. Each GOOSE model and

SVR model underwent training and evaluation five times, with mean scores reported, whereas the GPR model, due to its deterministic nature, was trained and evaluated only once [9].

To establish baseline heuristics, the study used the domain-independent heuristic h^{FF} (subsubsection 2.1.3.1) and Muninn [76][96] another GNN architectur for planning, which was adapted to learn heuristics for use in Greedy Best-First Search (GBFS). Additionally, the LAMA planner [13] was included as a robust satisficing planner baseline, utilizing its first plan output along with multiqueue heuristic search and other optimization techniques. All planners were allowed a maximum of 1800 seconds per evaluation problem. For the execution the study used for the Non-GNN models a cluster equipped with single Intel Xeon 3.2 GHz CPU cores and an 8GB memory limit, for the GOOSE models an NVIDIA RTX A6000 GPU and for Muninn an NVIDIA A10 GPU were used [9].

The score of a planner in the IPC for a solved task is the ratio C^*/C, where C is the cost of the cheapest discovered plan and C^* is the cost of a reference plan. The score for an unsolved task is 0. The overall score of a planner is the sum of its scores for all tasks [19]. So the higher the score the better the planner performed. The coverage measures the number of tasks a planner can solve within a given set. The optimal coverage for a problem domain in the IPC 2023 dataset would be 90. An analysis of the total coverage and IPC scores in Table 2.3 reveals that SVR and GPR outperform the other single queue heuristic planners but they can't surpas LAMA-first multiqueue heuristic search.

At a domain-specific level, SVR and GPR outperform Muninn and GOOSE GNNs across nine domains. They also either match or exceed LAMA's performance in four domains. Notably, GPR generates more optimal plans than LAMA in five domains, while LAMA outperforms GPR in only three domains (Rovers, Satellite, and Spanner). LAMA's poor performance in the Spanner domain is attributed to its heuristics lacking informativeness, causing it to operate like a blind search and thus yielding better plans only in solvable problems [9]. Moreover, SVR and GPR either outperform or match h^{FF} in six and seven domains, respectively. GOOSE GNNs surpass h^{FF} in three domains, LAMA-F only in the blocksworld domain, and GPR only in the Transport domain. GNNs with the max aggregation function show a slight performance edge over the mean aggregation function, solving 15 problems more in total, with the most significant differences in the childsnack and spanner domains.

As the complexity of problems increases, the number of problems solved by the planners decreases, as illustrated in Table 2.4. This trend is particularly pronounced in the context of domain drift, which seems to pose significant challenges for the different planners. This difficulty is expected, given that the GNNs and regression models based on kernels are only trained on easy problems. Consequently, these models struggle to adapt effectively to domains featuring medium and hard problems, as their heuristics follow a different distribution then the once seen in the easy training data. GPR is the top performer among the single queue heuristics because it translates best to medium and hard problems. Conversely, the performance differences among the different single queue heuristics is not that significant when considering only the easy problems.

	classical		GNN			WLF			
Domain	LAMA-F	h^{FF}	Muninn	GOOSE$^{‡}_{max}$	GOOSE$^{‡}_{mean}$	SVR‡	SVR$^{‡}_{\infty}$	SVR$^{‡}_{2\text{-LWL}}$	GPR
blocksworld	61	28	53	63.0	60.6	72.2	19.0	22.2	**75**
childsnack	35	26	12	23.2	15.6	25.0	13.0	9.8	**29**
ferry	68	68	38	70.0	70.0	**76.0**	32.0	60.0	**76**
floortile	11	**12**	1	0.0	1.0	2.0	0.0	0.0	2
miconic	90	**90**	**90**	88.6	86.8	**90.0**	30.0	67.0	**90**
rovers	67	34	24	25.6	28.8	**37.6**	28.0	33.6	37
satellite	89	**65**	16	31.0	27.4	46.0	29.4	19.0	53
sokoban	40	36	31	33.0	33.4	**38.0**	30.0	30.6	**38**
spanner	30	30	**76**	46.4	36.6	73.2	30.0	51.8	73
transport	66	**41**	24	32.4	38.0	30.6	27.0	34.2	29
all	557	430	365	413.2	398.2	490.6	238.4	328.2	**502**
IPC score	492.7	393.5	328.9	391.0	372.8	453.7	210.7	297.8	**461.3**

Table 2.3: Coverage of planners. The bottom-most row provides their overall IPC 2023 learning track score. Models marked ‡ are run 5 times with mean scores presented. LAMA-first is the only planner not performing single-queue GBFS. The top three single-queue heuristic search planners in each row are indicated by the cell colouring intensity, with the best one in bold. The best planner overall in each row is underlined.[9]

2.3.3.2 Domain Independent Results

For domain-independent heuristic learning the problems and domains from the 1998 to 2018 IPC dataset [97] were used. Five models were trained with optimal plans featuring unit costs, which were generated by Scorpion [32], an advanced classical planning system that enhances the capabilities of Fast Downward. The GNN models used in the experiments comprised of 64 hidden dimensions, 8 RGNN layers, using mean aggregation and a sum pooling function [6]. As graph representations the SLG, FLG anf LLG representations where used. Each representation was trained and tested separately to ensure thorough evaluation. The ILG representation was only developed later. The same holds for the WL-Goose kernel method so for them no domain independent results are available.

For comparison domain-dependent heuristic learning was also performed using the same models and graph representations, with adjustments made to the training process to suit the domain-dependent approach. In both training approaches, the models were initially trained on easy problems within the domains and later tested on easy, medium, and hard problems. In the study for all Goose models the training and execution of the GOOSE models was performed using a single NVIDIA GeForce RTX 3090 GPU. The evaluation process involved comparisons against several baseline methods: blind search, Fast Downward's eager GBFS with the h^{FF} heuristic [29] , and the domain-dependent STRIPS-HGN

	classical		GNN			WLF			
Difficulty	LAMA-F	h^{FF}	Muninn	$\mathrm{GOOSE}^{\ddagger}_{\mathrm{max}}$	$\mathrm{GOOSE}^{\ddagger}_{\mathrm{mean}}$	$\mathrm{SVR}^{\ddagger}$	$\mathrm{SVR}^{\ddagger}_{\infty}$	$\mathrm{SVR}^{\ddagger}_{\text{2-LWL}}$	GPR
easy	280	275	223	256.8 ±3.6	250.6 ±2.6	264.8 ±2.9	234.4 ±2.1	229.4 ±1.4	265
medium	190	112	96	109.6 ±7.4	105.8 ±10.3	152.6 ±1.0	4.0 ±0.0	91.8 ±2.2	161
hard	87	43	46	46.8 ±4.5	41.8 ±2.6	73.2 ±0.7	0.0 ±0.0	7.0 ±0.0	76
all	557	430	365	413.2 ±10.9	398.2 ±9.8	490.6 ±3.4	238.4 ±2.1	328.2 ±3.1	502

Table 2.4: Coverage of considered planners per difficulty level. The mean and standard deviation are taken for models with multiple repeats marked by ‡ [9].

[98]. STRIPS-HGN was configured according to the parameters from the original study but used the same dataset as GOOSE, and it was invoked from Fast Downward's eager GBFS for heuristic evaluation. The other baselines were run on CPUs. All baselines and GOOSE models were executed with a 600-second timeout and 8GB of main memory [6].

Table 2.5 displays the coverage results for different planners, with the number in parentheses indicating the maximum possible coverage for each domain. Except for Sokoban, the domain-independent GOOSE outperforms blind search, suggesting that the learned domain-independent heuristics are informative. The most effective graph representation in the domain-independent GOOSE models appears to be the SLG representation. This is likely because, compared to the LLG representation, SLG provides more information for the MPNN to learn the domain-independent heuristics without causing overfitting, which can occur with the more detailed FLG structure.

Figure 2.11 compares the number of expanded nodes and the plan cost for the most effective Goose models in both domain-dependent and domain-independent experimental setups, as well as the h^{FF} heuristic. Points in the bottom-right triangles indicate a preference for h^{FF}, while points in the top-left triangles indicate a preference for GOOSE. For configurations that do not solve a problem, the value is set to the maximum of the plot's axis. As observed in Figure 2.11, the domain-independent GOOSE with SLG generates higher quality plans and expands fewer nodes than h^{FF} in the VisitAll and VisitSome domains. However, compared to their domain-dependent counterparts, the domain-independent versions generally solve fewer problems, making the h^{FF} heuristic preferable in more domains [6].

	baselines			domain-dep.			domain-ind.		
	blind	h^{FF}	HGN	SLG	FLG	LLG	SLG	FLG	LLG
blocks (90)	-	19	-	-	6	**62**	9	8	6
ferry (90)	-	**90**	-	32	33	88	28	22	2
gripper (18)	1	**18**	5	9	6	**18**	5	3	9
n-puzzle (50)	-	**36**	-	10	10	-	6	3	-
sokoban (90)	74	**90**	10	31	29	34	45	40	15
spanner (90)	-	-	-	-	-	**60**	-	-	-
visitall (90)	-	6	25	46	**50**	44	16	41	-
visitsome (90)	3	26	33	72	39	65	**73**	65	15

Table 2.5: Coverage of planners and GOOSE over various domains. Cell intensities indicate the top 3 planners per row.[6]

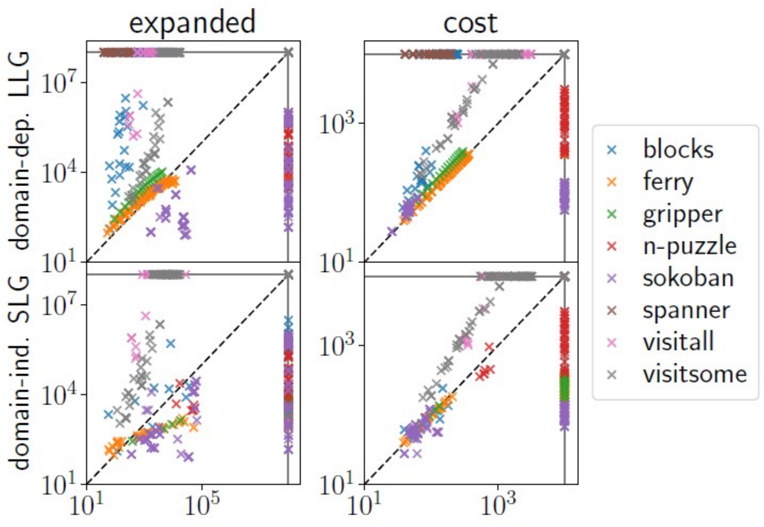


Figure 2.11: Comparison of GOOSE (x-axis) and h^{FF} (y-axis) on number of expanded nodes (left) and plan cost (right). [6]

2.4 Related Works

Neural networks have been increasingly applied in planning domains, offering various methods to address the challenges of solving planning tasks. By leveraging the strengths of neural networks in learning from data and recognizing patterns, these approaches enhance the structured, goal-oriented nature of planning. This section explores the related works that highlight the synergy between planning and neural networks. It is divided into two parts: the first provides an overview of the use of different neural network models, such as CNNs and feedforward models, in planning. The second part specifically focuses on GNNs, which have become more popular in recent years and are also the architecture used by Goose.

Most neural approaches aim to learn either a generalized policy or a heuristic function. In the context of planning generalized policies are mostly developed through reinforcement or imitation learning to create decision-making strategies that can adapt to a variety of problem instances, enabling real-time action selection in dynamic environments. In contrast, heuristic functions are learned through supervised learning to estimate the cost from a given state to the goal, guiding search algorithms by providing cost estimates that optimize the exploration of the state space. While generalized policies focus on directly determining the next action, heuristic functions indirectly influence the sequence of actions by shaping the search path, both contributing to efficient and effective planning.

2.4.1 Neural Networks

Over the years different neural architectures have been applied in automated planning. ASNet [15] introduces a neural network architecture designed for learning domain-dependent generalized policies in probabilistic planning problems. Utilizing weight sharing and binary vector representations derived from PPDDL descriptions, ASNet achieves the capability to generalize across different problem sizes within the same domain. Shen et al. [16] further enhance ASNet's capabilities by integrating it with Monte-Carlo Tree Search (MCTS), improving both generalization beyond trained problem distributions and navigation efficiency within the search space.

Groshev et al. [17] employ CNNs to learn generalized policies for the Sokoban domain, leveraging graphical representations of problems. The policies derived from the CNNs can also be used as heuristic functions in directed search algorithms, demonstrating CNNs' adaptability in spatial reasoning tasks.

Delfi [30], the online portfolio planner that won the optimal track of the 2018 International Planning Competition (IPC), treats planning tasks as images and employs CNNs to predict the probability of a planner successfully solving a given task within specified time and memory constraints.

Generalized Heuristic Networks (GHNs) [99] propose a method to learn generalizable heuristics in the absence of symbolic action models. By predicting actions and plan lengths from abstract state representations consisting of vectors and matrices, GHNs facilitate heuristic learning without requiring domain-specific knowledge, showcasing neural networks' flexibility in heuristic generation.

Ferber et al. [100] introduce domain-dependent heuristic functions, h^{Boot} and h^{AVI}, based on residual networks. These heuristics leverage bootstrapping and approximate value iteration during training data generation, comparing per-domain and per-instance learning approaches. Their findings indicate that per-instance learning generally outperforms per-domain learning in heuristic accuracy and effectiveness.

Chrestien et al. [101] focused on improving the loss function rather than enhancing the neural network models. Their approach utilizes ranking loss functions instead of absolute loss functions, based on the idea that the efficiency of a heuristic in forward search algorithms is controlled by the ranking of the states in the open list. This means it is more crucial for the heuristic to rank the states correctly than to achieve high accuracy. Their method of using a ranking loss can be integrated with various neural network approaches.

Silver et al. [102] explore the use of GPT-4 to automatically generate domain-specific Python programs that solve planning tasks. This approach not only automates task-specific programming but also enhances program performance through automated debugging. In certain domains, GPT-4 outperforms traditional planning methods like Fast Downward, showcasing the potential of LLMs in automating and optimizing planning processes.

In summary, neural networks have been utilized in planning for candidate planner selection and for learning both generalized policies and heuristic functions. These models typically focus on domain-dependent learning, training on problems within the same domain where they will be applied. With the advent of powerful GPT-based LLMs, there may be increased interest in leveraging the capabilities of LLMs in planning. Table 2.6 provides a summary of the papers discussed in this section.

Approach	**Category**	**Learning Outcome**	**Domains**
ASNet [15] [16]	Action Schema Network	Domain-dependent generalized policy	CosaNostra Pizza, Blocksworld, Triangle Tire World
E. Groshev et al. (2018) [17]	CNN	Domain-dependent generalized policy and heuristic function	Sokoban
Delfi [30]	CNN	Candidate planner selection	Domains of the classical tracks of all IPC, Genome Edit Distance Domain, Domains from Conformant-to-Classical Planning Compilation and Finite-State Controller Synthesis Compilation
GHNs [99]	FNN	Domain-dependent heuristic function	Blocksworld, Childsnack, Ferry, Goldminer, Grid, Gripper, Grippers, Logistics, Miconic, Sokoban, Sokoban2, Spanner, and Visitall.
h^{Boot}, h^{AVI} [100]	Residual network	Domain-dependent heuristic function	Blocks, Depots, Grid, Npuzzle, Pipes-nt, Rovers, Scanaly, Storage, Transport, Visitall
T. Silver et al. (2023) [102]	LLM (GPT 4.0 and GPT 3.5)	Domain-dependent generalized policy	Delivery, Forest, Gripper, Miconic, Ferry, Spanner, Heavy

Table 2.6: Overview related works neural networks in automated planning

2.4.2 Graph Neural Networks

Graph Neural Networks have been increasingly applied in planning domains, exploiting their ability to capture relational dependencies and structures within problems. This section reviews notable contributions in this field, highlighting advancements and limitations of GNN-based approaches.

Groshev et al. [17] employed GCNNs to learn generalized policies for the Traveling Salesman Problem. Their approach learns generalized policies that can also serve as heuristic functions within directed search algorithms, showcasing the utility of GNNs in combinatorial optimization tasks.

TORPIDO (Transfer of Reactive Policies Independent of Domains) [103] learns a transformation from the state and action spaces to latent state and action spaces and then accelerates policy learning by using GCNNs. Analyses of TORPIDO have shown that it is constrained to transferring policies only between problems of identical size, a limitation addressed by Garg et al. with TRAPSNET [104]. TRAP-

SNET uses the Asynchronous Advantage Actor-Critic (A3C) framework to facilitate neural transfer across different problem sizes in RDDL Markov Decision Processes (MDPs).

STRIPS-HGN [98] marks the first GNN approach that was able to learn domain-independent heuristics from scratch. In STRIPS-HGN, planning problems are modeled as grounded hypergraphs and hypergraph networks are trained over state-value pairs derived from optimal plans.

SYMNET [105] addresses relational Markov Decision Processes (RMDPs) formulated in the RDDL language by representing problems as dynamic Bayesian networks and training Graph Attention Networks (GATs) to derive domain-dependent policies. Despite its initial promise, SYMNET's effectiveness is hindered in some planning competition domains due to incomplete graph information during construction. This led to the development of SYMNET2.0 [106], which enhances graph construction with position-based graphs. SYMNET2.0 showed improved performance compared to SYMNET but struggled to learn policies that exploit long-range dependencies. This led to the introduction of SYMNET3.0 [107], which incorporates influence graphs to capture long-range dependencies.

Ma et al. [108] propose GNN-based methods for candidate planner selection and adaptive scheduling, utilizing lifted abstract structure graphs (ASGs) and grounded problem description graphs (PDGs) to encode planning tasks. They evaluate Graph Convolutional Networks (GCNs) and Gated Graph Neural Networks (GGNNs), highlighting their applicability in diverse planning scenarios.

Rivlin et al. [109] explore the use of GATs and deep reinforcement learning for learning generalized planning policies capable of handling larger problem instances from scratch without relying on pre-existing solutions or heuristics. This approach underscores the potential of GNNs in autonomously learning effective planning strategies.

Teichteil-Königsbuch et al. focus on Resource-Constrained Project Scheduling Problems (RCPSPs), proposing a GNN-based framework called SIREN [110]. Combining graph transformer models with schedule generation schemes, SIREN generates feasible solutions across various sizes and structures of RCPSPs, demonstrating the versatility of GNNs in complex scheduling problems.

Relational Graph Neural Networks (R-GNNs) [78] offer a new method for enhancing the expressive power of GNNs in classical planning scenarios. R-GNNs can extend expressive capabilities to C_3 features, which are essential for solving problems in specific domains, without the high memory and time requirements of 3-GNNs. Additionally, experimental results indicate that R-GNNs generalize better than edge-transformers.

In summary GNNs have been applied to a variety of planning domains, focusing on learning both heuristic functions and generalized policies. Among the different GNN architectures, GATs have emerged as the most popular choice in planning at the moment. Ongoing research for GNNs in planning is focused towards enhancing graph representations, adapting to more complex problems beyond the training data, and exploring transfer learning and domain-independent learning. These efforts aim to develop heuristics or policies that can be applied across different domains. Table 2.7 provides a summary of the papers discussed in this section.

