# Generalized Policies for Planning Domains

A report submitted for the course
*COMP8800, Advanced Computing Research Project*

24 pt research project, S2/S1 2023–2024


By:

**Rongjian Su**

**Supervisor:**
Dr. Felipe Werndl Trevizan


Australian
National
University

**School of Computing**
College of Engineering, Computing and Cybernetics (CECC)
The Australian National University


July 2024

## Declaration:

I declare that this work:

- upholds the principles of academic integrity, as defined in the University Academic Misconduct Rules;
- is original, except where collaboration (for example group work) has been authorised in writing by the course convener in the class summary and/or Wattle site;
- is produced for the purposes of this assessment task and has not been submitted for assessment in any other context, except where authorised in writing by the course convener;
- gives appropriate acknowledgement of the ideas, scholarship and intellectual property of others insofar as these have been used;
- in no part involves copying, cheating, collusion, fabrication, plagiarism or recycling.

July, Rongjian Su

# Abstract

Planning involves an agent automatically reasoning about the environment and inferring an action plan that achieves the goals with minimal cost based on predefined actions and limited resources. Currently, there are several methods to solve planning problems, including graph-based, SAT-based, and heuristic search planning methods. However, as the problems we work on are at least PSPACE-complete, these traditional planning algorithms struggle to find optimal solutions for large-size problems and complex domains within a reasonable time.

Given the limitations of traditional planning algorithms and the significant success of neural networks, researchers began to explore using neural networks to learn generalized policies for solving planning problems. The state-of-the-art model GBFS-GNN, which learns a generalized policy represented by a GNN using reinforcement learning, still faces several limitations: the plan quality for large-size problems and complex domains is poor, training times are too long, training on large problem-size datasets is challenging, and more importantly, it cannot train and infer on domains with higher-arity predicates.

In this project, we introduce several methods to improve these limitations, such as (1) the usage of advantage normalization, selecting the most likely action, and illegal edges deletion to improve the plan quality and planning efficiency; (2) a new incremental training procedure enabling to train on large problem-size datasets; and (3) handling domains with higher-arity predicates through different approaches. We evaluate the improved GBFS-GNN across multiple domains from the IPC 2023 Learning Tracks dataset. The experimental results demonstrate that the improved GBFS-GNN significantly enhances the plan quality and planning efficiency, reduces training time, and is capable of solving problems in domains with higher-arity predicates. Moreover, the plan quality and planning efficiency of the improved GBFS-GNN surpasses that of other advanced models.

# Table of Contents

Chapter 1

# Introduction

This chapter introduces planning, neural networks, and learning for planning and presents the contributions and outline of this thesis.

## 1.1 Planning

Formulating a well-reasoned plan before action is a sign of intelligence. Planning involves an agent automatically reasoning about the environment and inferring an action plan that achieves the goals with minimal cost based on predefined actions and limited resources (Geffner and Bonet, 2013). Planning is one of the fields where computer theory and industrial applications are most tightly integrated. It has been widely applied in aerospace, robotics, autonomous driving, and production scheduling. For example, it provides decision support for Mars rovers (Chien et al., 2000), analyzes network vulnerabilities (Boddy et al., 2005), controls industrial systems such as sheet metal processing (Gupta et al., 1998) and industrial printing (Do et al., 2008), and generates safe routes for commercial airlines (Geisser et al., 2020).

Currently, there are various methods to solve planning problems. The graph-based planning method (Blum and Furst, 1997) represents the planning problem through a planning graph and finds the solution with the backward backtracking search. The planning graph is a compressed representation of the state space, where the proposition and action mutexes explicitly represent the constraints in the planning problem. The SAT-based planning method (Kautz et al., 1992) regards the planning problem as propositional satisfiability (SAT) problem and encodes the problem description and constraints into a conjunctive normal form (CNF) as the input for the SAT problem, which is then solved by an efficient SAT solver to construct the planning solution. The heuristic search planning method treats the planning problem as a state space search problem, using common search algorithms such as Greedy Best First Search (GBFS) and A$^*$ to solve the planning

problem. To improve search efficiency, these planning methods typically use a heuristic to guide the search, with representative heuristic functions including the additive/max heuristic (Bonet and Geffner, 2001) and heuristic based on the relaxed plan (Hoffmann and Nebel, 2001).

## 1.2 Neural Networks

Artificial Neural Networks (ANNs) are computational models that mimic the structure and function of biological neural networks. Based on deep neural networks, deep learning (DL) learns inherent patterns in large-scale datasets, achieving automatic feature extraction and recognition. Its performance has surpassed that of human experts and traditional algorithms in various fields, and it is widely applied in computer vision (CV), natural language processing (NLP), healthcare, finance, and other areas. For instance, Convolutional Neural Networks (CNNs) (LeCun et al., 1998) have become the benchmark in the CV field, greatly outperforming traditional handcrafted algorithms like SIFT (Lowe, 1999) and HOG (Dalal and Triggs, 2005); Recurrent Neural Networks (RNNs) (Elman, 1990) have significantly improved language understanding and text generation abilities in the NLP field, greatly outperforming traditional handcrafted algorithms like BoW and n-gram. Neural network models based on Transformers (Vaswani et al., 2017), such as Swin Transformer (Liu et al., 2021) and ChatGPT (Brown et al., 2020), even outperform CNNs and RNNs in CV and NLP fields, demonstrating exceptional ability in complex patterns and sequential dependencies.

Deep reinforcement learning (DRL), inspired by the behavior of animals to seek rewards and avoid punishments, uses neural networks to represent environment states and action policies, using the rewards obtained from the interaction of the agent with the environment as feedback signals to train the agent, and optimizing the policy for problem-solving. This method has surpassed human experts and traditional algorithms in strategic planning and decision-making abilities and has been widely applied in games, robotics, parameter optimization, and other areas. For example, Google's AlphaGo (Silver et al., 2016) used DRL to successfully defeat the human world champion in Go at 2017. At the meantime, it is also considered an important pathway towards general artificial intelligence.

## 1.3 Learning for Planning

Given the limitations of traditional planning algorithms and the significant success of neural networks in multiple fields, researchers began to explore the application of neural networks in planning. Currently, neural network models used in planning can be divided into two categories: learning heuristics and learning generalized policies.

Learning heuristics involves using neural networks to learn and infer heuristics, and combining the output heuristics with heuristic search algorithms to solve planning problems. A typical example is STRIPS-HGN (Shen et al., 2020), which uses hypergraph

neural networks to represent state nodes. By relaxing the negative effects of actions, it constructs a hypergraph model between actions and propositions for training domain-dependent and domain-independent heuristics.

Learning generalized policies involves using neural networks to learn a generalized policy, inferring the action probability distribution based on the learned policy, and selecting actions through sampling to solve planning problems. For instance, GBFS-GNN (Rivlin et al., 2020) utilizes the graph neural network (GNN) (Scarselli et al., 2008) to represent the state and uses reinforcement learning techniques to learn generalized policies. It then combines the neural network's output of the action probability distribution with an improved GBFS search algorithm to solve planning problems.

## 1.4 Contributions

The objective of this thesis is to improve and remove some of the current limitations of GBFS-GNN, a state-of-the-art learning for planning model. The aim is to enable GBFS-GNN to train on large problem-size datasets and domains with arbitrary arity predicates within a reasonable time while reducing the training time and improving planning performance. The contributions of this thesis are as follows:

### Improving Training and Inference Performance

The original GBFS-GNN's planning performance for large-size problems and complex domains is poor, and its training times are too long. Moreover, the original GBFS-GNN was trained using a small problem-size dataset, and it cannot complete training on large problem-size datasets, such as the IPC 2023 Learning Tracks dataset, within a reasonable time. Therefore, we replicate the original GBFS-GNN and propose four optimization methods to address these limitations:

- **Advantage Normalization**: We use advantage normalization to standardize advantage estimation, reducing its variance, improving training stability, accelerating model convergence, and enhancing model performance.
- **Selecting Most Likely Action**: During inference, we select the action with the highest probability instead of sampling an action. This approach fully utilizes the trained policy, converging more rapidly to the optimal solution and improving planning performance.
- **Illegal Edges Deletion**: We delete illegal edges between nodes in the state graph (a directed complete graph) that do not share binary relations to reduce training and inference time and improve performance.
- **Incremental Training Procedure**: We divide the dataset sorted by problem size into multiple splits. Training begins with the first split, and after several epochs, the next split is merged into the current training set and training is resumed. This process is repeated until the entire dataset is used for training. Training is

stopped if the performance on the validation set does not improve for several consecutive evaluations. This method enables GBFS-GNN to train on large problem-size datasets within a reasonable time while saving computational resources and training time.

### Handling Domains with Higher-arity Predicates

The original GBFS-GNN can only encode 0-ary, 1-ary, and 2-ary predicates (low-arity predicates) as global, node, and edge features in the state graph, and it is unable to solve problems in domains with higher-arity predicates. To address this limitation, we propose two types of solutions:

- **Decomposition**: We decompose higher-arity predicates in the domain into binary predicates. The decomposed domain and problem are then input into the original GBFS-GNN for encoding, training, and inference. We develope two decomposition methods, relational decomposition and token-based decomposition, and compare their tradeoffs.

- **Architecture Modification**: We modify the architecture of the original GBFS-GNN to directly encode higher-arity predicates as nodes and edges of the state graph. We propose two modification methods: (1) simulating the relational decomposition method by representing higher-arity predicates as binary predicates for encoding states and effects; (2) simulating the token-based decomposition through introducing the hub nodes in the state graph representing the relation between parameters of higher-arity predicates to encode states and effects.

### Standardized Evaluation

The original GBFS-GNN trains and infers on their own ad-hoc datasets. Their experimental results depend heavily on the quality, quantity, and difficulty of the problems in the datasets, making it challenging to fairly and accurately evaluate the models' performance. Therefore, we use the IPC 2023 Learning Tracks dataset, a public dataset in the learning for planning field, to conduct experiments and compare the performance with other models participating in the IPC 2023 Learning Tracks to ensure fairness and accuracy.

Moreover, since only the Rovers domain in the IPC 2023 Learning Tracks dataset contains higher-arity predicates, we construct the Grippers domain dataset mimicking the structure and difficulty of the IPC 2023 Learning Tracks dataset to test our model, which could also be used for experiments on other models in domains with higher-arity predicates.

## 1.5 Thesis Outline

- **Chapter 2 – Background and Related Work** aims to provide the reader with background knowledge on planning and neural networks and relevant works in the learning for planning field.
- **Chapter 3 – GBFS-GNN for Domains with Low-arity Predicates** first discusses the original GBFS-GNN and its limitations. Then, it proposes four optimization methods: advantage normalization, selecting the most likely action, illegal edges deletion, and incremental training procedure to address the performance limitations.
- **Chapter 4 – GBFS-GNN for Domains with Higher-arity Predicates** proposes two types of solutions, decomposition and architecture modification, to address the limitation of the original GBFS-GNN's inability to handle domains with higher-arity predicates.
- **Chapter 5 – Evaluation** describes the datasets and experimental setup, presents the relevant experimental results in several domains, and provides a thorough discussion.
- **Chapter 6 – Conclusion and Future Work** summarizes our contributions and proposes several future research directions.

## 1.6 Terminology

**Object**: An entity within a planning problem.

**Low-arity predicate**: Predicates that include two or fewer parameters.

**Higher-arity predicate**: Predicates that include three or more parameters.

**Encoding**: An initial feature vector specified manually.

**Embedding**: A feature vector obtained through neural network training.

**State graph**: A graph output by the encoding of a state.

**State graph component**: Components of a state graph, including global attributes, nodes, and edges.

**Original GBFS-GNN**: The GBFS-GNN as proposed in the original paper.

**Improved GBFS-GNN**: A neural network model that applies various optimizations to the original GBFS-GNN.

**Improved GBFS-GNN (relational)**: The improved GBFS-GNN using simulating the relational decomposition method of architecture modification.