Approach	Category	Learning Outcome	Domains
E. Groshev et al. (2018) [17]	GCNN	Domain-dependent generalized policies Domain-dependent heuristic	traveling salesman problem
TORPIDO [103]	GCNN	Domain-dependent generalized policies	SysAdmin, Game of Life, Navigation
TRAPSNET [104]	GAT and fully connected network	Domain-dependent generalized policies	SysAdmin, Game of Life, Academic Advising
STRIPS-HGN [98]	Hypergraph network	Domain-dependent and independent heuristic functions	8-puzzle, Sokoban, Ferry, Blocksworld, Gripper, Zenotravel
SYMNET [105]	GAT	Domain-dependent generalized policies	Academic Advising, Crossing Traffic, Game of Life, Navigation, Skill teaching, Sysadmin, Tamarisk, Traffic, Wildfire
T. Ma et al. (2019) [108]	GCNN and Gated graph network	Candidate planner selection	domains of the classical tracks of all IPC, genome edit distance (GEDP) domain, domains from conformant-to-classical planning compilation and finite-state controller synthesis compilation
O. Rivlin et al. (2020) [109]	GAT	Domain-dependent generalized policies	Blocksworld, Satellite, Logistics, Gripper, ferry
SYMNET 2.0 [106]	GAT	Domain-dependent generalized policies	Academic Advising, Crossing Traffic, Game of Life, Navigation, Skill Teaching, Sysadmin, Tamarisk, Traffic, Wildfire, Recon, Triangle Tireworld, Elevators
SIREN [110]	Graph transformer model	Domain-dependent heuristic	RCPSP instances from psplib
SYMNET 3.0 [107]	GAT	Domain-dependent generalized policies	Deterministic Navigation, Stochastic Corridor Navigation, Extreme Academic Advising, Safe recon, Pizza Delivery, Stochastic Wall
R-GNNs [78]	RGNN	Domain-dependent generalized policies	Blocks, Grid, Gripper, Logistics, Miconic, Rovers, Vacuum, Visitall

Table 2.7: Overview related works GNNs in automated planning

3 Experiments

3.1 Dataset

All experiments were performed using the dataset from the learning track of the 2023 International Planning Competition (IPC) [19] which includes the following ten domains of planning problems:

- **Blocksworld**: Classic planning problems that involve a set of blocks that must be stacked in a specific order. The goal is to achieve a target configuration of blocks starting from an initial, often random, configuration using a robotic arm.
- **Childsnack**: These problems involve planning the preparation and delivery of snacks to children. The goal is to ensure that all children receive their desired snacks within a set timeframe, considering constraints such as ingredient availability and preparation time.
- **Ferry**: In these problems, a ferry needs to transport vehicles and passengers between different locations. The challenge is to devise an optimal sequence of trips to move all items to their destinations while adhering to the ferry's capacity constraints.
- **Floortile**: The task is to lay floor tiles in a specified pattern on a grid. The planner must determine the sequence of tile placements to achieve the desired final pattern, considering constraints such as tile orientation and positioning rules.
- **Miconic**: These problems involve the operation of an elevator system in a building. The objective is to transport passengers to their desired floors efficiently, minimizing waiting and travel times while adhering to elevator capacity limits.
- **Rovers**: In these scenarios a set of robotic rovers on an extraterrestrial surface needs to be controlled. The goal is to collect data from various locations, handle different scientific instruments, and transmit the collected data back to a base station while managing the rovers' resources.

- **Satellite**: The task is to manage a satellite's operations to capture images of specified targets on Earth. The planner must schedule the satellite's activities, such as camera activation and data transmission, while considering constraints like orbital dynamics and resource limitations.
- **Sokoban**: In these puzzle-like problems, an agent must push boxes to designated storage locations within a warehouse. The challenge is to find a sequence of moves that positions all boxes correctly without getting stuck or blocking essential pathways.
- **Spanner**: These problems involve a set of workers who must use spanners to assemble machinery parts. The goal is to determine the sequence of actions required to complete the assembly, taking into account the availability of spanners and the dependencies between assembly steps.
- **Transport**: The task is to transport goods between different locations using a fleet of vehicles. The planner must optimize the routes and loading schedules to ensure timely delivery while considering constraints such as vehicle capacities and travel times.

Each domain includes instances categorized into easy, medium, and hard difficulties based on the number of objects in each instance. For example, easy blocksworld problems consist of 5 to 30 blocks, medium problems range from 35 to 150 blocks, and hard problems contain 160 to 500 blocks. For each domain, the training set includes up to 99 easy instances. The test set consists of 30 instances for each difficulty level — easy, medium, and hard — that were not part of the training set. The IPC provides a Python generator script for each domain that generates the problems. We will use the same problems that were previously generated for the experiments described in subsection 2.3.3.

The generated planning problems are defined using the Planning Domain Definition Language (PDDL) [20], a standard language used to specify planning problems and domains, including the initial state, goal state, objects, and domain of the planning problem. Listing 3.1 shows an example of a blocksworld problem defined in PDDL. Additionally, for every problem domain, a domain file in PDDL is provided, detailing the predicates and actions of the domain. The solutions to the planning problems in the training dataset are provided as .plan files, which contain the sequence of actions that constitutes the solution to a planning problem.

Listing 3.1: PDDL Example - Blocksworld problem

```
;; base case
;;
(define (problem blocksworld-02)
 (:domain blocksworld)
 (:objects b1 b2 - object)
 (:init
    (arm-empty)
    (clear b2)
    (on-table b2)
    (clear b1)
    (on-table b1)
)
 (:goal (and
    (clear b2)
    (on b2 b1)
    (on-table b1)
)))
```

3.2 Goals of the Experiments

The primary goal of the experiments conducted in this thesis is to enhance the estimation of the heuristic function using GNNs, thereby improving the overall performance of the GOOSE framework. From the current results achieved with GOOSE described in subsubsection 2.3.3.2 and subsubsection 2.3.3.1, two main areas have been identified for improvement.

Firstly, there is the challenge of domain adaptation. When training on easy problems but testing across easy, medium, and hard problems, the performance of GOOSE on medium and hard problems suffers. The focus of this thesis is to address this domain adaptation issue. By exploring different approaches such as retraining and multiheuristic search, the aim is to improve performance on medium and hard problems while maintaining good performance on easy problems. Secondly, there are specific domains, such as floortile, where GOOSE struggles to solve even easy problems. This issue likely stems from the limited expressiveness of the current GNN model. Although the main focus is on improving domain adaptation, exploratory experiments will be conducted to test more expressive models for learning on graphs to address this secondary issue.

Throughout all experiments, it is crucial to remember that the objective is to estimate a heuristic function used by a planner. The heuristic must be evaluated thousands or millions of times by a planner, depending on the problem size, to solve a problem within a certain time limit. Thus, the experiments must focus not only on the accuracy of the GNNs but also on their evaluation speed to achieve an overall improvement in the number of solved problems.

3.3 Experiment Setup

Unless otherwise specified in the detailed experiment descriptions, the following setup and configurations were used for the experiments in this thesis.

The GNNs employed follow the architecture described in subsection 2.3.2. Two hyperparameter configurations were utilized: one with 64 hidden units and 4 RGNN layers, and another with 32 hidden units and 2 RGNN layers. In both cases, the pooling function was set to sum, and the aggregation function was set to mean. The first configuration was chosen because it had proven effective in domain-dependent testing in a previous GOOSE study [18]. The second configuration aimed to investigate whether GNNs with fewer parameters could enhance performance due to their faster evaluation by the planner. To enable this comparison, the aggregation and pooling functions were kept consistent across both models. Hyperparameter tuning was not conducted, as the primary goal was to improve performance on medium and hard problems. Tuning the hyperparameters on the easy training dataset was not expected to yield significant improvements for domain adaptation to medium and hard problems.

The graph generation step of the GOOSE framework remains unchanged. During training, information about the initial and goal states is taken from the PDDL problem files, and the states of the solution are generated based on the provided solution in the .plan files. All states are then represented in the selected graph representation using the GOOSE framework. During testing, the states that need to be represented as graphs and for which heuristics need to be calculated are returned by the Fast Downward planner. These states are determined by performing a heuristic search using the eager Greedy Best-First Search (GBFS) algorithm of the Fast Downward planner with the custom GOOSE heuristics. Eager GBFS is a heuristic search algorithm used in the context of planning to identify the shortest path or the most efficient sequence of actions to achieve a goal. It operates by expanding the most promising node first, based on a heuristic function that estimates the cost from the current node to the goal [1]. For the GOOSE GNNs, the heuristic evaluation of successor states can be parallelized on GPU for each opened node, enhancing evaluation efficiency [6].

For training, the graphs of the easy training problems are split into training and validation datasets with a ratio of 85% training and 15% validation and processed in batches of 16. The training process utilizes an MSE loss function, consistent with the previous GOOSE studies [6] [9]. Models are trained using the ADAM optimizer with an initial learning rate of 0.001, reduced by a factor of 0.1 if the validation loss did not improve for 10 epochs. This approach allows for quick initial convergence and finer adjustments as training progresses, improving generalization and preventing overfitting. Early stopping is employed with a minimum learning rate threshold of 1×10^{-6}, ensuring efficient training by halting once improvements cease. For the case that early stopping is not reached, the maximum number of training epochs is set to 500.

Testing of the models is conducted with a timeout of 20 minutes per problem. In previous studies, timeouts of 10 minutes [6] and 30 minutes [9] were used. Thus, 20 minutes is deemed a reasonable amount of time for solving planning problems while ensuring testing times remain manageable within the time constraints of the thesis. If a problem in the testing dataset can not be solved within 20 minutes, it is counted as not solved. Experiments are performed domain-dependently, meaning that for each planning domain, a separate GNN model is trained and tested only on problems from that domain. Domain-dependent training was chosen as in the previous GOOSE studies it solved more problems, providing more starting points for improvement. Additionally, the literature review in section 2.4 showed that most works focus on domain-dependent heuristics and policies, offering more resources. Domain-independent training and testing could be explored in future work. All experiments were run on a cluster with an Nvidia RTX A5000 GPU and a single Intel Xeon 1.9 GHz CPU with a memory limit of 16GB.

As presented in subsection 2.3.1, GOOSE provides four different graph representations (SLG, FLG, LLG, ILG). Due to resource constraints (shared usage of the cluster) and time limitations of this thesis, not all experiments can be conducted for all graph types. Therefore, the experiments focuse on a single graph type: the ILG graph type. This graph type was chosen because, under the described experiment settings using the standard GOOSE setup, it is able to solve the most medium and hard problems as shown in Figure 3.1, indicating its suitability for domain adaptation. Additionally, it is typically the smallest representation in terms of the number of nodes and edges, consuming less memory compared to especially the grounded representations.

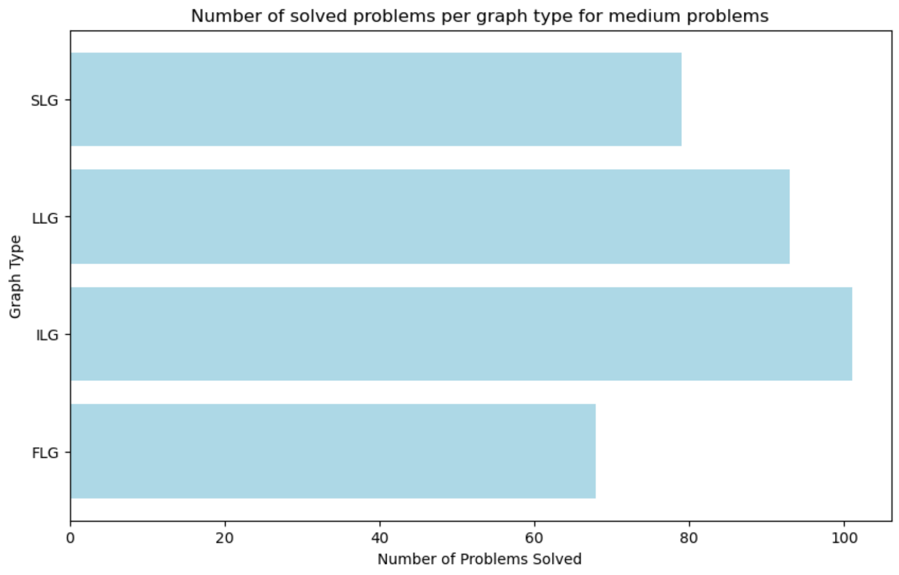


Figure 3.1: Comparison of the number of solved medium problems for the different graph representations provided by GOOSE. As we have 10 problem domains with 30 medium problems each, the maximum number of problems that could have been solved is 300.

3.4 Baseline Heuristics

To establish a foundation for evaluating the performance of the tested heuristics, we will conduct baseline experiments using the following heuristics:

- h_{blind}: The h_{blind} heuristic is a simple, uninformed heuristic defined as $h_{\text{blind}}(n) = 0$. This heuristic assigns a value of zero to every state n, providing no guidance to the search process. Consequently, all states are treated as equally desirable, and the search algorithm relies solely on other factors, such as the cost of actions in cost-based algorithms or the depth of states in depth-first search. Any heuristic proposed in this work must outperform h_{blind} to be considered informative.
- h^{ff}: The h^{ff} heuristic, detailed in subsubsection 2.1.3.1, will be used as a baseline due to its widespread use and extensive study within the planning community, providing a well-established standard for evaluating the performance of new heuristics. Furthermore, as it relies on relaxed planning graphs and graph algorithms, it offers a comparison to a non-neural network heuristic.
- **GOOSE**$_{\textbf{standard}}$: The standard GOOSE heuristic presented in [6], configured with 64 hidden units and 4 RGNN layers, utilizing a mean aggregation function and sum pooling, will be evaluated. The objective is to surpass the performance of this heuristic, especially on medium and hard problem instances.

All baseline heuristics are used with eager GBFS search, but only the heuristic GOOSE$_{\text{standard}}$ can parallelize the evaluation of successor states on the GPU for each opened node. The other baseline heuristics are evaluated using eager GBFS search on the CPU. Overall these baseline heuristics provide a comprehensive framework for assessing the effectiveness of the proposed heuristics. The comparison will help demonstrate the strengths and weaknesses of the new approaches in a variety of planning scenarios.

3.5 Fine Tuning by Retraining

The primary assumption underlying most machine learning algorithms, including the GNNs introduced in this thesis, is the independent and identically distributed (iid) nature of the data. This assumption is violated when the GNN models are trained on easy problems but tested on both medium and hard problems. The heuristics of the medium and hard planning problems follow different distributions compared to easy problems, presenting a challenge for generalization. To enable the GNN to adapt to new data distributions, we can use the domain adaptation methods introduced in subsection 2.2.7. One of the methods introduced there is fine-tuning through retraining. We chose this method for domain adaptation because it is the most convenient to integrate into the GOOSE framework.

We generate a second medium and hard dataset for the fine tuning with retraining approach. The IPC dataset includes a generator script for each domain that can be used to generate new problems. Thus, we can create 30 new medium and hard problems for the fine tuning. To maintain the same difficulty level as the initial medium and hard problems, we will use the same number of objects in each problem but vary the initial and goal states. This can be achieved by changing the random seeds in the generator script. For example, if the third problem in the initial medium blocksworld dataset involved stacking 42 blocks in a specific order, the third problem in the new medium test dataset will also involve 42 blocks, but the order of the blocks in the initial and goal states will be different. We are generating these new medium and hard datasets, to be able to use their problems in the retraining and then afterwards still have the initial medium and hard dataset for testing. By performing the retraining on a different dataset we ensure fairness in comparing model performance by testing on the same completely unseen medium and hard problems that the current GOOSE approach uses.

The initial training of the GNN is still conducted on the training set of easy problems. It is known from the results presented in subsection 2.3.3 that the GOOSE framework despite distribution shifts in certain planning problem domains can solve at least a few problems in the medium and hard difficulty levels. We aim to leverage this by applying the GNNs trained on easy problems to our newly generated medium training dataset and then using the solvable problems for retraining the network. The retraining process leverages the fact that problems are approximately ordered by difficulty within their respective levels. For instance, the first medium problem is easier to solve than the 15th medium problem, which is easier than the 30th, as the size of the problems increases within the difficulty level. We begin by attempting to solve the first 10 medium problems of the new medium dataset because, due to the difficulty ordering, the likelihood of solving those problems is higher than for the later ones. Solved problems, along with their solutions, are added to a retraining dataset, while unsolved problems are moved to a retesting dataset. If the retraining dataset contains solved problems, these problems, combined with the original easy training problems, are used for retraining. So for the retraining we have a training dataset containing easy and medium problems. Including the easy problems ensures the model retains its initial knowledge and remains effective for easy problems. These problems will persist in the retraining dataset for all subsequent retraining iterations. After retraining, the unsolved problems are retested to determine if the improved network can now solve some of these problems. This process continues iteratively for the next set of 10 problems until all 30 medium problems have been tested at least once. Then, the same process will be applied to the hard problems. Figure 3.2 illustrates the process of retraining a GNN in GOOSE.

It is crucial to note that one solved medium or hard problem does not equate to just one new graph for training. Each state of the solution, effectively each step, is represented as a graph. For instance, a medium problem with a solution length of 80 steps will contribute 80 new graphs to the retraining dataset. The described retraining approach should be well applicable in the GOOSE framework, as despite the addition of new data during retraining the dataset remains to be of relatively small size and the configuration of the GNNs, which are designed to be compact to be fast to evaluate, are also fast to train. The marginal training time compared to testing time ensures efficiency in adapting the model to new and more complex domains. Once the entire retraining process is completed, the obtained models are tested again on the initial medium and hard datasets to compare their performance to the previous GOOSE experiments. Through iteratively retraining and retesting, the GNNs are expected to hopefully have improved there performance, adapting to the distribution shifts and enhancing its capability to solve medium and hard problems more effectively.

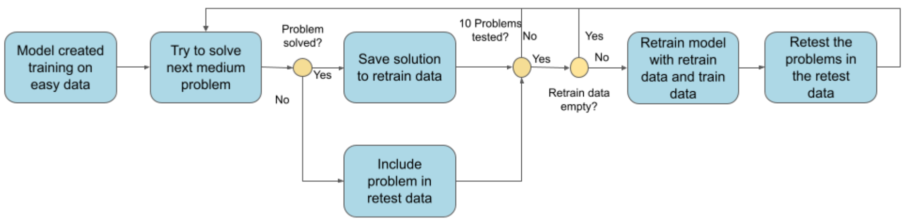

Model created training on easy data
Try to solve next medium problem
Problem solved?
Yes
No
Save solution to retrain data
Include problem in retest data
10 Problems tested?
No
Yes
Yes
No
Retrain data empty?
Retrain model with retrain data and train data
Retest the problems in the retest data

Figure 3.2: Retraining process used to improve the domain adaptation of the GOOSE GNNs.

3.6 Multi-heuristic Search

The Fast Downward planner supports multi-heuristic search, which means it can leverage multiple heuristics to solve a problem. These heuristics are combined by the planner using a weighted sum, with different weights assigned to each heuristic based on their perceived accuracy or importance [1]. Therefore, we evaluate how combining GOOSE with another heuristic will affect the number of solved problems in the test dataset, with the aim of increasing this number.