**Improved GBFS-GNN (token)**: The improved GBFS-GNN using simulating the token-based decomposition method of architecture modification.

**Relation token**: A predicate parameter or object introduced when decomposing a predicate or proposition to represent the relationships between parameters of a predicate.

**Relation node / Hub node**: A node corresponding to a relation token in the state graph.

**Relation edge**: An edge between a relation node and other nodes in the state graph.

Chapter 2

# Background and Related Work

This chapter provides readers with background knowledge related to planning and neural networks and relevant works in the learning for planning field. Section 2.1 discusses the planning representations and algorithms for solving planning problems. Section 2.2 covers concepts and neural network models in deep learning and reinforcement learning. Finally, Section 2.3 presents current neural network approaches applied to planning.

## 2.1 Planning

Planning involves an agent reasoning about the environment, making decisions based on the predefined initial states, goal conditions, and actions, and ultimately producing a sequence of actions that achieve the goals with minimal cost. Depending on the abstraction levels of states, environment, and actions, planning problems can be divided into classical and non-classical planning problems. Classical planning problems are based on strong constraints, requiring a known initial state, deterministic actions without durations, and a fully observable environment. In this thesis, we mainly focus on classical planning. A planning system will first represent the planning problem in a formal language and then solve the planning problem. Therefore, we introduce the formal representations of planning problems in Subsection 2.1.1, followed by algorithms to solve planning problems in Subsection 2.1.2.

### 2.1.1 Representations in Planning

#### Classical Planning Representation

A classical planning problem can be represented as the tuple $\langle S, s_0, S_G, A, f, c\rangle$ (Geffner and Bonet, 2013), where:

- $S$ represents the set of states;

- $s_0 \in S$ represents the initial state;
- $S_G \subset S$ represents the set of goal states;
- $A$ represents the set of actions. $A(s) \subseteq A$ represents the subset of actions executable in the state $s \in S$;
- $f(s, a) : S \times A \to S$ is the state transition function, indicating that executing action $a \in A(s)$ in state $s \in S$ results in a new state $s' \in S$;
- $c(s, a) : S \times A \to \mathbb{R}^+$ is the action cost function, indicating the cost of executing action $a \in A(s)$ in state $s \in S$.

Classical planning problems can be transformed into a search problem: starting from the initial state, finding a sequence of actions $\pi = [a_1, a_2, ..., a_T]$ called a plan to reach any goal state. At each step $t = 1, ..., T$, the cost for the agent to execute an action in the current state is $c(t) = c(s_t, a_t)$, then the cost of the plan is $\sum_{i=1,...,T} c(i)$. An optimal planner finds the minimal cost plan $\pi^*$ among all feasible plans $\Pi$, that is $\pi^* = \text{argmin}_{\pi \in \Pi} \sum_{i=1,...,T} c(i)$.

### STRIPS Representation

STRIPS (Fikes and Nilsson, 1971) is one of the earliest general planning description languages. It utilizes a conjunctive form of propositions to describe states, represented as the tuple $\langle F, O, I, G, c \rangle$, where:

- $F$ represents the set of propositions. Each state $s \subseteq F$ includes one or more propositions;
- $O$ represents the set of actions. Each action $o \in O$ can be described as the tuple $\langle pre(o), add(o), del(o) \rangle$, where $pre(o) \subseteq F$ represents the precondition of the action $a$, and if $pre(o) \subseteq s$, it implies that action $o$ can be executed in state $s$; $add(o) \subseteq F$ and $del(o) \subseteq F$ represent the additive and delete effects of executing action $o$. The successor state $s' = (s \setminus del(o)) \cup add(o)$ is achieved by executing action $o$ in state $s$;
- $I \subseteq F$ represents the initial state;
- $G \subseteq F$ represents the set of goal states;
- $c(o) : O \to \mathbb{R}^+$ is the action cost function, representing the cost of executing action $o \in O$.

It is apparent that STRIPS is an encoding form of the classical planning representation $\langle S, s_0, S_G, A, f, c \rangle$.

### PDDL Representation

The Planning Domain Definition Language (PDDL) (Aeronautiques et al., 1998) is a STRIPS-based general planning description language that uses a domain file and a prob-

lem file to describe the planning domain and specific planning problems. The domain file includes domain knowledge unrelated to specific planning problems, mainly including types of objects, predicates, and action schemas. The problem file details a specific planning problem, mainly including objects, the initial state, and goal conditions. A domain file combined with different problem files constitutes various specific planning tasks within a domain. For instance, in the Blocksworld domain and problem file:

```
(define (domain blocksworld)
    (:requirements :strips)
    (:types block)
    (:predicates
        (arm-empty)
        (clear ?x - block)
        (on-table ?x - block)
        (holding ?x - block)
        (on ?x - block ?y - block)
    )
    (:action pickup
        :parameters (?b - block)
        :precondition (and
            (clear ?b)
            (on-table ?b)
            (arm-empty)
        )
        :effect (and
            (holding ?b)
            (not (clear ?b))
            (not (on-table ?b))
            (not (arm-empty))
        )
    )
    ...
)

(define (problem blocksworld-01)
    (:domain blocksworld)
    (:objects b1 b2 - block)
    (:init
        (arm-empty)
        (clear b2)
        (on-table b2)
        (clear b1)
        (on-table b1)
    )
```

```
    (:goal (and
        (clear b1)
        (on b1 b2)
        (on-table b2)
    ))
)
```

In the domain file, the predicate $on(?x-block, ?y-block)$ indicates that the "block" type variable $?x$ is on top of the "block" type variable $?y$. The action schema $pickup(?b-block)$ indicates picking up the "block" type variable $?b$ from the table. This action schema requires the precondition that the predicates $clear(?b), on\text{-}table(?b), arm\text{-}empty$ are true to be executable. Executing this action schema results in a successor state where the predicate $holding(?b)$ is true, and the predicates $clear(?b), on\text{-}table(?b), arm\text{-}empty$ are false.

In the problem file, two "block" type objects $b_1$ and $b_2$ are defined. Predicates and action schemas can be instantiated through these objects into propositions and actions, thereby converting the PDDL into a STRIPS problem. The initial state and goal conditions are represented by (: *init*...) and (: *goal*...).

### 2.1.2 Heuristic Search Planning

Currently, there are various methods for solving classical planning problems, among which the most widely used is the state-space heuristic search. This approach views the state space of the planning problem as a directed graph, representing states as nodes in the graph, actions as directed edges from nodes that satisfy the action's preconditions to the nodes of the resulting successor states, and the directed edge weight as the cost of action. Therefore, solving the classical planning problem can be transformed into a shortest path search problem from the initial state node to a goal state node. Additionally, heuristic search methods employs a heuristic $h(s)$ as an estimate of the cost from state $s$ to a goal state, guiding the search algorithm to choose the best branch for searching, rapidly crossing the state space, thereby significantly reducing the computational complexity of the search process.

#### Heuristics

The heuristic function $h(s) : S \to \mathbb{R}^+$ is used to evaluate the cost from state $s$ to a goal state, enabling heuristic search algorithms to focus their search on the paths most likely to lead to a goal state. If for every state $s$ in the state space, it holds that $h(s) \leq h^*(s)$, then this heuristic function is admissible. Combined with an appropriate search algorithm, such as A$^*$, it can guarantee finding the optimal solution to the planning problem.

**Delete-Relaxation** In classical planning, the delete-relaxation is a commonly used relaxation to compute heuristics. For a STRIPS problem $P = \langle F, O, I, G, c \rangle$, the delete-relaxation is obtained

by removing the delete effects of each action $o \in O$. The original problem $P$ is transformed into $P' = \langle F, O', I, G, c \rangle$, where $O' = \{\langle pre(o), add(o), \phi \rangle \mid o \in O\}$, resulting in a simpler delete-relaxation problem. This approach, by ignoring the delete effects of actions, allows the number of true propositions in a state to monotonically increase, making each state contain more information. Therefore, the set of applicable actions is monotonically non-decreasing during search, thereby making the calculation of the cost estimate easier.

**FF heuristic**

In this thesis, we utilize the commonly used delete-relaxation heuristic $h^{FF}$ (Hoffmann and Nebel, 2001) for training and inference. $h^{FF}$ employs graph planning methods to non-optimal solve the delete-relaxation problem and uses either the total number of actions or the sum of action costs from the plan leading to each goal proposition as the heuristic estimate for the original problem. If the plan for solving the delete-relaxation problem is optimal, then $h^{FF}$ is admissible. But in practice, the admissible of $h^{FF}$ is often sacrificed for higher computational efficiency.

### Algorithms

**GBFS**

The Greedy Best-First Search (GBFS) algorithm is a widely used heuristic search algorithm. It begins from the initial state node, selecting the state node with the minimal $f(s) = h(s)$ from the candidate set for expansion, where $h(s)$ represents the heuristic estimate of the state. Then, successor state nodes generated by the expansion are added to the candidate set. This process repeats until a goal state node is reached or all state nodes have been explored. GBFS focuses only on the future costs when evaluating nodes and does not consider the cost already incurred to reach the current state node. GBFS is simple to implement and has high search efficiency, enabling it to find the plan that solves the planning problem quickly. However, GBFS does not ensure that the plan found is optimal.

**A$^*$**

The A$^*$ algorithm evaluates nodes by considering both the cost already incurred and the future costs. It starts from the initial state node, selecting the state node with the minimal $f(s) = g(s) + h(s)$ from the candidate set for expansion, where $h(s)$ represents the heuristic estimate of the state and $g(s)$ represents the cost to reach the current state. Then, successor state nodes generated by the expansion are added to the candidate set. This process repeats until a goal state node is reached or all state nodes have been explored. If the heuristic used is admissible, A$^*$ ensures that the plan found is optimal. Compared to GBFS, A$^*$ is more comprehensive and accurate, capable of finding higher quality plans despite its lower search efficiency.

## 2.2 Neural Networks

Artificial Neural Networks (ANNs) are computational models that mimic the structure and function of biological neural networks. It originated in the 1940s and has made significant progress through ongoing research, solving numerous problems that traditional

algorithms could not address and outperforming traditional algorithms in many fields. Thus, ANNs are widely used in computer vision (CV), natural language processing (NLP), robotics, healthcare, and finance. We introduce the deep learning concepts and models, including MLP and GNN, in Subsection 2.2.1. Then, Subsection 2.2.2 presents the reinforcement learning concepts and models, including Actor-Critic and PPO.

### 2.2.1 Deep Learning

Deep learning (DL) is a machine learning technique based on deep neural networks, primarily focusing on learning inherent patterns in large-scale datasets to achieve feature extraction and recognition. The core of deep learning is to automatically extract and combine low-level features from raw data to learn high-level abstract representations through multi-level non-linear transformations without the need for manually designed feature extraction algorithms. This technique has surpassed traditional algorithms and human expert performance in many fields.

#### MLP

The Multilayer Perceptron (MLP) (Rumelhart et al., 1986) is one of the simplest feed-forward neural networks. Figure 2.1 illustrates the structure of an MLP with one hidden layer. An MLP consists of multiple neuron layers, each containing several neurons. As shown in Figure 2.2, each neuron represents a function that applies a weighted sum to multiple inputs and then uses non-linear activation functions such as Sigmoid, ReLU, or Tanh to produce an output. The formula is as follows:

$$y = \phi(\sum_{i=0}^{n} w_i \cdot x_i + b)$$

where $w$ is the weight of summation, $b$ is the bias, $\phi$ is the non-linear activation function.