For this purpose, we use the h^{ff} heuristic, a standard heuristic within the Fast Downward planner and also one of our baseline heuristics, as presented in detail in subsubsection 2.1.3.1. The multi-heuristic search can also be used with eager GBFS search, so the search algorithm will remain the same. Since the h^{ff} heuristic is not implemented for parallelized evaluation on GPU, the multi-heuristic search will run only on the CPU. Combining h^{ff} heuristic evaluation on the CPU while evaluating GOOSE GNNs on the GPU is not feasible due to the implementation constraints of eager GBFS search in the Fast Downward planner. Expanding the planner implementation to handle this or evaluating h^{ff} on GPU would have exceeded the time limitations of this thesis.

We still decided to choose the h^{ff} heuristic as the second heuristic for the multi-heuristic experiments with GOOSE for two main reasons: first, as it is already used as a baseline heuristic, we are familiar with the setup and how to correctly call the heuristic. Second, from the results presented in subsubsection 2.3.3.2, we know that h^{ff} demonstrates superior performance over GOOSE in the satellite and transport domains. This suggests that graph-based techniques in some domains uncover insights that current GNNs fail to capture. Conversely, in domains such as blocksworld and spanner, GOOSE exhibits clear advantages over h^{ff}. The idea is that by combining these two heuristics, their complementary strengths are leveraged. For instance, in the satellite domain, Fast Downward assigns a higher weight to h^{ff}, while in the blocksworld domain, GOOSE would be weighted more. By this approach, we aim to achieve an overall enhancement in the framework's performance across various domains.

3.7 Multi-heuristic Search combined with Retraining

Based on the ideas from the previous sections, the next logical step to enhance the performance of the GOOSE frameworks involves combining the multi-heuristic search and the retraining. The rationale behind this strategy is to potentially solve a greater number of hard and medium-level problems by leveraging both GOOSE and the h^{ff} heuristic together. This expanded pool of solved problems can subsequently enhance the effectiveness of the GNN through retraining processes. Moreover, domains where previous retraining efforts did not yield improvements due to a scarcity of solved medium problems could now benefit from the combined influence of the GNNs and h^{ff}. It is anticipated that this approach will enable GOOSE to demonstrate enhanced performance in these domains during retraining phases. The retraining methodology will follow the same approach that is presented in section 3.5, focusing solely on the retraining of GOOSE's GNNs, as the h^{ff} heuristic relies exclusively on graph-based algorithms without trainable parameters.

For the retraining process, we will again use the newly generated medium and hard datasets, as presented in section 3.5. This approach ensures fairness and comparability to the current GOOSE methodology because using the same dataset for retraining and testing might give the models an unfair advantage if problems used for testing have already been seen during retraining. The strategy of using multi-heuristic search with retraining represents an effort to maximize potential improvements within the current GNN architecture. Subsequently, the forthcoming sections will explore changes to the architecture of the GNN and the use of more expressive models.

3.8 Graph Attention Networks

As presented in subsection 2.4.2 of the related works, GAT models have gained significant popularity for learning heuristics and generalized policies in planning tasks. To leverage this, we aim to incorporate the attention mechanism into the GOOSE framework to selectively focus on key features and filter out irrelevant information.

Given that all graph types in GOOSE are relational graphs and we would like to retain the information provided by the relational graph data. We will use RGAT layers introduced in subsection 2.2.4 instead of standard GAT layers. From the theoretical background, we know that there are two possible model configurations for RGAT models: WIRGAT and ARGAT. Additionally, we can choose between additive and multiplicative attention mechanisms. For inductive tasks, ARGAT with multiplicative attention is known to perform better, while for transductive tasks, WIRGAT with additive attention is slightly superior. In our case, as we are modeling a heuristic function with a GNN that performs graph-level predictions on during training unseen graphs, we are dealing with an inductive task. Therefore, we will employ ARGAT layers with a multiplicative attention mechanism.

Implementation-wise, switching from an R-GCN to an RGAT model involves replacing the RGNN layers with RGAT layers. The embedding layer, pooling layer, and MLP of the model architecture will remain unchanged as shown in Figure 3.3. For the RGAT layers, we utilize the `RGATConv` class from PyTorch Geometric [111]. To implement an ARGAT with multiplicative attention using this class, we need to set the hyperparameters `attention_mechanism = "across-relation"` and `attention_mode = "multiplicative-self-attention"`.

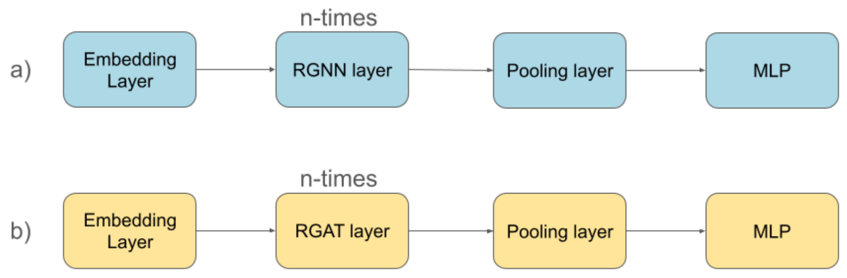


Figure 3.3: (a) Standard architecture of goose as presented in subsection 2.3.2 (b) GOOSE architecture incorporating RGAT layers

The experiments for the RGAT GOOSE models follow the standard GOOSE methodology, which involves initially training the model on easy problems and then testing it on easy, medium, and hard problems. Additionally, the retraining approach presented in section 3.5 is employed. This methodology allows us to evaluate whether the RGAT model initially performs better on medium and hard problems due to its ability to capture important patterns across all difficulty levels, or if it performs worse due to increased complexity and longer processing times by the planner. Furthermore, by applying the retraining approach, we aim to assess how well the RGAT model can be fine-tuned with a small amount of data and how effectively it can adapt to new patterns. For the evaluation of the retraining approach, we use the additional generated medium and hard training datasets to ensure fairness, ensuring that the model is tested on unseen problems during training, thus maintaining comparability with the initial GOOSE models.

RGAT models are still MPNNs, which have the expressiveness of the C2 fragment of first-order logic. Therefore, it cannot be expected that using an RGAT model will increase performance in problem domains that do not fall within the C2 class of first-order logic. Additionally, as indicated in subsection 2.2.4, RGAT models do not consistently outperform R-GCN models. Therefore, we need to thoroughly investigate the results obtained with the RGAT models to determine whether any improvement can be achieved in our specific context.

3.9 Masked Attention for Graphs

In these experiments, we aim to evaluate the performance of the GOOSE framework using the MAG model introduced in subsection 2.2.6, known for its more expressive modeling capabilities. The objective is to understand if a more expressive model like MAG enhances GOOSE's performance, keeping in mind that greater expressiveness does not always translate to better results due to the significant time required for heuristic evaluation, which may occur millions of times during a single planner run.

The MAG model will be integrated into the GOOSE framework by replacing the current GNN used for heuristic evaluation. The use of the Fast Downward planner by GOOSE will remain unchanged. Instead, the MAG model will be employed for evaluating the heuristics of given states within the Fast Downward planner, instead of the GNNs. This approach will also be applied to the training process, where the training configuration and data will remain the same, but the MAG model will be trained instead of the GNN model. When representing the planning problems as GOOSE graphs, minor adjustments are required in the graph generation process. Specifically, we need to generate an edge feature matrix to use as input when applying MAGE, as the current implementation lacks this matrix. The edge features will represent the different edge types of the GOOSE graphs. Additionally, we need to change the representation of the edge indices from a list of 2D tensors, where each list entry represents tensors of a different type, to a single 2D tensor representing the edges and another 1D tensor holding the edge type information. This format aligns with the requirements of the MAG implementation. As an orientation for the MAG model configuration, we use the setup used by Buterez et al. in their study [5] for benchmarking on graph-level tasks. The encoder is composed of three masked self-attention blocks and two self-attention blocks. Each block in the encoder is followed by a MLP, as recommended by Buterez et al. . Both the MLPs and the encoder blocks have 256 hidden dimensions, and the encoder blocks utilize 16 attention heads. Given the graph-level nature of our task, a decoder is required. We employ a single pooling by multihead attention block in the decoder, which also uses 256 hidden units and 16 attention heads. Notably, none of the blocks or MLPs incorporate dropout. Batch normalization is used as the normalization technique throughout the model.Figure 3.4 illustrates the architecture of the MAG model used in our experiments. For the sake of image size and readability, the MLPs following each block in the encoder are not depicted.

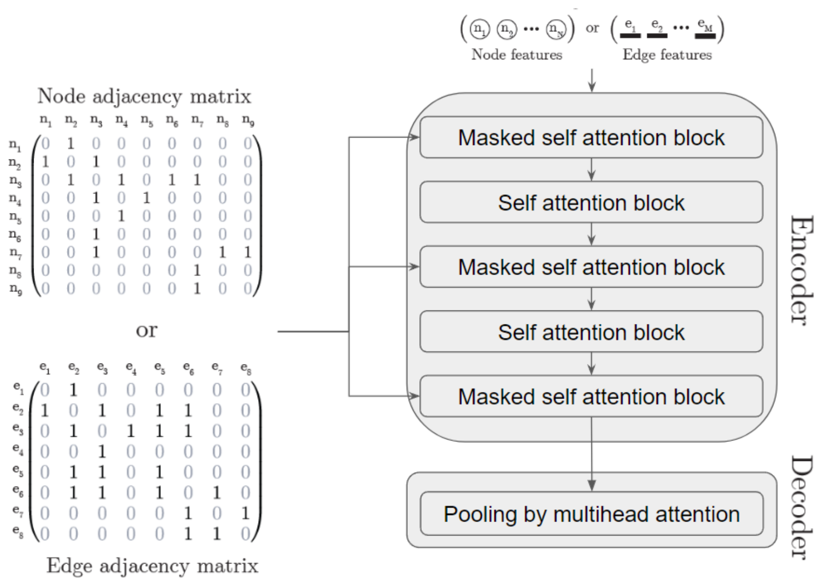


Figure 3.4: Architecture of the MAG model used for the experiments with GOOSE. The MLPs following each block in the encoder are not depicted.

While the implementation used in the study by Buterez et al. for graph level task is the MAG based on xformers, we switch to the PyTorch version of MAG to avoid compatibility issues with the GOOSE framework. In Buterez et al. study, the MAGE variant of MAG performed best for graph-level tasks. We will test both MAGE and MAGN approaches to determine whether node-based or edge-based representations yield better results with GOOSE, given that the graphs of our datasets typically contain more node features than edge features.

Due to the time constraints of this thesis, the MAG experiments will be the final ones conducted. Consequently, we may not have the opportunity to run these experiments across all domains and difficulty levels. Despite this limitation, we aim to gather valuable insights from the MAG experiments that we do manage to complete. Specifically, we hope to identify problem domains where GOOSE previously encountered challenges that could be better addressed with MAG. These preliminary results will help determine whether further exploration of the MAG approach is a possibility for future work.

4 Results

4.1 Variance

All results presented in the following sections that utilize GNN-based heuristics are non-deterministic, particularly those involving retraining. During retraining, different weights may be produced in each iteration, leading to varying outcomes. In a previous study on GOOSE [18], all GNN-based variants of GOOSE were run five times, and the average coverage was reported. However, due to the time constraints and limited computational resources available for this thesis, it was not feasible to replicate that approach. Instead, most experiments were repeated at least twice. The results presented are from the first run of each experiment, while the additional runs were used to estimate the variance. Variances can occur across all metrics, including the number of problems solved, the number of expanded nodes, and the cost of the generated plans. We assumed the variance to be consistent across all 10 problem domains in an experiment, and the variance was estimated across all difficulty levels. In practice, there may be deviations between the variances of individual problem domains and different difficulty levels, as the Fast Downward planner tends to perform more consistently on easier problems, meaning that small variations in heuristics have a lesser impact. However, with only two experiment runs, we could not confirm this, so we opted to estimate the variance across all difficulty levels. The results obtained using the heuristics h_{blind}, h^{ff}, and $GOOSE_{WL-GPR}$ are deterministic, meaning that rerunning the experiments yields identical outcomes. Therefore, we do not report variance for these experiments.

4.2 Fine Tuning by Retraining

The experiments to obtain the results for GOOSE with fine-tuning by retraining, referred to as $GOOSE_{retrain}$, were conducted as described in section 3.5. We applied retraining before testing on medium and hard problems. However, we did not perform any retraining for the easy problems, as the initial training problems are already easy problems, meaning the training and testing distributions are the same, so re-training is not necessary. Therefore, no performance difference is expected between $GOOSE_{retrain}$ and $GOOSE_{standard}$ on easy problems. The overall coverage - number of problems solved in the complete testing dataset including easy, medium and hard problems - by $GOOSE_{retrain}$ compared to the baseline approaches h_{blind}, h^{ff}, and $GOOSE_{standard}$ is shown in Table 4.1. For $GOOSE_{standard}$ and $GOOSE_{retrain}$, an estimate of the variance is provided. For example, the reported variance for $GOOSE_{standard}$ is ± 0.9, indicating that each result above the variance row has an uncertainty of approximately one problem solved more or less.

Domain	$\mathbf{h_{blind}}$	$\mathbf{h^{ff}}$	$\mathbf{GOOSE_{standard}}$	$\mathbf{GOOSE_{retrain}}$
blocksworld	8	27	61	53
childsnack	9	25	13	13
ferry	11	60	66	64
floortile	3	12	1	1
miconic	30	90	84	83
rovers	15	34	28	41
satellite	12	62	23	32
sokoban	28	36	31	34
spanner	30	30	34	61
transport	9	39	35	43
variance	0	0	$\pm$ 0.9	$\pm$ 1.9
all	155	421	373	425

Table 4.1: Coverage of the FastDownward planner using $GOOSE_{retrain}$ compared to the baseline heuristics. The top three heuristics in each row are highlighted with cell coloring intensity, with the best one in bold. The maximum number of problems that can be solved per domain is 90 (30 easy, 30 medium and 30 hard problems), so the maximum overall achievable coverage is 900.

Importantly, $GOOSE_{retrain}$ outperforms h_{blind}, confirming that it can be considered an informative heuristic. Additionally, $GOOSE_{retrain}$ outperforms $GOOSE_{standard}$, solving 425 problems compared to

373. The most significant improvement is observed in the spanner domain, where $GOOSE_{retrain}$ solves 27 more problems than $GOOSE_{standard}$. $GOOSE_{retrain}$ performs similarly to the h^{ff} heuristic, with 425 and 421 problems solved, respectively. While $GOOSE_{retrain}$ shows a clear advantage in the spanner and blocksworld domains, h^{ff} performs better in the satellite and childsnack domains. Given the variance associated with the $GOOSE_{retrain}$ results, the overall performance of these two heuristics can be considered equally.

As the primary goal of this thesis is to enhance the performance on medium and hard problems. Table 4.2 presents the number of medium and hard problems solved by $GOOSE_{retrain}$. For the baseline models h_{blind}, h^{ff} and $GOOSE_{standard}$ the table indicates the number of problems solved compared to $GOOSE_{retrain}$. For instance, the +6 in the rovers domain in the h^{ff} medium column signifies that $GOOSE_{retrain}$ solved 6 more medium rovers problems then h^{ff} was able to solve. On contrary the -7 in miconic domain in the h^{ff} hard column signifies that $GOOSE_{retrain}$ solved 7 problems less then h^{ff}. We can observe that the improvement made compared to h_{blind} is equal to the number of solved problems by $GOOSE_{retrain}$ in both the medium and hard domain meaning that h_{blind} does not manage to solve any medium or hard problem therefore every solved problem being an improvement.

Detailed analysis reveals that $GOOSE_{retrain}$ solves 38 more medium problems than the h^{ff} heuristic and 47 more than the $GOOSE_{standard}$ heuristic. $GOOSE_{retrain}$ shows improvements in five out of ten problem domains compared to $GOOSE_{standard}$. Notably, in the spanner domain, the retraining process enabled $GOOSE_{retrain}$ to solve all 30 medium problems, significantly surpassing the four problems solved by $GOOSE_{standard}$. In the ferry and miconic domains, all models, including $GOOSE_{retrain}$, solve all medium problems, indicating that retraining does not compromise previously learned patterns. Conversely, $GOOSE_{retrain}$, like $GOOSE_{standard}$, fails to solve any medium or hard floortile or childsnack problems, which makes sense as when we cannot solve a single medium problem there is no chance of improvement through retraining. The only performance decline for $GOOSE_{retrain}$ in the medium problems can be observed in the blocksworld domain, where it solves 20 problems compared to 28 by $GOOSE_{standard}$, suggesting potential model confusion due to retraining. Compared to the other baselines, $GOOSE_{retrain}$ does not offer a significant performance advantage for the hard problems. The issue appears to be that even though the models are retrained on the medium problems when available, transitioning to the hard problems still results in domain drift, with very few hard problems being solved. These few solved hard problems are likely insufficient to help the RGNN models overcome the domain drift in the retraining. In the upcoming sections, we will investigate whether multi-heuristic search or multi-heuristic search with retraining can address this issue and potentially further improve performance on medium and hard data.

	$\text{GOOSE}_{retrain}$		h_{blind}		h^{ff}		$\text{GOOSE}_{standard}$	
Domain	**medium**	**hard**	**medium**	**hard**	**medium**	**hard**	**medium**	**hard**
blocksworld	20	3	+20	+3	+20	+3	-8	0
childsnack	0	0	0	0	0	0	0	0
ferry	30	4	+30	+4	0	-2	0	+1
floortile	0	0	0	0	0	0	0	0
miconic	30	23	+30	+23	0	-7	0	-1
rovers	11	0	+11	0	+6	0	+10	0
satellite	9	0	+9	0	-21	-2	+9	0
sokoban	5	0	+5	0	-1	0	+3	0
spanner	30	1	+30	+1	+30	+1	+26	+1
transport	13	0	+13	0	+4	0	+7	0
all	148	31	+148	+31	+38	-7	+47	+1

Table 4.2: The first column lists the number of medium and hard problems solved using the $GOOSE_{retrain}$ heuristic. The remaining columns show the difference in this number compared to baseline heuristics. For example, a value of +20 under h^{blind} means $GOOSE_{retrain}$ solves 20 problems more than h^{blind} in that domain.

In addition to evaluating overall planner performance and detailed performance on medium and hard problems, it is interesting to examine the number of expanded nodes and the cost of the plans found. By comparing the number of expanded nodes and plan costs of the $GOOSE_{standard}$ heuristic to those of $GOOSE_{retrain}$, we can assess potential quality improvements in domains like miconic and ferry, where the number of solved medium problems is equal. Figure 4.1 illustrates this comparison focusing on medium problems due to their higher quantity of solved problems compared to hard problems and to remain readability compared to when including problems of all difficulty levels. The experiments revealed a variance of ± 35 in the number of expanded nodes and ± 1.5 in plan cost when using $GOOSE_{standard}$. For $GOOSE_{retrain}$, the variances were ± 41.3 and ± 2.3, respectively. Since the differences in variance are relatively small, comparing the results should be valid. The figure demonstrates that, regarding expanded nodes, a significant number of data points is located in the top-left triangle, indicating fewer nodes expanded using $GOOSE_{retrain}$. This is especially evident in the miconic domain (purple data points), where both heuristics solve all problems, but $GOOSE_{retrain}$ requires fewer expanded nodes, signifying more efficient heuristic performance.