In an MLP, information flows only in one direction: from the input nodes through the neurons of the hidden layers to the output nodes, without any cycles. MLPs are fully connected between adjacent layers, with no connections within the same layer. The first layer of an MLP receives the input feature vector, which then passes through one or more hidden layers to compute hidden representations, eventually producing the final output of the MLP in the output layer. As the number of hidden layers increases, the neural network extracts more complex features from the input. The formula for computing the hidden representations and final output in the MLP shown in Figure 2.1 is as follows:

$$\mathbf{h} = \phi(W_h\mathbf{x} + \mathbf{b}_h)$$
$$\mathbf{y} = \phi(W_y\mathbf{h} + \mathbf{b}_y)$$

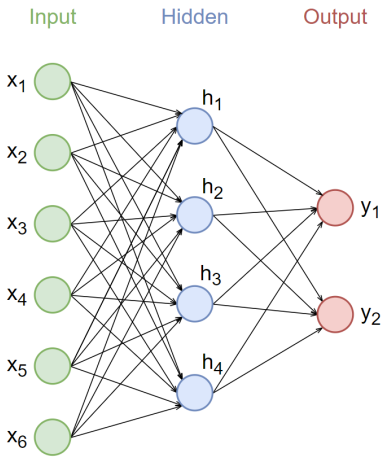


Figure 2.1: An example of a MLP with 6 neurons in the input layer, 4 neurons in a hidden layer, and 2 neurons in the output layer.

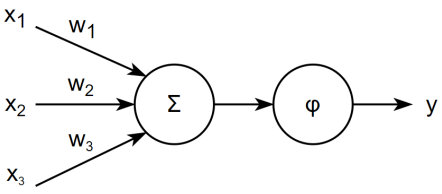


Figure 2.2: An example of a single neuron with 3 inputs, 3 weight, and 1 output.

where $W_h$ and $\mathbf{b}_h$ represent the weight matrix and bias vector from the input layer to the hidden layer, $W_y$ and $\mathbf{b}_y$ represent the weight matrix and bias vector from the hidden layer to the output layer, $\mathbf{x}$ is the input feature vector, $\mathbf{h}$ is the hidden representation, $\mathbf{y}$ is the final output vector.

### GNN

Traditional deep neural network models like Convolutional Neural Networks (CNNs) (LeCun et al., 1998) are mainly used for processing Euclidean space data such as images. Euclidean space has direction and location, and each node has a fixed number of neighboring nodes. Therefore, we can use the same convolutional kernel to operate on all nodes. However, for non-Euclidean space data such as graphs, the number of neighboring nodes per node is not fixed, and non-Euclidean space lacks the direction and location, which is challenging for traditional deep neural network models to handle efficiently.

Graph Neural Networks (GNNs) (Scarselli et al., 2008) are neural network models designed to handle graph-structured data, generalizing traditional deep learning techniques to non-Euclidean space. The goal of GNNs is to learn a representation vector for each node in a graph based on the structural features of the graph that encodes information from neighboring nodes. GNNs use a message-passing framework, which mainly involves two steps:

1. Collecting messages from neighboring nodes, which is the process of information propagation;
2. Using neural networks to update node representations, which is the process of information aggregation.

GNNs can be divided into two categories:

- **Spectral-based GNNs** model from a traditional graph signal processing perspective and define graph convolution operations in the Fourier domain of graph data based on spectral graph theory, having strict mathematical derivations and theoretical support. Graph Convolutional Network (GCN) (Kipf and Welling, 2016) is a widely applied spectral-based GNN model, utilizing the eigendecomposition of the Laplacian matrix to aggregate and update node features.
- **Spatial-based GNNs** learn node representations by aggregating information from neighboring graph components from the graph structure perspective. Graph Networks (GNs) (Battaglia et al., 2018) are one of the best-known general spatial-based GNN frameworks, which generalize several existing GNNs and have a strong relational inductive bias.

**GN** In GNs, a graph is defined as the tuple $G = \langle u, V, E \rangle$, where $u$ represents global attributes, $V = \{V_i\}$ represents a set of nodes, and $E = \{\langle e_k, r_k, s_k \rangle\}$ represents a set of edges, where $r_k$ is the receiving node and $s_k$ is the sending node. The aggregation

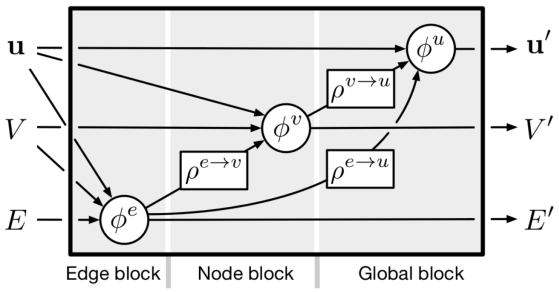


Figure 2.3: The computational steps of the GN block are as follows: (1) Use $\phi^e$ to update each edge's embedding, use $\rho^{e\to v}$ to aggregate the embeddings of edges starting from $V_i$ for subsequent node updates, and use $\rho^{e\to u}$ to aggregate all edge's embeddings for subsequent global attribute updates; (2) Use $\phi^v$ to update each vertex's embedding, use $\rho^{e\to v}$ to aggregate all vertex embeddings for subsequent global attribute updates; (3) Use $\phi^u$ to update the global attributes. Source: Battaglia et al. (2018)

and update functions in GNs are as follows, with the computational steps illustrated in Figure 2.3:

$$
\begin{aligned}
&e'_k = \phi^e(e_k, v_{r_k}, v_{s_k}, u), \bar{e}'_i = \rho^{e\to v}(E'_i), \bar{e}' = \rho^{e\to u}(E') \\
&v'_i = \phi^v(\bar{e}'_i, v_i, u), \bar{v}' = \rho^{v\to u}(V') \\
&u' = \phi^u(\bar{e}', \bar{v}', u)
\end{aligned}
$$

where $E'_i = \{\langle e'_k, r_k, s_k\rangle\}_{r_k=i}$, $E' = \{\langle e'_k, r_k, s_k\rangle\}$, $V' = \{V'_i\}$

GN blocks offer flexible representations, allowing global, node, and edge attributes to be represented in various formats, including vectors, tensors, sequences, sets, and even graphs. The output of a GN block can also be customized based on the specific tasks. Furthermore, the structure and aggregation and update functions within GN blocks can be configured in various ways, and diverse GNN models can be derived through different configurations. Additionally, complex model structures can be constructed by stacking any number of GN blocks.