In terms of plan cost, equivalent to plan length since all actions have a cost of 1 in our benchmarking domains, the two heuristics exhibit similar performance. Minor deviations are observed in the transport domain, favoring $GOOSE_{retrain}$, and the spanner domain, favoring $GOOSE_{standard}$, for problems solved by both heuristics but overall, the plan costs remain very similar. In summary, $GOOSE_{retrain}$ enables the planner to operate more efficiently, expanding fewer nodes while achieving plans of equivalent cost to those found using $GOOSE_{standard}$.

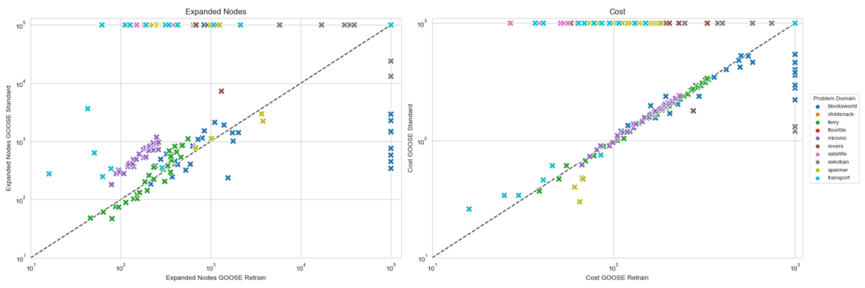


Figure 4.1: The number of expanded nodes and the plan cost of the medium problems of $GOOSE_{standard}$ and $GOOSE_{retrain}$. For problems not solved by one planner, the corresponding metric is set to the axis limit. Points located in the top-left triangle favor $GOOSE_{retrain}$, while those in the bottom-right triangle favor $GOOSE_{standard}$.

4.3 Multi-heuristic Search

This section presents the results obtained by combining the h^{ff} heuristic and the $GOOSE_{standard}$ heuristic for multiheuristic search using the Fast Downward planner. The experiments were conducted as described in section 3.6. In the following presentation of the results, the multiheuristic approach will be referred to as $GOOSE_{mh}$.

The overall performance of the Fast Downward planner using $GOOSE_{mh}$ as a heuristic, compared to the baseline heuristic and the $GOOSE_{retrain}$ heuristic presented in the previous section, is shown in Table 4.3. Importantly $GOOSE_{mh}$ clearly outperforms h_{blind} so it can be considered an informative heuristic. In addition it also outperforms the other baseline heuristics and manages to outperform $GOOSE_{retrain}$, solving 442 problems compared to 425. Examining the individual domains, we find that $GOOSE_{mh}$ generally achieves coverage between that of h^{ff} and $GOOSE_{standard}$. This can be explained by the planner assigning different weights to each heuristic based on their performance during the search. Exceptions are the spanner and transport domains, where $GOOSE_{mh}$ solves more problems than either $GOOSE_{standard}$ or h^{ff} alone. In these cases, the planner leverages information from both heuristics to achieve superior performance. Notably, in the transport domain, $GOOSE_{mh}$ outperforms all other heuristics. As our primary goal was to improve the performance on medium and hard problems, we will closely examine how $GOOSE_{mh}$ performed in these categories compared to the baseline heuristics and $GOOSE_{retrain}$. Table 4.4 shows the number of solved medium and hard problems for $GOOSE_{mh}$ and how this number differs from the baseline heuristics and $GOOSE_{retrain}$. The baseline heuristic h_{blind} was omitted as we know from the previously presented results that it does not solve any medium or hard problems, so comparisons are not that informative.

Domain	**h**$_{blind}$	**h**^{ff}	**GOOSE**$_{standard}$	**GOOSE**$_{retrain}$	**GOOSE**$_{mh}$
blocksworld	8	27	61	53	49
childsnack	9	25	13	13	18
ferry	11	66	63	64	63
floortile	3	12	1	1	9
miconic	30	90	84	83	81
rovers	15	34	28	41	36
satellite	12	62	23	32	46
sokoban	28	36	31	34	36
spanner	30	30	34	61	56
transport	9	39	35	43	48
variance	0	0	$\pm$ 0.9	$\pm$ 1.9	$\pm$ 0.8
all	155	421	373	425	442

Table 4.3: Coverage of the FastDownward planner using *GOOSE*$_{mh}$ compared to the baseline and *GOOSE*$_{retrain}$ heuristic. The top three heuristics in each row are highlighted with cell coloring intensity, with the best one in bold. The maximum number of problems that can be solved per domain is 90 (30 easy, 30 medium and 30 hard problems), so the maximum overall achievable coverage is 900.

We observe that for medium problems, *GOOSE*$_{mh}$ yields significant improvements compared to h^{ff} and *GOOSE*$_{standard}$, solving 41 and 50 more problems, respectively. This improvement is mainly due to *GOOSE*$_{mh}$ leveraging the strengths of both heuristics. For example, h^{ff} performs well in the satellite domain where *GOOSE*$_{standard}$ struggles, and the opposite is true for the blocksworld domain. *GOOSE*$_{mh}$ solves 19 problems in the blocksworld domain and 16 in the satellite domain at a medium level. While not achieving top performance in any single domain, *GOOSE*$_{mh}$ maintains a balanced overall performance across most domains, unlike the single heuristics which tend to fail in some domains. Compared to *GOOSE*$_{retrain}$, the overall performance of *GOOSE*$_{mh}$ on medium problems is similar, with *GOOSE*$_{mh}$ solving overall three more problems. Given the variance in the results, this difference is considered negligible. A closer look at the domains reveals that the satellite domain benefits most from the multi heuristic search, while the rovers and spanner domains benefit more from retraining.

For hard problems, *GOOSE*$_{mh}$ does not show any performance advantage compared to the other models. This is likely because the single heuristics h^{ff} and *GOOSE*$_{standard}$ do not perform well on hard problems. Which makes combining them risky as they are more likely to be misleading, thus misguiding the search of the Fast Downward planner and ultimately leading to fewer solved problems. Additionally, when both heuristics perform poorly, it becomes harder for the planner to assign appropriate weights to them, as it needs to determine which is the least ineffective heuristic.

	GOOSE$_{mh}$		**h^{ff}**		**GOOSE$_{standard}$**		**GOOSE$_{retrain}$**	
Domain	**medium**	**hard**	**medium**	**hard**	**medium**	**hard**	**medium**	**hard**
blocksworld	19	0	+19	0	-9	-3	-1	-3
childsnack	0	0	0	0	0	0	0	0
ferry	30	3	0	-3	0	0	0	-1
floortile	0	0	0	0	0	0	0	0
miconic	30	21	0	-9	0	-3	0	-2
rovers	6	0	+1	0	+5	0	-5	0
satellite	16	0	-14	-2	+16	0	+7	0
sokoban	6	0	0	0	+4	0	+1	0
spanner	26	0	+26	0	+22	0	-4	-1
transport	18	0	+9	0	+12	0	+5	0
all	151	24	+41	-14	+50	-6	+3	-7

Table 4.4: The first column lists the number of medium and hard problems solved using the *GOOSE$_{mh}$* heuristic. The remaining columns show the difference in this number compared to baseline and previously introduced heuristics. For example, a value of +19 under h^{ff} means *GOOSE$_{mh}$* solves 19 problems more than h^{ff} in that domain. The baseline h_{blind} was omitted because it does not solve any medium or hard problems.

As observed on the medium problems, *GOOSE$_{mh}$* and *GOOSE$_{retrain}$* exhibit similar overall performance. Therefore, we will also evaluate the number of expanded nodes and the cost of the found plans of the medium problems to determine if these metrics indicate a preferable approach. Figure 4.2 compares the number of expanded nodes and the plan costs for *GOOSE$_{mh}$* against *GOOSE$_{standard}$* and *GOOSE$_{retrain}$*. The variances of *GOOSE$_{mh}$* are with ± 39.1 for the expanded nodes and ± 1.4 for the plan cost similar to the variances of *GOOSE$_{standard}$* and *GOOSE$_{retrain}$*, so they should not falsify the comparison. Data points in the upper left triangle represent problems where *GOOSE$_{mh}$* is favored. In comparison to *GOOSE$_{standard}$*, *GOOSE$_{mh}$* generally requires fewer expanded nodes across all domains except the blocksworld domain, making it the preferred heuristic. The plan costs for *GOOSE$_{mh}$* and *GOOSE$_{standard}$* are very similar, with minor deviations favoring *GOOSE$_{standard}$* in the spanner domain and *GOOSE$_{mh}$* in the transport domain.

When comparing *GOOSE$_{mh}$* with *GOOSE$_{retrain}$*, *GOOSE$_{retrain}$* is generally the preferred heuristic in terms of the number of expanded nodes. Specifically, performance in the miconic domain is very similar, while in the transport domain, preference varies from problem to problem. The most significant differences are observed in the spanner, blocksworld, and rovers domains, where $GOOSE_{retrain}$ usually expands fewer nodes and is thus the preferred heuristic. Regarding plan cost, both $GOOSE_{mh}$ and $GOOSE_{retrain}$ produce relatively similar results. While there are some outliers where one heuristic finds a significantly better solution, there is no consistent occurrence where one heuristic consistently finds cheaper plans across an entire problem domain.

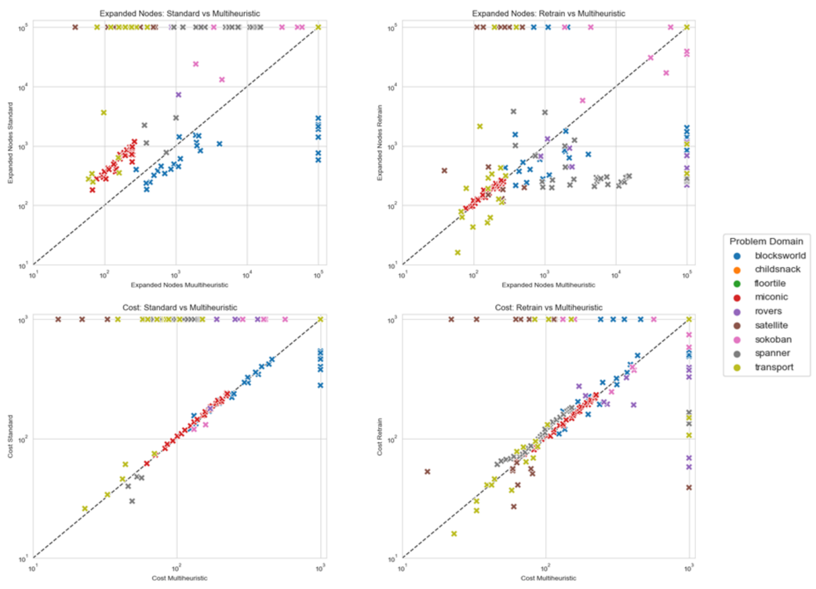


Figure 4.2: The number of expanded nodes and the plan cost for $GOOSE_{standard}$ compared to $GOOSE_{mh}$ (graphs on the left) and $GOOSE_{retrain}$ compared to $GOOSE_{mh}$ (graphs on the right) of the medium problems. For problems unsolved by one planner, the corresponding metric is set to the axis limit. Points located in the top-left triangle favor $GOOSE_{mh}$.

Overall, we can summarize that $GOOSE_{mh}$ provides a more balanced performance across all domains, particularly with medium problems. It has fewer domains where it fails to solve any problems compared to other heuristics. This is because $GOOSE_{mh}$ delivers decent performance in any domain where either $GOOSE_{standard}$ or h^{ff} perform well, effectively leveraging the strengths of both heuristics. In terms of the overall number of medium problems solved, $GOOSE_{mh}$ performs similar to $GOOSE_{retrain}$. However, when considering the number of expanded nodes as an additional quality metric, we conclude that $GOOSE_{retrain}$ is more efficient than $GOOSE_{mh}$ for medium problems. For hard problems, we observed that $GOOSE_{mh}$ does not show a performance improvement. This is likely because the individual heuristics, $GOOSE_{standard}$ and h^{ff}, only manage to solve a few problems on their own, and $GOOSE_{mh}$ cannot significantly benefit from combining these two heuristics.

4.4 Multi-heuristic Search combined with Retraining

As demonstrated in the previous sections, incorporating retraining or utilizing multiheuristic search within the GOOSE framework enhances performance compared to the $GOOSE_{standard}$ approach. Consequently, we combined these two methods to investigate whether this would lead to further performance improvements. This section presents the results achieved using multiheuristic search combined with retraining. The experiments were conducted as described in section 3.7, and this combined approach is referred to as $GOOSE_{mh-retrain}$.

Table 4.5 illustrates the overall coverage using $GOOSE_{mh-retrain}$ as a heuristic compared to the previously presented heuristics and the baseline heuristics. Importantly, $GOOSE_{mh-retrain}$ outperforms h_{blind}, proving that the heuristic is informative. Furthermore, considering overall performance, $GOOSE_{mh-retrain}$ successfully outperforms both $GOOSE_{retrain}$ and $GOOSE_{mh}$ by solving 40 more problems than $GOOSE_{retrain}$ and 23 more problems than $GOOSE_{mh}$.

Domain	$\mathbf{h}_{blind}$	$\mathbf{h}^{ff}$	$\mathbf{GOOSE}_{standard}$	$\mathbf{GOOSE}_{retrain}$	$\mathbf{GOOSE}_{mh}$	$\mathbf{GOOSE}_{mh-retrain}$
blocksworld	8	27	61	53	49	57
childsnack	9	25	13	13	18	18
ferry	11	66	63	64	63	64
floortile	3	12	1	1	9	9
miconic	30	90	84	83	81	79
rovers	15	34	28	41	36	43
satellite	12	62	23	32	46	53
sokoban	28	36	31	34	36	36
spanner	30	30	34	61	56	60
transport	9	39	35	43	48	46
variance	0	0	± 0.9	± 1.9	± 0.8	± 1.8
all	155	421	373	425	442	465

Table 4.5: Coverage of the FastDownward planner using $GOOSE_{mh-retrain}$ as a heuristic compared to the baseline heuristics, $GOOSE_{retrain}$ and $GOOSE_{mh}$. The top three heuristics in each row are highlighted with cell coloring intensity, with the best one in bold. The maximum number of problems that can be solved per domain is 90 (30 easy, 30 medium and 30 hard problems), so the maximum overall achievable coverage is 900.

A detailed anaylsis of individual domains reveals that in the satellite, rovers, and blocksworld domains, $GOOSE_{mh-retrain}$ solves more problems than either $GOOSE_{retrain}$ or $GOOSE_{mh}$. This suggests that by combining retraining with multiheuristic search, we leverage the strengths of both approaches to solve problems that neither $GOOSE_{mh}$ nor $GOOSE_{retrain}$ could solve independently. The likely reason for this improvement is that retraining the network based on the results of the multiheuristic search tailors the networks specifically for this search method. In contrast, the RGNNs used in the $GOOSE_{mh}$ approach are not specifically trained for multiheuristic search but are the same as those used in $GOOSE_{standard}$. In other domains, the performance of $GOOSE_{mh-retrain}$ is very similar to that of $GOOSE_{mh}$, indicating that in those domains retraining does not create a performance gain but also does not result in any information loss. Similar to the previously presented heuristics, $GOOSE_{mh-retrain}$ outperforms all baseline heuristics.

A detailed examination of the performance on medium and hard problems, as displayed in Table 4.6, reveals that compared to $GOOSE_{retrain}$, $GOOSE_{mh}$, and $GOOSE_{standard}$, the $GOOSE_{mh-retrain}$ approach either improves or maintains performance on medium problems across nearly all domains. This results in an overall improvement of 72 more solved medium problems compared to $GOOSE_{standard}$, 25 more than $GOOSE_{retrain}$, and 24 more than $GOOSE_{mh}$. When compared to the h^{ff} heuristic, $GOOSE_{mh-retrain}$ under performs only in the satellite domain, solving 7 fewer problems. In all other domains, $GOOSE_{mh-retrain}$ matches or exceeds the performance of h^{ff} on medium problems. The most significant improvements $GOOSE_{mh-retrain}$ achieves in the transport and rovers domains. In these domains, retraining appears to mitigate domain drift and enhance overall performance the most. Similar to the other heuristics $GOOSE_{mh-retrain}$ does not solve any medium problems in the childsnack or floortile domains. An improvement in those domains using $GOOSE_{mh-retrain}$ could not be expected because $GOOSE_{mh}$ also fails to solve problems in these domains, leaving no examples for retraining to improve performance.

Regarding hard problems, $GOOSE_{mh-retrain}$ does not perform as well as on medium problems. It solves fewer hard problems compared to h^{ff} and $GOOSE_{retrain}$, and has a performance similar to $GOOSE_{standard}$ and $GOOSE_{mh}$. This could be due to several reasons: first, retraining is ineffective if no hard problems are solved initially, which is the case for most domains. Additionally, combining heuristics that individually do not perform well can mislead and slow down the search, as it seems to be difficult for the planner to identify the least suboptimal heuristic in the multiheuristic search. From the same issue already the performance of the $GOOSE_{mh}$ heuristic on hard problems is affected. Moreover, since two heuristics need to be evaluated for each state, the search process is slower compared to evaluating a single heuristic like h^{ff}. While $GOOSE_{mh-retrain}$ can compensate for this slower evaluation speed on medium problems by overcoming domain drift with retraining, this is not feasible for most domains in the hard problems. Given the increased complexity of hard problems the number of evaluations of the heuristic increases making the evaluation speed increasingly important.

	$GOOSE_{mh-retrain}$		h^{ff}		$GOOSE_{standard}$		$GOOSE_{retrain}$		$GOOSE_{mh}$	
Domain	**medium**	**hard**	**medium**	**hard**	**medium**	**hard**	**medium**	**hard**	**medium**	**hard**
blocksworld	27	0	+27	0	-1	0	+7	0	+8	0
childsnack	0	0	0	0	0	0	0	0	0	0
ferry	30	4	0	-2	0	+1	0	0	0	+1
floortile	0	0	0	0	0	0	0	0	0	0
miconic	30	19	0	-11	0	-5	0	-4	0	-2
rovers	13	0	+8	0	+12	0	+2	0	+7	0
satellite	23	0	-7	-2	+23	0	+14	0	+7	0
sokoban	6	0	0	0	+4	0	+1	0	0	0
spanner	30	0	+30	0	+26	0	0	-1	+4	0
transport	14	0	+5	0	+8	0	+1	0	-2	0
all	173	23	+63	-15	+72	-4	+25	-5	+24	-1

Table 4.6: The first column lists the number of medium and hard problems solved using the $GOOSE_{mh-retrain}$ heuristic. The remaining columns show the difference in this number compared to baseline and previously introduced heuristics. For example, a value of +27 under h^{ff} means $GOOSE_{mh-retrain}$ solves 27 problems more than h^{ff} in that domain. The baseline h_{blind} was omitted because it does not solve any medium or hard problems.