Moreover, for a given node in a graph, the influence of its neighboring nodes may vary. Therefore, attention mechanisms have been integrated into the GNN's message-passing process, assigning different weights to neighboring nodes and updating the hidden state of a node using the weighted aggregation of neighboring nodes. Velickovic et al. (2017) proposed the Graph Attention Network (GAT), which employs a multi-head attention mechanism to represent relationships between nodes, capturing the varying importance of neighboring nodes to a central node, thereby enabling adaptive weight allocation to neighboring nodes.

### Training

**Loss**

The loss function measures the difference between the model outputs and the truth labels, and the training objective of deep neural networks is to minimize the loss by optimizing model parameters. The loss function formula is as follows:

$$\hat{y} = f(x, \theta)$$
$$\mathcal{L} = L(\hat{y}, y)$$

where $x$ represents the input feature, $\theta$ represents all the parameters, including weight matrices and bias vectors, $\hat{y}$ is the model output, $y$ is the truth label, and $L$ is the loss function, such as the mean squared error (MSE) loss function or cross-entropy loss function. To prevent overfitting in neural networks, a regularization term is usually included in the loss function, calculated as follows:

$$\mathcal{L} = L(\hat{y}, y) + \lambda \cdot \Omega(\theta)$$

**Gradient Descent**

Deep learning typically uses the gradient descent algorithm to update the weights and biases of the neural network. Specifically, during each iteration, the gradient descent algorithm calculates the gradient of the loss with respect to each parameter and adjusts the parameters in the direction of the negative gradient to minimize the loss.

**Backpropagation**

Backpropagation (Rumelhart et al., 1986) is an efficient technique for computing parameter gradients, consisting of forward and backward propagation stages. During forward propagation, data flows from the input layer to the output layer, computing the corresponding hidden representations at each hidden layer. After reaching the output layer, the loss is evaluated by a loss function. Back propagation starts from the output layer and propagates errors backward through the network, using the chain rule to calculate the error gradients for each layer, which are then used to update the weights and biases of the neural network.

## 2.2.2 Reinforcement Learning

Reinforcement learning (RL) is inspired by the behavior of animals seeking rewards and avoiding harm and trains agents with rewards received from their interactions with the environment as feedback signals. It is generally represented using a Markov decision process described by the tuple $\langle S, A, R, P \rangle$, where:

- $S$ represents the state space of the environment;
- $A$ represents the action space of the agent;
- $R$ represents the reward function, which evaluates the quality of the actions performed by the agent;

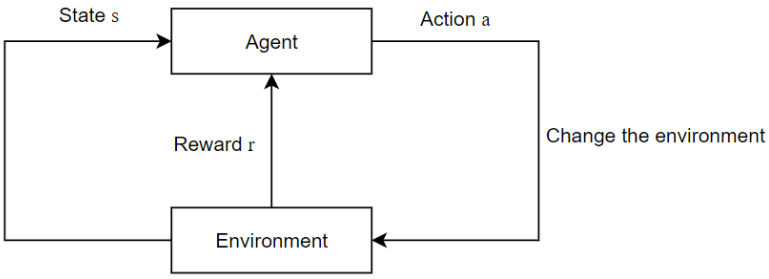


Figure 2.4: The structure of reinforcement learning.

- $P$ represents the state transition probability function, where for every state $s \in S$ and every action $a \in A$, $p(s'|s,a)$ is the probability distribution for transitioning to the next state $s'$.

The agent's policy $\pi$ maps the state space to the action space. At each time step $t$, based on the current state $s_t \in S$ and the policy $\pi$, the agent takes action $a_t \in A$. It then transitions to the next state $s_{t+1} \in S$ according to the state transition probability function $P$, while receiving a reward $r_t$ from the environment according to the reward function $R$. Due to the Markov property, the new state $s_{t+1}$ depends only on the current state $s_t$ and action $a_t$, not on any prior states or actions. The objective of reinforcement learning is to continually interact with the environment to optimize the policy $\pi$, thereby maximizing the cumulative reward. The overall framework of reinforcement learning is shown in Figure 2.4.

Deep learning focuses on the representation of objects, while reinforcement learning emphasizes learning policies to solve problems. Deep reinforcement learning (DRL) combines both advantages, utilizing deep learning to learn abstract representations from large-scale data automatically, and based on these representations, it conducts self-motivated reinforcement learning to optimize problem-solving policies. DRL is highly generalizable, with agents learning and constructing their knowledge directly from raw input signals without prior domain knowledge or human intervention. Therefore, DRL has been widely applied in game, robotics, and parameter optimization and is considered a significant way toward general artificial intelligence.

### Value-based Model vs. Policy Gradient-based Model

Deep reinforcement learning models are mainly divided into value-based and policy gradient-based models. Value-based models use deep neural networks to approximate the value function and iteratively update it using TD algorithms or Q-learning, indirectly obtaining the optimal policy from the optimal value function. A widely used value-based model is the Deep Q-Network (DQN) (Mnih et al., 2015). Policy gradient-based models establish a policy network to learn the action probability distribution for each state and optimize the parameters by computing the gradient of the expected rewards, eventually converging on the optimal policy. A common policy gradient-based model is the Trust

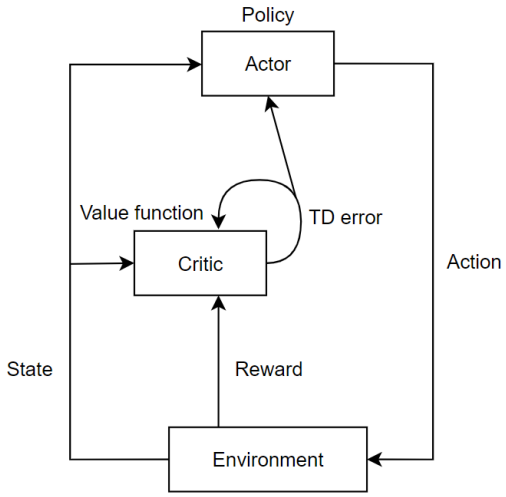


Figure 2.5: The structure of Actor-Critic.

Region Policy Optimization (TRPO) (Schulman et al., 2015).

### Actor-Critic

Regarding application scenarios, value-based models output the value of each action and are suitable only for discrete action space; policy gradient-based models can directly output the action probability distribution, thus suitable for both discrete and continuous action space. However, policy gradient-based models have poorer convergence and require sampling a series of data at each iteration to update the policy gradient, unlike value-based models, which can perform single-step updates.

Actor-Critic models (Sutton et al., 1999) integrate the advantages of both value-based and policy gradient-based models, also addressing their shortcomings. Its structure includes a policy network (Actor) that selects and updates actions based on the policy gradient algorithm and a value network (Critic) that evaluates actions based on the value function. During training, the parameters of the policy and value networks are updated alternately. Actor-Critic can handle continuous actions like policy gradient-based models, turning the sequential updates of policy gradient-based models into single-step updates and reducing the variance of the policy gradient algorithm. The structure of the Actor-Critic framework is shown in Figure 2.5.

### PPO

The Proximal Policy Optimization (PPO) (Schulman et al., 2017) model is a reinforcement learning model based on the Actor-Critic framework. Previous policy gradient-based models were highly sensitive to update steps, often causing significant differences between old and new policies during training, leading to instability and poor performance. PPO aims to optimize the policy within a certain constraint range, avoiding too

significant changes that could destabilize training and degrade performance. Thus, PPO uses an "alternative" objective to limit the difference between new and old policies:

$$L(\theta) = \hat{\mathbb{E}}\left[\min(r_t(\theta), \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon))\hat{A}_t\right]$$

where $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$ represents the ratio of new to old policies, $\hat{A}_t$ represents the advantage estimates, and $\text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)$ clips the ratio $r_t(\theta)$ to keep it within $[1-\epsilon, 1+\epsilon]$. PPO iterates with small batches of samples over multiple episodes, optimizing using the ratio of new to old policies and limiting the extent of policy improvement with the clip function. Among many reinforcement learning models, PPO is stable, adaptable, performs well, and is easier to implement than the earlier TRPO model.

## 2.3 Learning for Planning

Before the popularity of machine learning, solving planning problems primarily relied on traditional algorithms and heuristics such as GBFS and $h^{FF}$. In recent years, with the significant successes of neural networks in CV, NLP, and game, deep learning and reinforcement learning began to be applied in the planning field. Currently, neural network models used in planning can be divided into two categories: learning heuristics and learning generalized policies, which we will discuss in Subsection 2.3.1 and Subsection 2.3.2, respectively.

### 2.3.1 Learning Heuristics

Learning heuristics can be seen as supervised learning that uses neural networks to train and infer heuristics, which are then combined with heuristic search algorithms such as A$^*$ and GBFS to solve planning problems. Yoon et al. (2008) learn a domain-dependent linear heuristic function through $\delta(s) = h^*(s) - h^{FF}(s)$, which are linear combinations of features extracted from delete-relaxation plans, including the length of the delete-relaxation plan and ignored delete effects information.

Arfaee et al. (2010) employs a bootstrap learning technique that adaptively updates training data, learning a stronger heuristic function from training samples provided by a weak heuristic function. Starting from a weak heuristic model, if the heuristic can solve a training problem, the state sequence of the plan is added to the training set. If it cannot, states are generated starting from a goal state using a random walk algorithm and added to the training set. The weak heuristic model is then trained with the updated dataset to develop a stronger heuristic model. By iterating these steps, the model's performance continuously improves. Geissmann (2015) and Ferber et al. (2022) have shown that this bootstrap learning technique can be used to learn domain-specific heuristic functions.

Garrett et al. (2016) does not treat learning heuristic as a regression problem like previous research but instead approaches it as a state ranking problem. Using feature vectors of

action pairs as inputs, he trains a RankSVM to rank states. Features for each pair of actions $(a_1, a_2)$ include the intersection of preconditions and effects of $a_1$ and $a_2$, their temporal order in approximate plans, and existing domain-independent heuristic estimates. Ultimately, the state rankings rather than the estimated cost from the current state to a goal state are used as the domain-dependent heuristic estimate, which achieves better results than the regression-based learning heuristic. Hao et al. (2024) extends this ranking approach by using DL methods to compute state embeddings as well as rankings.

STRIPS-HGN (Shen et al., 2020) is a supervised learning method that uses the hypergraph neural network, a generalization of GNN, to represent state nodes and constructs a hypergraph between actions and propositions under the delete-relaxation problem for training the domain-dependent and domain-independent heuristics. It supports generalization across different problem sizes and performs better than $h^{max}$. Furthermore, the heuristics learned by STRIPS-HGN can be combined with any heuristic search algorithms, including A$^*$, to achieve near-optimal plans, unlike models like Garrett's, where learned heuristics can only be combined with rank-based heuristic search algorithms, such as GBFS. GOOSE (Chen et al., 2024) is another GNN method with a novel graph representation created specifically for learning heuristics.