We also conduct an analysis of the number of expanded nodes and the cost of the plans generated using $GOOSE_{mh-retrain}$ as an heuristic for the medium-sized problems. Figure 4.4 presents a comparison of the number of expanded nodes and plan costs when using $GOOSE_{mh-retrain}$ versus $GOOSE_{standard}$, and $GOOSE_{mh-retrain}$ versus $GOOSE_{mh}$ as heuristics. The variances of $GOOSE_{mh-retrain}$ are with ± 41.2 for the expanded nodes and ± 2.2 for the plan cost similar to the variances of $GOOSE_{standard}$ and $GOOSE_{mh}$, so they should not mislead the comparison. Compared to $GOOSE_{standard}$, $GOOSE_{mh-retrain}$ generally outperforms $GOOSE_{standard}$ in terms of the number of expanded nodes across all domains, with the exception of the blocksworld and ferry domains. In the ferry domain, the distribution is relatively balanced: for approximately half of the problems, $GOOSE_{standard}$ results in fewer expanded nodes, while for the other half, $GOOSE_{mh-retrain}$ leads to fewer expanded nodes. Therefore, there is no clear preference for one heuristic over the other in this domain. In the blocksworld domain, $GOOSE_{standard}$ results in fewer expanded nodes for the majority of problems. However, since this is the only domain where $GOOSE_{standard}$ is preferred, we can conclude that, overall, $GOOSE_{mh-retrain}$ is the more effective heuristic in terms of reducing the number of expanded nodes by the Fast Downward planner. When comparing $GOOSE_{mh-retrain}$ with $GOOSE_{mh}$, we observe a significant reduction in the number of expanded nodes in the spanner domain when using $GOOSE_{mh-retrain}$. In other domains, the performance is more balanced: for some problems, $GOOSE_{mh-retrain}$ results in fewer expanded nodes, while for others, $GOOSE_{mh}$ is more efficient. Considering the overall data distribution the graph indicates that $GOOSE_{mh-retrain}$ is generally the preferred heuristic as we have more data points in the top left triangle. Regarding the plan costs, there is no significant difference between the plans generated by $GOOSE_{mh-retrain}$ and those generated by $GOOSE_{standard}$, with most data points located close to the identity line. This suggests that neither heuristic consistently produces more optimal plans. In the comparison between $GOOSE_{mh-retrain}$ and $GOOSE_{mh}$, the data points are slightly more scattered, particularly in the satellite domain, where there is a noticeable variation in plan costs. However, there is no clear tendency favoring either heuristic overall. Thus, we conclude that there is no definitive winner between $GOOSE_{mh-retrain}$ and $GOOSE_{mh}$ in terms of producing more optimal plans.

In conclusion, $GOOSE_{mh-retrain}$ performs very well on medium-sized problems, effectively combining the strengths of the previously tested heuristics, $GOOSE_{mh}$ and $GOOSE_{retrain}$. Notably, it outperforms both $GOOSE_{mh}$ and $GOOSE_{standard}$ in terms of the number of expanded nodes, as it requires fewer nodes to be expanded on average for medium problems. The plan costs generated by $GOOSE_{mh-retrain}$ are comparable to those found using $GOOSE_{mh}$ and $GOOSE_{standard}$. On hard problems $GOOSE_{mh}$ does not perform as well as on the medium problems and does not manage any other heuristic except h_{blind}. In the hard problems, the benefits of retraining diminish, and the faster evaluation time of the single heuristics result in a small performance advantage. In addition the combination of individual heuristics used in the multiheuristic search, which already do not perform well on difficult problems, likely misguides and slows down the search process, reducing the overall effectiveness of the planner.

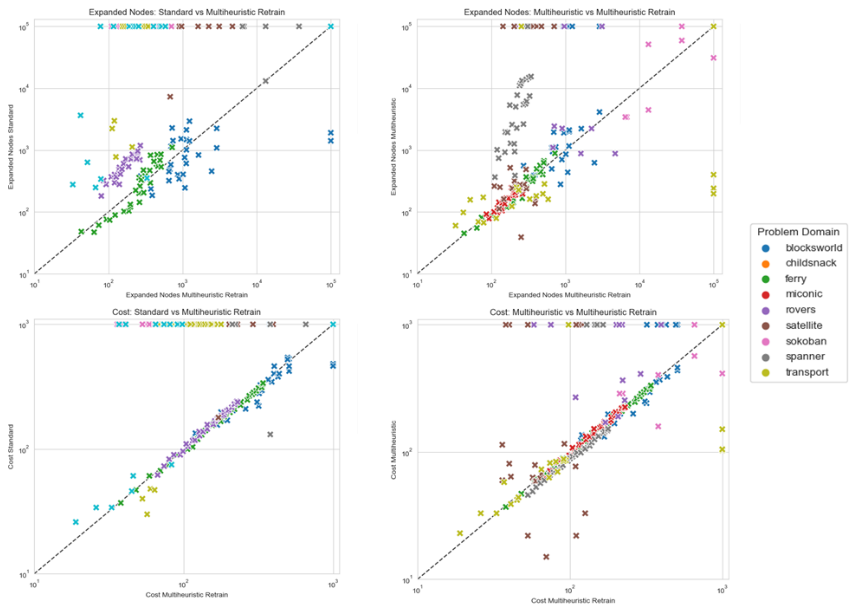


Figure 4.3: The number of expanded nodes and the plan cost for $GOOSE_{standard}$ compared to $GOOSE_{mh-retrain}$ (graphs on the left) and $GOOSE_{mh}$ compared to $GOOSE_{mh-retrain}$ (graphs on the right) of the medium problems. For problems unsolved by one planner, the corresponding metric is set to the axis limit. Points located in the top-left triangle favor $GOOSE_{mh-retrain}$.

4.5 Relational Graph Attention

The previously presented results explored the potential of enhancing the performance of the standard RGNN Models within the GOOSE framework. In this section we discuss the outcomes achieved by transitioning from RGNN to RGAT models described in subsection 2.2.4. The experiments followed the methodology outlined in section 3.8, with this approach referred to as $GOOSE_{gat}$.

Table 4.7 provides a comparison of the overall coverage of the Fast Downward planner using $GOOSE_{gat}$ as a heuristic, alongside the results of the previously tested heuristics and baseline heuristics. The results show that $GOOSE_{gat}$ surpasses h_{blind} in the overall coverage, confirming its status as an informative heuristic, and also outperforms $GOOSE_{standard}$. However, when compared to the other baseline approache hff, the performance of $GOOSE_{gat}$ is inferior. Similarly, $GOOSE_{gat}$ performs worse than our previously tested heuristics $GOOSE_{mh}$, $GOOSE_{mh-retrain}$, and $GOOSE_{retrain}$.

Domain	**h**$_{blind}$	**h**^{ff}	**GOOSE**$_{standard}$	**GOOSE**$_{retrain}$	**GOOSE**$_{mh}$	**GOOSE**$_{mh-retrain}$	**GOOSE**$_{gat}$
blocksworld	8	27	61	53	49	57	55
childsnack	9	25	13	13	18	18	20
ferry	11	66	63	64	63	64	64
floortile	3	12	1	1	9	9	1
miconic	30	90	84	83	81	79	83
rovers	15	34	28	41	36	43	31
satellite	12	62	23	32	46	53	31
sokoban	28	36	31	34	36	36	32
spanner	30	30	34	61	56	60	55
transport	9	39	35	43	48	46	39
variance	0	0	± 0.9	± 1.9	± 0.8	± 1.8	± 0.9
all	155	421	373	425	442	465	411

Table 4.7: Coverage of the FastDownward planner using $GOOSE_{gat}$ as a heuristic compared to the baseline heuristics, $GOOSE_{retrain}$, $GOOSE_{mh}$ and $GOOSE_{mh-retrain}$. The top three heuristics in each row are highlighted with cell coloring intensity, with the best one in bold. The maximum number of problems that can be solved per domain is 90 (30 easy, 30 medium and 30 hard problems), so the maximum overall achievable coverage is 900.

Notably, the childsnack domain is the only problem domain where $GOOSE_{gat}$ shows superior performance compared to the previously presented heuristics but it does also not outperform the baseline heuristic h^{ff}. In the other domains $GOOSE_{gat}$ either matches the performance of previously tested heuristics or falls short, offering no improvement. Furthermore, $GOOSE_{gat}$ also does not enhance the performance in the floortile domain where most of the other heuristics have also shown suboptimal performance. The overall performance of $GOOSE_{gat}$ suggests that incorporating attention mechanisms into the heuristic models does not enhance the heuristic's effectiveness. We will again conduct a more detailed analysis of the medium and hard problem instances to assess whether $GOOSE_{gat}$ also exhibits suboptimal performance in these cases, or if improvements can be identified. Table 4.8 presents a comparison of the performance of $GOOSE_{gat}$ on medium and hard problems against the baseline models and $GOOSE_{mh-retrain}$, our so far best-performing heuristic on medium problems. For readability, $GOOSE_{mh}$ and $GOOSE_{retrain}$ have been omitted from the table.

	GOOSE$_{gat}$		**h^{ff}**		**GOOSE$_{standard}$**		**GOOSE$_{mh-retrain}$**	
Domain	**medium**	**hard**	**medium**	**hard**	**medium**	**hard**	**medium**	**hard**
blocksworld	26	0	+26	0	-2	-3	-1	0
childsnack	0	0	0	0	0	0	0	0
ferry	30	4	0	-2	0	+1	0	0
floortile	0	0	0	0	0	0	0	0
miconic	30	23	0	-7	0	-1	0	+4
rovers	0	0	0	0	0	0	0	0
satellite	1	0	-29	-2	+1	0	-22	0
sokoban	0	0	0	0	0	0	0	0
spanner	25	0	+25	0	+21	0	-5	0
transport	9	0	0	0	+3	0	-5	0
all	121	27	+22	-11	+23	-3	-33	+4

Table 4.8: The first column lists the number of medium and hard problems solved using the $GOOSE_{gat}$ heuristic. The remaining columns show the difference in this number compared to baseline and previously introduced heuristics. For example, a value of +26 under h^{ff} means $GOOSE_{gat}$ solves 11 problems more than h^{ff} in that domain. The baseline h_{blind} was omitted because it does not solve any medium or hard problems.

Contrary to the overall performance results, $GOOSE_{gat}$ demonstrates a notable advantage on medium problems by outperforming h^{ff} by 33 instances, primarily due to its strong performance in the blocksworld and spanner domains, which compensates for its weaker results in the satellite domain. However, on hard problems, h^{ff} once again outperforms $GOOSE_{gat}$. $GOOSE_{gat}$ also surpasses $GOOSE_{standard}$ on medium problems and performs similarly on hard problems, making it a preferable choice over $GOOSE_{standard}$.

In comparison to $GOOSE_{mh-retrain}$, $GOOSE_{gat}$ performs worse on medium problems, solving significantly fewer instances. As a result, $GOOSE_{mh-retrain}$ remains the best-performing heuristic for medium problems, with $GOOSE_{gat}$ solving 33 fewer instances.

In terms of the number of expanded nodes and the cost of the plans, Figure 4.4 illustrates that in the miconic domain, $GOOSE_{gat}$ outperforms $GOOSE_{standard}$ and $GOOSE_{mh-retrain}$ by expanding fewer nodes and generating plans with lower costs. However, in other domains, $GOOSE_{standard}$ and $GOOSE_{mh-retrain}$ are preferable heuristics when considering the number of expanded nodes. Regarding plan costs, the performance of the heuristics across the other domains is comparable, with no significant differences observed. Overall, compared to $GOOSE_{standard}$, $GOOSE_{gat}$ can be justified as the preferred heuristic because it solves more problems and finds cheaper plans, even though it may require expanding more nodes in some domains. On the other hand, when compared to $GOOSE_{mh-retrain}$, despite $GOOSE_{gat}$ finding cheaper plans in the miconic domain, it is not considered the preferred heuristic due to its significantly lower success in solving medium-sized problems and its lack of advantage in the number of expanded nodes across all domains. The variances of $GOOSE_{gat}$, with ± 34.5 for expanded nodes and ± 1.55 for plan cost, are similar to those of $GOOSE_{standard}$ and $GOOSE_{mh-retrain}$. Therefore, they should not affect the validity of the comparisons made.

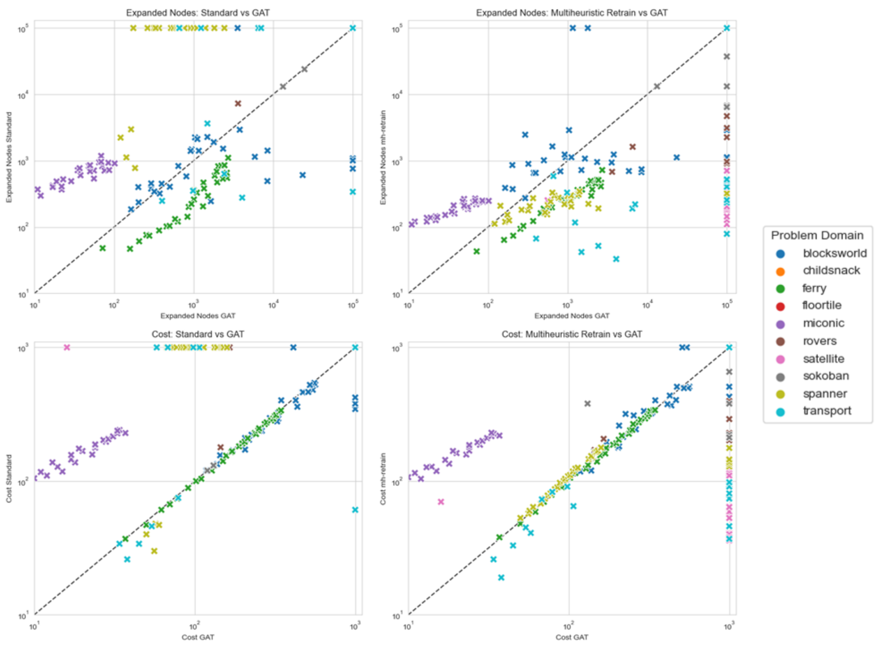


Figure 4.4: The number of expanded nodes and the plan cost for $GOOSE_{standard}$ compared to $GOOSE_{gat}$ (graphs on the left) and $GOOSE_{mh-retrain}$ compared to $GOOSE_{gat}$ (graphs on the right) of the medium problems. For problems unsolved by one planner, the corresponding metric is set to the axis limit. Points located in the top-left triangle favor $GOOSE_{gat}$.

Although attention-based models are popular in the literature for automated planning, the application of RGAT layers within the GOOSE framework on our benchmarking dataset does not provide a performance advantage compared to the previously presented heuristics. To understand the suboptimal performance of $GOOSE_{gat}$ on medium difficulty problems, we examined the inference time of the models. Table 4.9 presents the inference time needed by the model of each problem domain to process one graph instance of a medium problem through the RGNN models used in $GOOSE_{standard}$, $GOOSE_{retrain}$, $GOOSE_{mh}$ and $GOOSE_{mh-retrain}$ and the RGAT models used in $GOOSE_{gat}$. The data indicates that, except for the floortile domain, the RGAT model consistently has a slower inference time compared to the RGNN model. The fact that $GOOSE_{gat}$ cannot solve the problem in the floortile domain, despite having a better inference time, is likely related to the expressiveness limitations of the model. As discussed in the theory, RGAT models have the same level of expressiveness as RGNN models, and the floortile domain probably requires a more expressive model, making the inference time advantage negligible. In all other domains, the RGAT models demonstrate a faster inference time, with RGAT models taking approximately 1.7 times longer on average. This extended inference time is likely a reason for the suboptimal performance of $GOOSE_{gat}$ on medium and hard problems, as the number of evaluated states rises in the medium and hard problems and thus the inference time becomes more important because the heuristic is used more often.

Domain	**RGNN**	**RGAT**	**Difference**
blocksworld	0.0120	0.0285	+0.0165
childsnack	0.0512	0.0751	+0.0239
ferry	0.0128	0.0145	+0.0017
floortile	0.0610	0.0355	-0.0255
miconic	0.1298	0.2766	+0.1468
rovers	0.0175	0.0903	+0.0728
satellite	0.0467	0.1182	+0.0715
sokoban	0.0386	0.0416	+0.0030
spanner	0.0156	0.0306	+0.0150
transport	0.0758	0.0772	+0.0014
Average Time	0.0461	0.07881	+0.03271

Table 4.9: Comparison of the inference times of RGNN and RGAT models for a graph instance of a medium problem. All numbers are averages taken by passing the medium instance 100 times through each of the models.

Overall, we can conclude that $GOOSE_{gat}$ is not the optimal choice for our objective of enhancing performance on medium and hard problems. It is clearly outperformed by our previous approach, $GOOSE_{mh-retrain}$ on medium problems and on hard problems already from $GOOSE_{standard}$. Additionally, in terms of overall planner performance, $GOOSE_{gat}$ does not achieve better results than any of the heuristics we previously presented. The likely reason for the suboptimal performance of $GOOSE_{gat}$ is the longer inference time associated with the RGAT model compared to the RGNN models. However, it is important to note that $GOOSE_{gat}$ still outperforms h_{blind} in overall planner performance, which means it can still be considered an informative heuristic.

4.6 Relational Graph Attention with Retraining

In the previous section, we presented the results of replacing RGNN layers with RGAT layers in our models. As discussed in section 3.8, we also explored the combination of RGAT models with retraining to evaluate the effectiveness of attention mechanisms during fine-tuning. This approach will be referred to as $GOOSE_{gat-retrain}$. Similar to the previously tested retraining heuristics, retraining in this context is applied only to medium and hard problems, aiming to mitigate the domain drift from the easier training problems. For easy problems, retraining is not triggered, meaning the results of $GOOSE_{gat-retrain}$ on easy problems are expected to be the same as those achieved by $GOOSE_{gat}$.

Domain	$\mathbf{h}_{blind}$	$\mathbf{h}^{ff}$	$\mathbf{GOOSE}_{standard}$	$\mathbf{GOOSE}_{retrain}$	$\mathbf{GOOSE}_{mb}$	$\mathbf{GOOSE}_{mb-retrain}$	$\mathbf{GOOSE}_{gat}$	$\mathbf{GOOSE}_{gat-retrain}$
blocksworld	8	27	61	53	49	57	55	40
childsnack	9	25	13	13	18	18	20	20
ferry	11	66	63	64	63	64	64	64
floortile	3	12	1	1	9	9	1	1
miconic	30	90	84	83	81	79	83	83
rovers	15	34	28	41	36	43	31	32
satellite	12	62	23	32	46	53	31	32
sokoban	28	36	31	34	36	36	32	33
spanner	30	30	34	61	56	60	55	51
transport	9	39	35	43	48	46	39	40
variance	0	0	$\pm$ 0.9	$\pm$ 1.9	$\pm$ 0.8	$\pm$ 1.8	$\pm$ 0.9	$\pm$ 1.8
all	155	421	373	425	442	465	411	396

Table 4.10: Coverage of the FastDownward planner using $GOOSE_{gat-retrain}$ as a heuristic compared to the baseline heuristics, $GOOSE_{retrain}$, $GOOSE_{mb}$, $GOOSE_{mb-retrain}$ and $GOOSE_{gat}$. The top three heuristics in each row are highlighted with cell coloring intensity, with the best one in bold. The maximum number of problems that can be solved per domain is 90 (30 easy, 30 medium and 30 hard problems), so the maximum overall achievable coverage is 900.

Thus, any improvements are anticipated only in the medium and hard difficulty levels. Table 4.10 compares the coverage of the Fastdownward planner using $GOOSE_{gat-retrain}$ as a heuristic against previously presented heuristics and the baseline heuristics.We can observe that $GOOSE_{gat-retrain}$ manages to outperform h_{blind} so we can consider it an informative heuristic. It is also evident that, in terms of overall performance, retraining does not enhance the heuristics compared to $GOOSE_{gat}$; in fact, fewer problems are solved. This is primarily due to the blocksworld domain, where 15 fewer problems are solved with $GOOSE_{gat-retrain}$ than with $GOOSE_{gat}$. The retraining process appears to negatively impact the blocksworld domain, possibly causing some crucial information to be lost. In the other domains, $GOOSE_{gat-retrain}$ performs on par with $GOOSE_{gat}$, but does not surpass it, resulting in overall poorer performance compared to $GOOSE_{gat}$. The only heuristics $GOOSE_{gat-retrain}$ manages to outperform in terms of the overall performance are h_{blind} and $GOOSE_{standard}$.

Table 4.11 also highlights the results for medium and hard problems where we hoped to see improvements using $GOOSE_{gat-retrain}$. We compare $GOOSE_{gat-retrain}$ against the baseline heuristics, the best-performing heuristic on medium problems so far, $GOOSE_{mh-retrain}$, and $GOOSE_{gat}$, the version of the heuristic without retraining. The only heuristic outperformed by $GOOSE_{gat-retrain}$ on medium problems is $GOOSE_{standard}$; all other heuristics achieve similar or better performances, with the blocksworld domain again being the primary area of weakness for $GOOSE_{gat-retrain}$. On hard problems, the situation is similar, with $GOOSE_{gat-retrain}$ only slightly outperforming $GOOSE_{mh-retrain}$ by solving four additional problems, but it performs similarly or worse in other domains.

	GOOSE$_{gat-retrain}$		**h^{ff}**		**GOOSE$_{standard}$**		**GOOSE$_{mh-retrain}$**		**GOOSE$_{gat}$**	
Domain	**medium**	**hard**	**medium**	**hard**	**medium**	**hard**	**medium**	**hard**	**medium**	**hard**
blocksworld	11	0	+11	0	-17	-3	-16	0	-15	0
childsnack	0	0	0	0	0	0	0	0	0	0
ferry	30	4	0	-2	0	+1	0	0	0	0
floortile	0	0	0	0	0	0	0	0	0	0
miconic	30	23	0	-7	0	-1	0	+4	0	0
rovers	3	0	-2	0	+2	0	-10	0	0	0
satellite	2	0	-28	-2	+2	0	-21	0	+1	0
sokoban	3	0	-3	0	+1	0	-3	0	0	0
spanner	21	0	+21	0	+17	0	-9	0	-4	0
transport	10	0	+1	0	+4	0	-4	0	+1	0
all	110	27	0	-11	+9	-3	-63	+4	-17	0

Table 4.11: The first column lists the number of medium and hard problems solved using the $GOOSE_{gat-retrain}$ heuristic. The remaining columns show the difference in this number compared to baseline and previously introduced heuristics. For example, a value of +11 under h^{ff} means $GOOSE_{WL-GPR}$ solves 11 problems more than h^{ff} in that domain. The baseline h_{blind} was omitted because it does not solve any medium or hard problems.

The results presented suggest that retraining attention models is not an optimal approach in our use case. The heuristic $GOOSE_{gat-retrain}$ fails to outperform $GOOSE_{gat}$, the non-retrained version, and also exhibits worse performance compared to most of the other heuristics we previously introduced, as well as the baseline heuristics. Figure 4.5 underlines that $GOOSE_{gat-retrain}$ offers no significant performance advantage over $GOOSE_{gat}$. In terms of plan cost and the number of expanded nodes, $GOOSE_{gat-retrain}$ is the preferred heuristic for only a few problems. For the majority of problems, $GOOSE_{gat}$ either matches or outperforms $GOOSE_{gat-retrain}$ with respect to both expanded nodes and plan cost. Moreover, considering the additional time required to generate a separate training dataset and conduct the retraining process, the results of $GOOSE_{gat-retrain}$ do not justify the extra effort compared to other heuristics. The suboptimal performance with retraining might be linked to the increased complexity introduced by the attention mechanism. During retraining, attention weights may become less stable or overfit to specific patterns, which could lead to a decline in performance if the model overly focuses on relationships that do not generalize well. In conclusion, retraining the RGAT model is not the most effective strategy for enhancing the GOOSE framework, as other heuristics have demonstrated greater effectiveness.

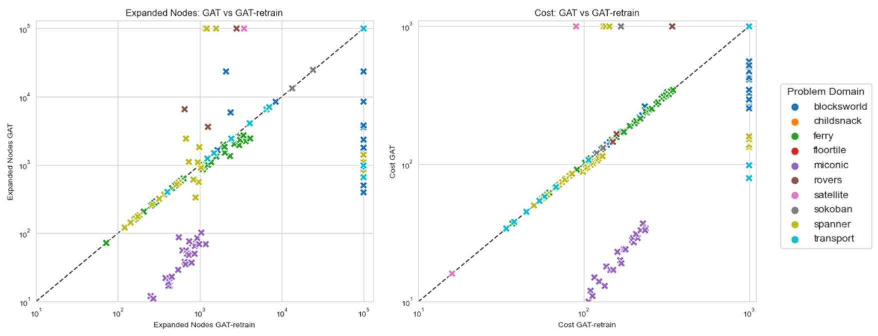


Figure 4.5: The number of expanded nodes and the plan cost for $GOOSE_{gat}$ compared to $GOOSE_{gat-retrain}$ of the medium problems. For problems unsolved by one planner, the corresponding metric is set to the axis limit. Points located in the top-left triangle favor $GOOSE_{gat-retrain}$.

4.7 Masked Attention for Graphs

This section presents the results obtained using the MAG model described in subsection 2.2.6, replacing the previously used GNN models for heuristic modeling in GOOSE. The experiments were conducted as detailed in section 3.9. The approach is referred to as $GOOSE_{MAG-node}$ or $GOOSE_{MAG-edge}$ depending on whether the MAG model is based on the node features or edge features.

Due to time constraints, we were unable to complete experiments for all difficulty levels and problem domains. While we trained models for all domains, we could not finish testing for all of them. Therefore, the following preliminary results are shown only for the easy difficulty level across five domains:

Blocksworld, Floortile, Miconic, Rovers, and Sokoban. The domains were selected based on specific characteristics. Miconic was chosen because in the previous experiments nearly all heuristics performed well in this domain, hoping that the same is true for the MAG models. Blocksworld and Rovers were included because GOOSE$_{standard}$ performs well, but h_{ff} does not, allowing us to evaluate the effectiveness of MAG models. Floortile and Sokoban were selected as GOOSE$_{standard}$ performs poorly in these domains due to limited expressiveness since these domains do not fall into the C2 subclass of first-order logic. This allows us to assess if MAG can enhance expressiveness. Due to time constraints, we used a 10-minute timeout for solving each problem, instead of the 20 minutes used in previous experiments, to expedite the testing process. Baseline heuristics include h_{blind} and GOOSE$_{standard}$, both performed with a 10-minute timeout per problem. We evaluated both MAG approaches: GOOSE$_{MAG-edge}$, which uses edge features and the edge adjacency matrix as a mask, and $GOOSE_{MAG-node}$, which uses node features and the node adjacency matrix as a mask. Table 4.12 shows the number of solved problems for the baseline heuristics, $GOOSE_{MAG-node}$, and $GOOSE_{MAG-edge}$ for the domains Blocksworld, Floortile, Miconic, Rovers, and Sokoban at the easy difficulty level.

Analyzing the results, we observe that the performance of $GOOSE_{MAG-edge}$ and $GOOSE_{MAG-node}$ is suboptimal, as they are outperformed by h_{blind} in all domains. This suggests that these models may mislead the planner's search process. The only domain where they manage to outperform $GOOSE_{standard}$ is Sokoban. Overall, $GOOSE_{MAG-node}$ and $GOOSE_{MAG-edge}$ solve only 38 and 39 problems, which is significantly fewer problems than the baseline models manage to solve. This indicates that the MAG approach does not work well as a heuristic. There is almost no difference in the number of solved problems between $GOOSE_{MAG-node}$ and $GOOSE_{MAG-edge}$, providing no clear indication of which approach is more suitable as a heuristic in GOOSE.

	baseline		**MAG**	
Domain	h_{blind}	**GOOSE**$_{standard}$	**GOOSE**$_{MAG-node}$	**GOOSE**$_{MAG-edge}$
blocksworld	8	30	2	2
floortile	2	0	0	0
miconic	30	30	18	19
rovers	15	26	5	5
sokoban	27	1	13	13
all	82	87	38	39

Table 4.12: Number of solved problems for GOOSE$_{MAG-node}$ and GOOSE$_{MAG-edge}$ compared to the baseline heuristics. All tests were conducted on problems of easy difficulty level with a timeout of 10 minutes per problem. In each domain 30 easy problems were tested.

To investigate the suboptimal performance of $GOOSE_{MAG}$, we will first examine the validation loss.

Table 4.13 presents a comparison of the validation loss for the $GOOSE_{MAG}$ models and the $GOOSE_{standard}$ models. Since training could be completed for all problem domains, we can compare the validation losses across all domains. For $GOOSE_{MAG}$, the loss for both the edge and node approaches is shown, as indicated by the "type" column.

Domain	**Type**	**GOOSE$_{standard}$**	**GOOSE$_{MAG}$**	**Difference**
blocksworld	Node	0.24	2.78	+2.54
	Edge	0.24	3.46	+3.22
ferry	Node	0.01	0.82	+0.81
	Edge	0.01	0.73	+0.72
floortile	Node	0.78	0.99	+0.21
	Edge	0.78	1.08	+0.30
miconic	Node	0.85	0.74	-0.11
	Edge	0.85	0.95	+0.10
rovers	Node	1.01	1.17	+0.16
	Edge	1.01	1.04	+0.03
satellite	Node	0.09	0.65	+0.56
	Edge	0.09	0.60	+0.51
sokoban	Node	47.41	36.33	-11.08
	Edge	47.41	27.24	-20.17
spanner	Node	1.63	2.19	+0.56
	Edge	1.63	1.96	+0.33
transport	Node	0.80	1.57	+0.77
	Edge	0.80	1.42	+0.62

Table 4.13: Validation loss of $GOOSE_{standard}$ and $GOOSE_{mag}$. The "type" column applies only to $GOOSE_{mag}$, indicating whether the loss is from a node or edge model. In $GOOSE_{standard}$, there is no such distinction, so the results are simply repeated for comparison.

In most domains, the loss is significantly higher than in $GOOSE_{standard}$, suggesting that the model does not effectively capture the heuristic function. One possible reason for this is the limited amount of data. As attention mechanisms add complexity to the model, and considering that the MAG models have more parameters than the RGNN models used in $GOOSE_{standard}$ they typically require a larger amount of training data to fit well. For example the $GOOSE_{standard}$ RGNN modle used for the blocksworld domain has 54721 trainable parameters compared to 1794529 trainable parameters of the $GOOSE_{mag-node}$ model used for the blocksworld domain. An exception to the higher validation loss is the Sokoban domain, where the validation loss, although still high compared to other domains, shows clear improvement with MAG in both the node and edge approach. This explains the results in Table 4.12, where the Sokoban domain is the only domain where the $GOOSE_{MAG}$ heuristics outperform the $GOOSE_{standard}$

heuristic. Additionally, in the Miconic domain, the MAG model results in a better validation loss. To gain further insights into why $GOOSE_{MAG}$ still performs significantly worse than $GOOSE_{standard}$ in this domain, the next step is to examine the inference time.

Table 4.14 presents the inference time of the $GOOSE_{MAG}$ models compared to the $GOOSE_{standard}$ models. It is evident that the inference time for all $GOOSE_{MAG}$ models is significantly higher than that of the RGNN models used in $GOOSE_{standard}$. On average, the inference time for the $GOOSE_{MAG}$ models is approximately 36 times longer than for the $GOOSE_{standard}$ models.

Domain	**Type**	**GOOSE**$_{standard}$	**GOOSE**$_{MAG}$	**Difference**
blocksworld	Node	0.012	1.7700	1.7580
	Edge	0.012	1.8581	1.8461
ferry	Node	0.0128	1.7863	1.7735
	Edge	0.0128	1.7389	1.7261
floortile	Node	0.061	1.7328	1.6718
	Edge	0.061	1.7369	1.6759
miconic	Node	0.1298	1.7255	1.5957
	Edge	0.1298	1.7132	1.5834
rovers	Node	0.0175	1.7137	1.6962
	Edge	0.0175	1.8539	1.8364
satellite	Node	0.0467	1.7207	1.6740
	Edge	0.0467	1.7443	1.6976
sokoban	Node	0.0386	1.7083	1.6697
	Edge	0.0386	1.7403	1.7017
spanner	Node	0.0156	1.7031	1.6875
	Edge	0.0156	1.7536	1.7380
transport	Node	0.0758	1.7042	1.6284
	Edge	0.0758	1.7349	1.6591
Average		0.0455	1.7466	1.7011

Table 4.14: Inference time of $GOOSE_{standard}$ and $GOOSE_{mag}$. The "type" column applies only to $GOOSE_{mag}$, indicating whether the time is from a node or edge model. In $GOOSE_{standard}$, ther e is no such distinction, so the results are simply repeated for comparison.

Given that the heuristic may be evaluated millions of times by the Fast Downward planner when solving a problem, this substantially higher inference time can significantly slow down the search process. This is a key explanation for the poor performance of the $GOOSE_{MAG}$ models and is probabaly the reason why despite a better validation loss in the miconic domain the $GOOSE_{MAG}$ heuristic still performs worse then the $GOOSE_{standard}$ heuristic. The primary reason for the extended inference time is the generation of the node and edge feature matrices, as well as the creation of the masks needed for the attention mechanism. Once these elements are generated, the forward pass itself does not take much longer than the forward pass through the RGNN models of $GOOSE_{standard}$. Therefore, if we can find a way to optimize the generation of the feature matrices and the masks within the GOOSE framework, performance may improve.

Overall, we can conclude that the current $GOOSE_{MAG-node}$ and $GOOSE_{MAG-edge}$ approaches lead to suboptimal performance on easy problems and do not outperform the baseline heuristics. In the way they are currently implemented, they are not effective heuristics and are not viable alternatives to our previously presented approaches. We could not discern a clear difference in the performance of the $GOOSE_{MAG-node}$ and $GOOSE_{MAG-edge}$ models, providing no indication of which approach might work better within the GOOSE framework. The suboptimal performance of the $GOOSE_{MAG}$ heuristics is due to two main reasons. Firstly, the MAG models do not fit the data as well as the RGNN models of $GOOSE_{standard}$, as indicated by their higher validation losses. This is likely because the MAG models have significantly more parameters and would require more data to achieve a better fit. Secondly, the generation of the feature matrix and masks results in much higher inference times for the $GOOSE_{MAG}$ models compared to the $GOOSE_{standard}$ models, which significantly slows down the planner's search process. Before further exploring the $GOOSE_{MAG}$ approaches, either a more optimized implementation of the matrix generation needs to be integrated into the GOOSE framework, or more training data needs to be available. Ideally, both improvements should be made before continuing exploring the approach. With the current setup, further exploration is likely not worthwhile.

4.8 Weisfeiler Leman Model

For the final comparison, we will evaluate the performance of the previously introduced heuristics against one of the latest enhancements to the GOOSE framework: the WL-kernel models. Specifically, we will focus on the model utilizing Gaussian Process Regression (GPR) with a dot product kernel, as this approach demonstrated the best performance in a domain-dependent setting according to the most recent GOOSE study, as outlined in subsubsection 2.3.3.2. This approach will be referred to as $GOOSE_{WL-GPR}$.

Table 4.15 presents the overall coverage of $GOOSE_{WL-GPR}$ in comparison to the baseline heuristics and the heuristics introduced in this thesis. Since $GOOSE_{WL-GPR}$ is a deterministic model, no variance is reported. In terms of the overall performance $GOOSE_{WL-GPR}$ manages to outperform all baseline heuristics as well as heuristics presented so far in this thesis. The closest competitor in terms of overall performance is $GOOSE_{mh-retrain}$, which still solves approximately 20 fewer problems. However, some of our previously introduced heuristics surpass $GOOSE_{WL-GPR}$ in specific domains, including transport, rovers, and sokoban. Additionally, h^{ff} achieves better performance in the satellite and floortile domains. In the remaining five domains, $GOOSE_{WL-GPR}$ emerges as the best-performing heuristic, particularly excelling in the ferry and spanner domains. The superior performance of $GOOSE_{WL-GPR}$ can be attributed to its ability to solve a significantly higher number of hard problems. As shown in Table 4.16, $GOOSE_{WL-GPR}$ outperforms other heuristics in the number of solved medium and hard problems, solving a total of 68 hard problems, 37 more than the next best-performing heuristic $GOOSE_{retrain}$. Additionally, $GOOSE_{WL-GPR}$ delivers strong results on medium problems, being the only heuristic capable of solving any medium childsnack problems. However, in terms of overall performance on medium problems, it is outperformed by $GOOSE_{mb-retrain}$, particularly in the transport domain.

Domain	**h**$_b$*lind*	**h***ff*	**GOOSE**$_{standard}$	**GOOSE**$_{retrain}$	**GOOSE**$_{mb}$	**GOOSE**$_{mb-retrain}$	**GOOSE**$_{gat}$	**GOOSE**$_{gat-retrain}$	**GOOSE**$_{WL-GPR}$
blocksworld	8	27	61	53	49	57	55	40	70
childsnack	9	25	13	13	18	18	20	20	29
ferry	11	66	63	64	63	64	64	64	75
floortile	3	12	1	1	9	9	1	1	2
miconic	30	90	84	83	81	79	83	83	90
rovers	15	34	28	41	36	43	31	32	36
satellite	12	62	23	32	46	53	31	32	53
sokoban	28	36	31	34	36	36	32	33	33
spanner	30	30	34	61	56	60	55	51	71
transport	9	39	35	43	48	46	39	40	29
variance	0	0	± 0.9	± 1.9	± 0.8	± 1.8	± 0.9	± 1.9	0
all	155	421	373	425	442	465	411	396	488

Table 4.15: Coverage of the FastDownward planner using $GOOSE_{gat-retrain}$ as a heuristic compared to the baseline heuristics, $GOOSE_{retrain}$, $GOOSE_{mb}$, $GOOSE_{mb-retrain}$ and $GOOSE_{gat}$. The top three heuristics in each row are highlighted with cell coloring intensity, with the best one in bold. The maximum number of problems that can be solved per domain is 90 (30 easy, 30 medium and 30 hard problems), so the maximum overall achievable coverage is 900.

The likely reason for the superior performance of $GOOSE_{WL-GPR}$ is its very fast evaluation time. Kernel models are quick to train and evaluate, which provides a significant advantage in more difficult problems. Typically, as problem difficulty increases, the planner evaluates the heuristic more frequently, making the speed of evaluation crucial. In terms of expressiveness, as discussed in subsection 2.2.5, the WL kernel models share the same expressiveness as GNNs. This might explain why $GOOSE_{WL-GPR}$ does not perform well in the floortile domain, where a more expressive model is likely required.

Overall, the performance of $GOOSE_{WL-GPR}$ compared to other GNN-based heuristics tested in this research appears very promising. Future work could explore approaches such as retraining or multi-heuristic search using kernel models instead of GNNs to determine whether these methods can further enhance the performance of kernel based heuristics as they have proven to work for GNN based heuristics.

	GOOSE_{WL-GPR}		h^{ff}		$\text{GOOSE}_{standard}$		$\text{GOOSE}_{retrain}$		$\text{GOOSE}_{mh-retrain}$	
Domain	**medium**	**hard**	**medium**	**hard**	**medium**	**hard**	**medium**	**hard**	**medium**	**hard**
blocksworld	28	12	+28	+12	0	+9	+8	+9	+1	+12
childsnack	5	0	+5	0	+5	0	+5	0	+5	0
ferry	30	15	0	+9	0	+27	0	+11	0	+11
floortile	0	0	0	0	0	0	0	0	0	0
miconic	30	30	0	0	0	+6	0	+7	0	+11
rovers	6	0	+1	0	+5	0	+5	0	-7	0
satellite	23	0	-7	-2	+23	0	+14	0	0	0
sokoban	4	0	-2	0	+2	0	-1	0	-2	0
spanner	30	11	+30	+11	+26	+11	0	+10	0	+11
transport	0	0	-9	0	-6	0	-13	0	-14	0
all	156	68	+46	+30	+55	+53	+18	+37	-17	+45

Table 4.16: The first column $GOOSE_{WL-GPR}$ lists the number of medium and hard problems solved using the $GOOSE_{WL-GPR}$ heuristic. The remaining columns show the difference in this number compared to baseline and previously introduced heuristics. For example, a value of +28 under h^{ff} means $GOOSE_{WL-GPR}$ solves 28 problems more than h^{ff} in that domain. The baseline h_{blind} was omitted because it does not solve any medium or hard problems.

5 Conclusion

This work explored various approaches for enhancing the GOOSE framework to solve problems in automated planning, particularly focusing on mitigating domain drift that occurs when domain-dependent heuristics, trained on simpler problems, are applied to more complex ones. Therefore we introduced six new heuristics: $GOOSE_{retrain}$, $GOOSE_{mh}$, $GOOSE_{mh-retrain}$, $GOOSE_{gat}$, $GOOSE_{gat-retrain}$, and $GOOSE_{mag}$. Most of these were further developments of previously implemented heuristics in GOOSE based on RGNN models. We conducted a comparative analysis of these heuristics against each other and three baseline heuristics—h_{blind}, h^{ff} and $GOOSE_{standard}$ —evaluating them based on coverage of the Fast-Downward planner, the number of expanded nodes by the planner, and the cost of the resulting plans when using the heuristics. In addition we provided a comparison to the $GOOSE_{WL-GPR}$ heuristic one of the most recently published heuristics in the GOOSE framework. All our proposed heuristics, except $GOOSE_{mag}$, outperformed the previously used RGNN model ($GOOSE_{standard}$). However, not all of them managed to surpass all the baseline heuristics as they were outperformed by h^{ff} in terms of the overall performance. The kernel-based heuristic, $GOOSE_{WL-GPR}$, remained the most effective overall, although $GOOSE_{mh-retrain}$ closely followed, particularly excelling in medium-difficulty problems. This suggests that retraining and multi-heuristic search are promising approaches for mitigating domain drift between training on easy and testing on medium-difficulty problems. Our introduced heuristics did not achieve significant improvements on the hard problems. The limited number of retraining problems hindered the effectiveness of retraining, and multi-heuristic search slowed down the search process, which is critical in more complex scenarios. The heuristics $GOOSE_{gat}$ and $GOOSE_{gat-retrain}$, which incorporated attention mechanisms to the framework, did not perform that well due to the longer inference times and increased model complexity that results in more trainable parameters which would benefit from more training data. Similarly, the novel approach in $GOOSE_{mag}$, which provides more expressiveness because it does not rely on message passing, faced challenges due to high inference times and likely also requires more training data, as suggested by the relatively high validation losses.

In summary, we developed multiple heuristics that effectively reduced domain drift from easy to medium problems, with $GOOSE_{mh-retrain}$ showing the best performance. The attention-based approaches were hindered by limited training data and increased complexity, leading to fewer solved problems. On harder problems, we observed less improvement, with the kernel model $GOOSE_{WL-GPR}$ maintaining its status as the best-performing heuristic, likely due to its fast evaluation time.

5.1 Future Work

This thesis focused on domain-dependent model training, where models were trained and tested within the same problem domain. However, previous GOOSE studies have demonstrated the feasibility of learning domain-independent heuristics. A direction for future work could involve replicating the experiments described in this thesis using a domain-independent approach. This would entail training a model on data from multiple domains and testing it across all those domains. Such an approach could provide valuable insights into whether multiheuristic search with retraining still yields the best results in a broader context or if models with attention mechanisms exhibit better performance in a more generalized setting.

Exploring the integration of the $GOOSE_{WL-GPR}$ heuristic, which has shown to outperform multiple of our heuristics in the experiments, into the multiheuristic search framework could be another promising direction for future research. This could involve incorporating it as a third heuristic or replacing one of the current heuristics. Evaluating this integration could reveal whether it leads to further performance improvements and potentially refine the multiheuristic search strategy for better optimization results. The retraining approach could as well be applied to the $GOOSE_{WL-GPR}$ models to evaluate if retraining also helps the kernel models to overcome the domain drift. Given that these models are very fast to train, the retraining process would not be particularly time-consuming. Additionally, a deeper exploration of the MAG approach could be valuable. Our findings indicated that one of the primary drawbacks of the MAG approach is the time required to generate feature and adjacency matrices. Optimizing this process could enhance the overall effectiveness of the MAG approach. Investigating more efficient methods for generating these matrices might lead to better performance outcomes. This would be especially interesting in domains such as floortile, where GNNs face limitations due to expressiveness constraints. By addressing these computational challenges, the MAG approach could yield better results and expand its applicability across a wider range of problem domains.

In conclusion, while this thesis has laid a strong foundation in domain-dependent heuristic search, there remains a wealth of opportunities for expanding and enhancing the current methodologies. By exploring domain-independent approaches, optimizing the MAG framework, and integrating kernel methods to the multiheuristic search future research can further push the boundaries of what is achievable with the GOOSE framework.

References

[1] M. Helmert, “The Fast Downward Planning System,” *Journal of Artificial Intelligence Research*, vol. 26, pp. 191–246, Jul. 2006, arXiv:1109.6051 [cs]. [Online]. Available: http://arxiv.org/abs/1109.6051

[2] B. Khemani, S. Patil, K. Kotecha, and S. Tanwar, “A review of graph neural networks: concepts, architectures, techniques, challenges, datasets, applications, and future directions,” *Journal of Big Data*, vol. 11, no. 1, p. 18, Jan. 2024. [Online]. Available: https://doi.org/10.1186/s40537-023-00876-4

[3] M. Schlichtkrull, T. N. Kipf, P. Bloem, R. v. d. Berg, I. Titov, and M. Welling, “Modeling Relational Data with Graph Convolutional Networks,” Oct. 2017, arXiv:1703.06103 [cs, stat]. [Online]. Available: http://arxiv.org/abs/1703.06103

[4] B. Zhang, C. Fan, S. Liu, K. Huang, X. Zhao, J. Huang, and Z. Liu, “The Expressive Power of Graph Neural Networks: A Survey,” Aug. 2023, arXiv:2308.08235 [cs]. [Online]. Available: http://arxiv.org/abs/2308.08235

[5] D. Buterez, J. P. Janet, D. Oglic, and P. Lio, “Masked Attention is All You Need for Graphs,” Feb. 2024, arXiv:2402.10793 [cs]. [Online]. Available: http://arxiv.org/abs/2402.10793

[6] D. Z. Chen, S. Thiébaux, and F. Trevizan, “Learning Domain-Independent Heuristics for Grounded and Lifted Planning,” Dec. 2023, arXiv:2312.11143 [cs]. [Online]. Available: http://arxiv.org/abs/2312.11143

[7] M. Helmert and S. Richter, “Doc/Evaluator - Fast Downward Homepage.” [Online]. Available: https://www.fast-downward.org/Doc/Evaluator

[8] M. Helmert, S. Richter, and J. Seipp, “Doc/SearchAlgorithm - Fast Downward Homepage.” [Online]. Available: https://www.fast-downward.org/Doc/SearchAlgorithm

[9] S. Chen and K. Mao, “Explicit and implicit knowledge-enhanced model for event causality identification,” *Expert Systems with Applications*, vol. 238, p. 122039, Mar. 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417423025411

[10] H. Geffner and B. Bonet, “Classical Planning: Full Information and Deterministic Actions,” in *A Concise Introduction to Models and Methods for Automated Planning*, H. Geffner and

B. Bonet, Eds. Cham: Springer International Publishing, 2013, pp. 15–36. [Online]. Available: https://doi.org/10.1007/978-3-031-01564-9_2

[11] I. Partalas, D. Vrakas, and I. Vlahavas, "Reinforcement Learning and Automated Planning: A Survey," *Artificial Intelligence for Advanced Problem Solving Techniques*, Jan. 2012.

[12] M. Ghallab, D. Nau, and P. Traverso, *Automated Planning and Acting*, 1st ed. Cambridge University Press, Jul. 2016. [Online]. Available: https://www.cambridge.org/core/product/identifier/9781139583923/type/book

[13] S. Richter and M. Westphal, "The LAMA Planner: Guiding Cost-Based Anytime Planning with Landmarks," *Journal of Artificial Intelligence Research*, vol. 39, pp. 127–177, Sep. 2010, arXiv:1401.3839 [cs]. [Online]. Available: http://arxiv.org/abs/1401.3839

[14] J. Hoffmann, "The Metric-FF Planning System: Translating "Ignoring Delete Lists" to Numeric State Variables," *Journal of Artificial Intelligence Research*, vol. 20, pp. 291–341, Dec. 2003, arXiv:1106.5271 [cs]. [Online]. Available: http://arxiv.org/abs/1106.5271

[15] S. Toyer, F. Trevizan, S. Thiébaux, and L. Xie, "Action Schema Networks: Generalised Policies with Deep Learning," Dec. 2017, arXiv:1709.04271 [cs]. [Online]. Available: http://arxiv.org/abs/1709.04271

[16] W. Shen, F. Trevizan, S. Toyer, S. Thiebaux, and L. Xie, "Guiding Search with Generalized Policies for Probabilistic Planning," *Proceedings of the International Symposium on Combinatorial Search*, vol. 10, no. 1, pp. 97–105, 2019, number: 1. [Online]. Available: https://ojs.aaai.org/index.php/SOCS/article/view/18507

[17] E. Groshev, M. Goldstein, A. Tamar, S. Srivastava, and P. Abbeel, "Learning Generalized Reactive Policies using Deep Neural Networks," Jul. 2018, arXiv:1708.07280 [cs]. [Online]. Available: http://arxiv.org/abs/1708.07280

[18] D. Z. Chen, F. Trevizan, and S. Thiébaux, "Return to Tradition: Learning Reliable Heuristics with Classical Machine Learning," Mar. 2024, arXiv:2403.16508 [cs]. [Online]. Available: http://arxiv.org/abs/2403.16508

[19] J. Seipp and J. Segovia-Aguas, "International Planning Competition 2023," 2023. [Online]. Available: https://ipc2023-learning.github.io/

[20] D. McDermott, M. Ghallab, A. Howe, C. A. Knoblock, A. Ram, M. Veloso, D. S. Weld, and D. Wilkins, "PDDL-the planning domain definition language," 1998. [Online]. Available: https://www.semanticscholar.

org / paper / PDDL-the-planning-domain-definition-language-McDermott-Ghallab / d82c6b8081343b2eae63d45feefe630233ad60e1

[21] D. Busbridge, D. Sherburn, P. Cavallo, and N. Y. Hammerla, "Relational Graph Attention Networks," Apr. 2019, arXiv:1904.05811 [cs, stat]. [Online]. Available: http://arxiv.org/abs/1904.05811

[22] R. E. Fikes and N. J. Nilsson, "Strips: A new approach to the application of theorem proving to problem solving," *Artificial Intelligence*, vol. 2, no. 3, pp. 189–208, Dec. 1971. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0004370271900105

[23] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, Jan. 2016, publisher: Nature Publishing Group. [Online]. Available: https://www.nature.com/articles/nature16961

[24] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis, "Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm," Dec. 2017, arXiv:1712.01815 [cs]. [Online]. Available: http://arxiv.org/abs/1712.01815

[25] H. Geffner and B. Bonet, *A Concise Introduction to Models and Methods for Automated Planning*, ser. Synthesis Lectures on Artificial Intelligence and Machine Learning. Cham: Springer International Publishing, 2013. [Online]. Available: https://link.springer.com/10.1007/978-3-031-01564-9

[26] M. Helmert, "Concise finite-domain representations for PDDL planning tasks," *Artificial Intelligence*, vol. 173, no. 5, pp. 503–535, Apr. 2009. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0004370208001926

[27] P. Lauer, A. Torralba, D. Fišer, D. Höller, J. Wichlacz, and J. Hoffmann, "Polynomial-Time in PDDL Input Size: Making the Delete Relaxation Feasible for Lifted Planning," in *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence*. Montreal, Canada: International Joint Conferences on Artificial Intelligence Organization, Aug. 2021, pp. 4119–4126. [Online]. Available: https://www.ijcai.org/proceedings/2021/567

[28] B. Bonet and H. Geffner, "Planning as heuristic search," *Artificial Intelligence*, vol. 129, no. 1, pp. 5–33, Jun. 2001. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0004370201001084

[29] J. Hoffmann and B. Nebel, "The FF Planning System: Fast Plan Generation Through Heuristic Search," *Journal of Artificial Intelligence Research*, vol. 14, pp. 253–302, May 2001, arXiv:1106.0675 [cs]. [Online]. Available: http://arxiv.org/abs/1106.0675

[30] M. Katz, S. Sohrabi, and H. Samulowitz, "Delfi: Online Planner Selection for Cost-Optimal Planning," 2018. [Online]. Available: https://www.semanticscholar.org/paper/Del%EF%AC%81%3A-Online-Planner-Selection-for-Cost-Optimal-Katz-Sohrabi/43d2dd86ee01fb14536d86c6bfac3bc3dada270f

[31] S. Sievers, "Merge-and-Shrink Heuristics for Classical Planning: Efficient Implementation and Partial Abstractions," *Proceedings of the International Symposium on Combinatorial Search*, vol. 9, no. 1, pp. 90–98, 2018, number: 1. [Online]. Available: https://ojs.aaai.org/index.php/SOCS/article/view/18450

[32] J. Seipp, T. Keller, and M. Helmert, "Saturated Cost Partitioning for Optimal Classical Planning," *Journal of Artificial Intelligence Research*, vol. 67, pp. 129–167, Jan. 2020. [Online]. Available: https://www.jair.org/index.php/jair/article/view/11673

[33] A. I. Coles and A. J. Smith, "Marvin: A Heuristic Search Planner with Online Macro-Action Learning," *Journal of Artificial Intelligence Research*, vol. 28, pp. 119–156, Feb. 2007, arXiv:1110.2736 [cs]. [Online]. Available: http://arxiv.org/abs/1110.2736

[34] W. Ju, Z. Liu, Y. Qin, B. Feng, C. Wang, Z. Guo, X. Luo, and M. Zhang, "Few-shot Molecular Property Prediction via Hierarchically Structured Learning on Relation Graphs," *Neural Networks*, vol. 163, pp. 122–131, Jun. 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0893608023001685

[35] Y. Xu, L. Zhu, J. Li, F. Li, and H. Shen, "Temporal Social Graph Network Hashing for Efficient Recommendation," *IEEE Transactions on Knowledge and Data Engineering*, vol. PP, pp. 1–14, Jan. 2024.

[36] R. Zhu, K. Zhao, H. Yang, W. Lin, C. Zhou, B. Ai, Y. Li, and J. Zhou, "AliGraph: A Comprehensive Graph Neural Network Platform," Feb. 2019, arXiv:1902.08730 [cs]. [Online]. Available: http://arxiv.org/abs/1902.08730

[37] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, "A Comprehensive Survey on Graph Neural Networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 1, pp. 4–24, Aug. 2019, arXiv:1901.00596 [cs, stat]. [Online]. Available: http://arxiv.org/abs/1901.00596

[38] W. Ju, S. Yi, Y. Wang, Z. Xiao, Z. Mao, H. Li, Y. Gu, Y. Qin, N. Yin, S. Wang, X. Liu, X. Luo, P. S. Yu, and M. Zhang, "A Survey of Graph Neural Networks in Real world: Imbalance, Noise, Privacy and OOD Challenges," Mar. 2024, arXiv:2403.04468 [cs]. [Online]. Available: http://arxiv.org/abs/2403.04468

[39] M. Gori, G. Monfardini, and F. Scarselli, "A new model for learning in graph domains," in *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, vol. 2. Montreal, Que., Canada: IEEE, 2005, pp. 729–734. [Online]. Available: http://ieeexplore.ieee.org/document/1555942/

[40] F. Scarselli, M. Gori, Ah Chung Tsoi, M. Hagenbuchner, and G. Monfardini, "The Graph Neural Network Model," *IEEE Transactions on Neural Networks*, vol. 20, no. 1, pp. 61–80, Jan. 2009. [Online]. Available: http://ieeexplore.ieee.org/document/4700287/

[41] A. Sandryhaila and J. M. F. Moura, "Discrete Signal Processing on Graphs," *IEEE Transactions on Signal Processing*, vol. 61, no. 7, pp. 1644–1656, Apr. 2013, arXiv:1210.4752 [physics]. [Online]. Available: http://arxiv.org/abs/1210.4752

[42] D. I. Shuman, S. K. Narang, P. Frossard, A. Ortega, and P. Vandergheynst, "The Emerging Field of Signal Processing on Graphs: Extending High-Dimensional Data Analysis to Networks and Other Irregular Domains," *IEEE Signal Processing Magazine*, vol. 30, no. 3, pp. 83–98, May 2013, arXiv:1211.0053 [cs]. [Online]. Available: http://arxiv.org/abs/1211.0053

[43] T. N. Kipf and M. Welling, "Variational Graph Auto-Encoders," Nov. 2016, arXiv:1611.07308 [cs, stat]. [Online]. Available: http://arxiv.org/abs/1611.07308

[44] B. Yu, H. Yin, and Z. Zhu, "Spatio-Temporal Graph Convolutional Networks: A Deep Learning Framework for Traffic Forecasting," in *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*, Jul. 2018, pp. 3634–3640, arXiv:1709.04875 [cs, stat]. [Online]. Available: http://arxiv.org/abs/1709.04875

[45] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph Attention Networks," Feb. 2018, arXiv:1710.10903 [cs, stat]. [Online]. Available: http://arxiv.org/abs/1710.10903

[46] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*, ser. Adaptive computation and machine learning. Cambridge, Massachusetts: The MIT Press, 2016. [Online]. Available: http://www.deeplearningbook.org

[47] A. Micheli, "Neural Network for Graphs: A Contextual Constructive Approach," *IEEE Transactions on Neural Networks*, vol. 20, no. 3, pp. 498–511, Mar. 2009. [Online]. Available: http://ieeexplore.ieee.org/document/4773279/

[48] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural Message Passing for Quantum Chemistry," Jun. 2017, arXiv:1704.01212 [cs]. [Online]. Available: http://arxiv.org/abs/1704.01212

[49] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, C. Gulcehre, F. Song, A. Ballard, J. Gilmer, G. Dahl, A. Vaswani, K. Allen, C. Nash, V. Langston, C. Dyer, N. Heess, D. Wierstra, P. Kohli, M. Botvinick, O. Vinyals, Y. Li, and R. Pascanu, "Relational inductive biases, deep learning, and graph networks," Oct. 2018, arXiv:1806.01261 [cs, stat]. [Online]. Available: http://arxiv.org/abs/1806.01261

[50] I. Bica, H. Andrés-Terré, A. Cvejic, and P. Liò, "Unsupervised generative and graph representation learning for modelling cell differentiation," *Scientific Reports*, vol. 10, no. 1, p. 9790, Jun. 2020, publisher: Nature Publishing Group. [Online]. Available: https://www.nature.com/articles/s41598-020-66166-8

[51] L. Wang, Z.-H. You, Y.-M. Li, K. Zheng, and Y.-A. Huang, "GCNCDA: A new method for predicting circRNA-disease associations based on Graph Convolutional Network Algorithm," *PLoS Computational Biology*, vol. 16, no. 5, p. e1007568, May 2020. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7266350/

[52] N. Karalias and A. Loukas, "Erdos Goes Neural: an Unsupervised Learning Framework for Combinatorial Optimization on Graphs," Mar. 2021, arXiv:2006.10643 [cs, stat]. [Online]. Available: http://arxiv.org/abs/2006.10643

[53] C. K. Joshi, T. Laurent, and X. Bresson, "An Efficient Graph Convolutional Network Technique for the Travelling Salesman Problem," Oct. 2019, arXiv:1906.01227 [cs, stat]. [Online]. Available: http://arxiv.org/abs/1906.01227

[54] Z. Li, Q. Chen, and V. Koltun, "Combinatorial Optimization with Graph Convolutional Networks and Guided Tree Search," Oct. 2018, arXiv:1810.10659 [cs, stat]. [Online]. Available: http://arxiv.org/abs/1810.10659

[55] O. G. Sierra, A. A. Galaz, M. O. Martín, J. A. Rodríguez, and A. A. Barriuso, "Temporal Relation Prediction from Electronic Health Records Using Graph Neural Networks and Transformers Embeddings," in *Artificial Intelligence for Healthy Longevity*, A. Moskalev, I. Stambler, and

A. Zhavoronkov, Eds. Cham: Springer International Publishing, 2023, pp. 143–152. [Online]. Available: https://doi.org/10.1007/978-3-031-35176-1_7

[56] B. Zhu, Y. Cai, and H. Ren, "Graph neural topic model with commonsense knowledge," *Information Processing & Management*, vol. 60, no. 2, p. 103215, Mar. 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0306457322003168

[57] M.-H. Guo, T.-X. Xu, J.-J. Liu, Z.-N. Liu, P.-T. Jiang, T.-J. Mu, S.-H. Zhang, R. R. Martin, M.-M. Cheng, and S.-M. Hu, "Attention Mechanisms in Computer Vision: A Survey," *Computational Visual Media*, vol. 8, no. 3, pp. 331–368, Sep. 2022, arXiv:2111.07624 [cs]. [Online]. Available: http://arxiv.org/abs/2111.07624

[58] C. Sun, C. Li, X. Lin, T. Zheng, F. Meng, X. Rui, and Z. Wang, "Attention-based graph neural networks: a survey," *Artificial Intelligence Review*, vol. 56, no. 2, pp. 2263–2310, Nov. 2023. [Online]. Available: https://doi.org/10.1007/s10462-023-10577-2

[59] S. Chaudhari, V. Mithal, G. Polatkan, and R. Ramanath, "An Attentive Survey of Attention Models," Jul. 2021, arXiv:1904.02874 [cs, stat]. [Online]. Available: http://arxiv.org/abs/1904.02874

[60] J. Gasteiger, S. Weißenberger, and S. Günnemann, "Diffusion Improves Graph Learning," Apr. 2022, arXiv:1911.05485 [cs, stat]. [Online]. Available: http://arxiv.org/abs/1911.05485

[61] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention Is All You Need," Aug. 2023, arXiv:1706.03762 [cs]. [Online]. Available: http://arxiv.org/abs/1706.03762

[62] A. Mayr, G. Klambauer, T. Unterthiner, and S. Hochreiter, "DeepTox: Toxicity Prediction using Deep Learning," *Frontiers in Environmental Science*, vol. 3, Feb. 2016, publisher: Frontiers. [Online]. Available: https://www.frontiersin.org/journals/environmental-science/articles/10.3389/fenvs.2015.00080/full

[63] L. Wu, P. Cui, J. Pei, and L. Zhao, Eds., *Graph Neural Networks: Foundations, Frontiers, and Applications*. Singapore: Springer Nature, 2022. [Online]. Available: https://link.springer.com/10.1007/978-981-16-6054-2

[64] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of Control, Signals and Systems*, vol. 2, no. 4, pp. 303–314, Dec. 1989. [Online]. Available: https://doi.org/10.1007/BF02551274

[65] K. Oono and T. Suzuki, “Graph Neural Networks Exponentially Lose Expressive Power for Node Classification,” Jan. 2021, arXiv:1905.10947 [cs, stat]. [Online]. Available: http://arxiv.org/abs/1905.10947

[66] J. Köbler, U. Schöning, and J. Torán, *The Graph Isomorphism Problem.* Boston, MA: Birkhäuser, 1993. [Online]. Available: http://link.springer.com/10.1007/978-1-4612-0333-9

[67] B. Weisfeiler and A. Leman, “The reduction of a graph to canonical form and the algebra which appears therein,” *nti, Series*, no. 2, pp. 12–16, 1968.

[68] L. Babai and L. Kucera, “Canonical labelling of graphs in linear average time,” in *20th Annual Symposium on Foundations of Computer Science (sfcs 1979)*, Oct. 1979, pp. 39–46, iSSN: 0272-5428. [Online]. Available: https://ieeexplore.ieee.org/document/4567999

[69] L. Babai, P. Erdo˝s, and S. M. Selkow, “Random Graph Isomorphism,” *SIAM Journal on Computing*, vol. 9, no. 3, pp. 628–635, Aug. 1980, publisher: Society for Industrial and Applied Mathematics. [Online]. Available: https://epubs.siam.org/doi/abs/10.1137/0209047

[70] C. Morris, M. Ritzert, M. Fey, W. L. Hamilton, J. E. Lenssen, G. Rattan, and M. Grohe, “Weisfeiler and Leman Go Neural: Higher-order Graph Neural Networks,” Nov. 2021, arXiv:1810.02244 [cs, stat]. [Online]. Available: http://arxiv.org/abs/1810.02244

[71] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, “How Powerful are Graph Neural Networks?” Feb. 2019, arXiv:1810.00826 [cs, stat]. [Online]. Available: http://arxiv.org/abs/1810.00826

[72] A. Fern, R. Givan, and S. Yoon, “Approximate Policy Iteration with a Policy Language Bias: Solving Relational Markov Decision Processes,” *Journal of Artificial Intelligence Research*, vol. 25, pp. 75–118, Jan. 2006, arXiv:1109.2156 [cs]. [Online]. Available: http://arxiv.org/abs/1109.2156

[73] M. Martín and H. Geffner, “Learning Generalized Policies from Planning Examples Using Concept Languages,” *Applied Intelligence*, vol. 20, no. 1, pp. 9–19, Jan. 2004. [Online]. Available: https://doi.org/10.1023/B:APIN.0000011138.20292.dd

[74] J.-Y. Cai, M. Fürer, and N. Immerman, “An optimal lower bound on the number of variables for graph identification,” *Combinatorica*, vol. 12, no. 4, pp. 389–410, Dec. 1992. [Online]. Available: https://doi.org/10.1007/BF01305232

[75] P. Barceló, E. V. Kostylev, M. Monet, J. Pérez, J. Reutter, and J. P. Silva, “The Logical Expressiveness of Graph Neural Networks,” Sep. 2019. [Online]. Available: https://openreview.net/forum?id=r1lZ7AEKvB

[76] S. Ståhlberg, B. Bonet, and H. Geffner, "Learning General Optimal Policies with Graph Neural Networks: Expressive Power, Transparency, and Limits," May 2022, arXiv:2109.10129 [cs]. [Online]. Available: http://arxiv.org/abs/2109.10129

[77] M. Grohe, "The Logic of Graph Neural Networks," Jan. 2022, arXiv:2104.14624 [cs]. [Online]. Available: http://arxiv.org/abs/2104.14624

[78] S. Ståhlberg, B. Bonet, and H. Geffner, "Learning General Policies for Classical Planning Domains: Getting Beyond C2," Mar. 2024, arXiv:2403.11734 [cs]. [Online]. Available: http://arxiv.org/abs/2403.11734

[79] J. Lee, Y. Lee, J. Kim, A. R. Kosiorek, S. Choi, and Y. W. Teh, "Set Transformer: A Framework for Attention-based Permutation-Invariant Neural Networks," May 2019, arXiv:1810.00825 [cs, stat]. [Online]. Available: http://arxiv.org/abs/1810.00825

[80] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "PyTorch: An Imperative Style, High-Performance Deep Learning Library," Dec. 2019, arXiv:1912.01703 [cs, stat]. [Online]. Available: http://arxiv.org/abs/1912.01703

[81] B. Lefaudeux, F. Massa, D. Liskovich, W. Xiong, V. Caggiano, S. Naren, M. Xu, J. Hu, M. Tintore, S. Zhang, P. Labatut, D. Haziza, L. Wehrstedt, J. Reizenstein, and G. Sizov, "xFormers: A modular and hackable Transformer modelling library," 2022. [Online]. Available: https://github.com/facebookresearch/xformers

[82] H. Li, X. Wang, Z. Zhang, and W. Zhu, "OOD-GNN: Out-of-Distribution Generalized Graph Neural Network," Dec. 2021, arXiv:2112.03806 [cs]. [Online]. Available: http://arxiv.org/abs/2112.03806

[83] Y. Sui, X. Wang, J. Wu, M. Lin, X. He, and T.-S. Chua, "Causal Attention for Interpretable and Generalizable Graph Classification," in *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, Aug. 2022, pp. 1696–1705, arXiv:2112.15089 [cs]. [Online]. Available: http://arxiv.org/abs/2112.15089

[84] A. Farahani, S. Voghoei, K. Rasheed, and H. R. Arabnia, "A Brief Review of Domain Adaptation," in *Advances in Data Science and Information Engineering*, R. Stahlbock, G. M. Weiss, M. Abou-Nasr, C.-Y. Yang, H. R. Arabnia, and L. Deligiannidis, Eds. Cham: Springer International Publishing, 2021, pp. 877–894.

[85] C. Cortes, Y. Mansour, and M. Mohri, “Learning Bounds for Importance Weighting,” in *Advances in Neural Information Processing Systems*, vol. 23. Curran Associates, Inc., 2010. [Online]. Available: https://papers.nips.cc/paper_files/paper/2010/hash/59c33016884a62116be975a9bb8257e3-Abstract.html

[86] M. Sugiyama, M. Krauledat, and K.-R. Müller, “Covariate Shift Adaptation by Importance Weighted Cross Validation,” *Journal of Machine Learning Research*, vol. 8, no. 35, pp. 985–1005, 2007. [Online]. Available: http://jmlr.org/papers/v8/sugiyama07a.html

[87] A. Gretton, A. Smola, J. Huang, M. Schmittfull, K. Borgwardt, B. Schölkopf, J. Candela, M. Sugiyama, A. Schwaighofer, and N. Lawrence, “Covariate Shift by Kernel Mean Matching,” *Dataset Shift in Machine Learning, 131-160 (2009)*, Jan. 2009.

[88] N. Yin, L. Shen, B. Li, M. Wang, X. Luo, C. Chen, Z. Luo, and X.-S. Hua, “DEAL: An Unsupervised Domain Adaptive Framework for Graph-level Classification,” in *Proceedings of the 30th ACM International Conference on Multimedia*, ser. MM ’22. New York, NY, USA: Association for Computing Machinery, 2022, pp. 3470–3479. [Online]. Available: https://doi.org/10.1145/3503161.3548012

[89] Y.-X. Wu, X. Wang, A. Zhang, X. He, and T.-S. Chua, “Discovering Invariant Rationales for Graph Neural Networks,” Jan. 2022, arXiv:2201.12872 [cs]. [Online]. Available: http://arxiv.org/abs/2201.12872

[90] S. Fan, X. Wang, C. Shi, P. Cui, and B. Wang, “Generalizing Graph Neural Networks on Out-Of-Distribution Graphs,” Mar. 2024, arXiv:2111.10657 [cs]. [Online]. Available: http://arxiv.org/abs/2111.10657

[91] D. Buffelli, P. Liò, and F. Vandin, “SizeShiftReg: a Regularization Method for Improving Size-Generalization in Graph Neural Networks,” Oct. 2022, arXiv:2207.07888 [cs]. [Online]. Available: http://arxiv.org/abs/2207.07888

[92] H. Li, Z. Zhang, X. Wang, and W. Zhu, “Learning Invariant Graph Representations for Out-of-Distribution Generalization,” *Advances in Neural Information Processing Systems*, vol. 35, pp. 11 828–11 841, Dec. 2022. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2022/hash/4d4e0ab9d8ff180bf5b95c258842d16e-Abstract-Conference.html

[93] M. Ghallab, D. Nau, and P. Traverso, *Automated Planning: Theory and Practice*. Elsevier, May 2004, google-Books-ID: eCj3cKC_3ikC.

[94] N. Pochter, A. Zohar, and J. Rosenschein, “Exploiting Problem Symmetries in State-Based Planners,” *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 25, no. 1, pp.

1004–1009, Aug. 2011, number: 1. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/8014

[95] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," Jan. 2017, arXiv:1412.6980 [cs]. [Online]. Available: http://arxiv.org/abs/1412.6980

[96] S. Ståhlberg, B. Bonet, and H. Geffner, "Learning General Policies with Policy Gradient Methods," *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning*, vol. 19, no. 1, pp. 647–657, Aug. 2023, conference Name: Proceedings of the 20th International Conference on Principles of Knowledge Representation and Reasoning. [Online]. Available: https://proceedings.kr.org/2023/63/

[97] F. Pommerening and A. Torralba, "IPC 2018," 2018. [Online]. Available: https://ipc2018-classical.bitbucket.io/

[98] W. Shen, F. Trevizan, and S. Thiébaux, "Learning Domain-Independent Planning Heuristics with Hypergraph Networks," Nov. 2019, arXiv:1911.13101 [cs]. [Online]. Available: http://arxiv.org/abs/1911.13101

[99] R. Karia and S. Srivastava, "Learning Generalized Relational Heuristic Networks for Model-Agnostic Planning," Oct. 2020, arXiv:2007.06702 [cs]. [Online]. Available: http://arxiv.org/abs/2007.06702

[100] P. Ferber, F. Geißer, F. Trevizan, M. Helmert, and J. Hoffmann, "Neural Network Heuristic Functions for Classical Planning: Bootstrapping and Comparison to Other Methods," *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 32, pp. 583–587, Jun. 2022. [Online]. Available: https://ojs.aaai.org/index.php/ICAPS/article/view/19845

[101] L. Chrestien, T. Pevný, S. Edelkamp, and A. Komenda, "Optimize Planning Heuristics to Rank, not to Estimate Cost-to-Goal," Oct. 2023, arXiv:2310.19463 [cs]. [Online]. Available: http://arxiv.org/abs/2310.19463

[102] T. Silver, S. Dan, K. Srinivas, J. B. Tenenbaum, L. P. Kaelbling, and M. Katz, "Generalized Planning in PDDL Domains with Pretrained Large Language Models," Dec. 2023, arXiv:2305.11014 [cs]. [Online]. Available: http://arxiv.org/abs/2305.11014

[103] A. Bajpai, S. Garg, and Mausam, "Transfer of Deep Reactive Policies for MDP Planning," Oct. 2018, arXiv:1810.11488 [cs]. [Online]. Available: http://arxiv.org/abs/1810.11488

[104] S. Garg, A. Bajpai, and Mausam, "Size Independent Neural Transfer for RDDL Planning," Apr. 2019, arXiv:1902.03081 [cs, stat]. [Online]. Available: http://arxiv.org/abs/1902.03081

[105] S. Garg and A. Bajpai, "Symbolic Network: Generalized Neural Policies for Relational MDPs," Jun. 2020, arXiv:2002.07375 [cs, stat]. [Online]. Available: http://arxiv.org/abs/2002.07375

[106] V. Sharma, D. Arora, F. Geißer, Mausam, and P. Singla, "SymNet 2.0: Effectively handling Non-Fluents and Actions in Generalized Neural Policies for RDDL Relational MDPs," in *Proceedings of the Thirty-Eighth Conference on Uncertainty in Artificial Intelligence*. PMLR, Aug. 2022, pp. 1771–1781, iSSN: 2640-3498. [Online]. Available: https://proceedings.mlr.press/v180/sharma22a.html

[107] V. Sharma, D. Arora, Mausam, and P. Singla, "SymNet 3.0: Exploiting Long-Range Influences in Learning Generalized Neural Policies for Relational MDPs," in *Proceedings of the Thirty-Ninth Conference on Uncertainty in Artificial Intelligence*. PMLR, Jul. 2023, pp. 1921–1931, iSSN: 2640-3498. [Online]. Available: https://proceedings.mlr.press/v216/sharma23c.html

[108] T. Ma, P. Ferber, S. Huo, J. Chen, and M. Katz, "Online Planner Selection with Graph Neural Networks and Adaptive Scheduling," Nov. 2019, arXiv:1811.00210 [cs, stat]. [Online]. Available: http://arxiv.org/abs/1811.00210

[109] O. Rivlin, T. Hazan, and E. Karpas, "Generalized Planning With Deep Reinforcement Learning," May 2020, arXiv:2005.02305 [cs]. [Online]. Available: http://arxiv.org/abs/2005.02305

[110] F. Teichteil-Königsbuch, G. Povéda, G. G. d. G. Barba, T. Luchterhand, and S. Thiébaux, "Fast and Robust Resource-Constrained Scheduling with Graph Neural Networks," *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 33, pp. 623–633, Jul. 2023. [Online]. Available: https://ojs.aaai.org/index.php/ICAPS/article/view/27244

[111] "torch_geometric.nn.conv.RGATConv — pytorch_geometric documentation." [Online]. Available: https://pytorch-geometric.readthedocs.io/en/latest/generated/torch_geometric.nn.conv.RGATConv.html#torch_geometric.nn.conv.RGATConv

Acknowledgments

I would like to express my gratitude to my advisor, Prof. Nicolò Navarin of the University of Padova, for his invaluable guidance and support throughout the duration of this research. I am also grateful to Andrea Micheli and Elisa Tosello from the FBK research institute, whose insights into automated planning greatly enhanced my understanding of the subject. Their constructive feedback on my results was instrumental to the success of this work. I would also like to thank my BDMA colleagues for their support and for the countless unforgettable memories we created together throughout the program. I am equally thankful to my friends back home, who, despite the distance, were always there for me. Lastly, I owe my deepest gratitude to my family, whose unwavering support has been a constant source of strength throughout this journey.