Learning heuristics models can lead to heuristics with better performance than traditional heuristics, and some models can learn domain-independent heuristics, which eliminates the need for neural networks to be trained separately for each domain. However, these models cannot directly output a probability distribution of candidate actions, and the ultimate effectiveness depends on the heuristic search algorithm. Moreover, this approach needs to use the neural network to infer heuristic estimates for all successor states to select the best action at each state, which will significantly increase runtime.

### 2.3.2 Learning Generalized Policies

Given a specific planning problem with state set $S$ and action set $A$, a policy $\pi : S \to A$ is a function that maps the current state to an action to be taken, which can only output a solution for a particular problem. We aim to find a higher level of generalized policy that can be used to solve any problem within the given domain, thus sharing the underlying planning logic. For example, in the Blocksworld domain, finding the optimal solution is usually an NP-hard problem. However, we can use a generalized policy—first, unstack all blocks and put them on the floor; then, starting from the bottom blocks, stack blocks according to the target configuration—to find a feasible solution for any size of Blocksworld problem in polynomial time, although this policy and plan are not optimal.

Neural networks have the ability to infer general principles that remain valid beyond the observed range from examples. Thus, we can use neural networks to learn such a generalized policy from small-size problems within a domain and then use the learned generalized policy to solve large-size problems in the same domain. From another point of view, heuristic search algorithms can be considered a manually designed generalized

policy, and we can use the powerful learning capabilities of neural networks to automatically learn a generalized policy with better performance than manually designed generalized policy through data.

Groshev et al. (2018) use a handcrafted translator to convert states into an image form, for example, transforming the state of the Sokoban domain into a grid-based image representation that includes the warehouse layout, the position of boxes, the initial state, and the target state. The images are then used as input to train the CNN or GNN to learn a generalized policy, predict the action to be taken in the current state and the length of the plan, and use the bootstrap method to update the training set. However, this approach depends on the visual representation of the problem, requiring a handcrafted translator designed for each domain.

ASNets (Toyer et al., 2018, 2020) mimic traditional non-learning planners through supervised learning, learning solutions from small-size problems and integrating domain-independent heuristics to improve performance, thereby obtaining a generalized policy. ASNets consist of alternating action layers and proposition layers, connected according to the action schema in the (P)PDDL. ASNets can be seen as a graph neural network architecture that represents the state as a graph, performs message passing between propositions and actions, and uses different update functions according to different action schemas. The weight-sharing scheme of this architecture theoretically allows it to be applied to any size problem in the given domain. Additionally, ASNets can be combined with Upper Confidence Bounds applied to Trees (UCT) (Shen et al., 2019) to optimize suboptimal generalized policies due to insufficient training or architectural limitations. However, ASNet's fixed receptive field limits its ability to infer over long paths.

Another state-of-the-art neural network model, GBFS-GNN (Rivlin et al., 2020), uses GNN and reinforcement learning to learn generalized policies and combines the inference results of the neural network with an improved GBFS search algorithm to enhance performance. We will introduce it in detail in Section 3.1.

ToRPIDo (Bajpai and Garg, 2018) and TraPSNet (Garg et al., 2019) learn generalized policies for probabilistic planning problems defined in RDDL. They encode the state as a graph, where nodes represent objects and edges indicate two objects corresponding to two edge nodes are connected by non-fluents. Then, they use GCN and GAT to obtain state representation in latent space. However, these models assume the domain only contains unary actions and binary non-fluents.

Compared to traditional non-learning planners, generalized policies learned from neural networks can provide higher-quality solutions with lower computational complexity. Unlike learning heuristics methods, generalized policies can directly output the action probability distribution and only require a single inference by the neural network at each state to determine the action to be executed, thus solving the planning problem more efficiently. However, most models learning generalized policies can only learn domain-dependent generalized policies and need to be trained separately for each domain.

Chapter 3

# GBFS-GNN for Domains with Low-arity Predicates

In this chapter, we will introduce the original GBFS-GNN and present its limitations in Section 3.1, and then propose improvements to address these limitations in Section 3.2.

## 3.1 Original GBFS-GNN

In common discrete action reinforcement learning problems, the set of actions is fixed. For example, in the Grid World problem, the agent can only take one of four movements: up, down, left, or right. Hence, the core approach of common reinforcement learning models involves inputting the state embedding into the MLP to obtain the value or probability distribution of each action in the fixed action set, and then selecting an action to interact with the environment. Therefore, common reinforcement learning models only require embedding the state as a single vector without needing to embed the action.

However, in classical planning problems, the action set is not fixed, and the specific actions and the number of actions in the action set under each state are different, meaning the action set is state-dependent. Taking the Blocksworld domain as an example, when the $state = \{arm\text{-}empty, on\text{-}table(block1), on(block2, block1), clear(block2)\}$, the action set includes the action $unstack(block2, block1)$; while when the $state = \{on\text{-}table(block1), holding(block2)\}$, the action set includes actions $putdown(block2)$ and $stack(block2, block1)$. Moreover, the same actions within different states should have different representations. Therefore, we cannot simply use common discrete action reinforcement learning models to solve classical planning problems.

The training and inference process of the original GBFS-GNN is illustrated in Figure 3.1 and encompass the following steps:

1. Encode the state as a graph and input it into the GNN to obtain the state embedding, including global, node, and edge embeddings.
2. Combine the effects of actions and the state embedding affected by the effects as the effect encoding and input it into the MLP to obtain the action embedding. The action embedding is state-dependent, containing both the action and state information.
3. Input the action embeddings into the reinforcement learning model to obtain the value of each action, then use softmax to normalize these action values into an action probability distribution.
4. Integrate the output of the neural network model with the heuristic search algorithm to solve the planning problems.

This section will introduce these four main steps in order and discuss the limitations with the original GBFS-GNN.

### 3.1.1 State Representation

#### State Encoding

In planning, the state is a complex data structure containing many objects and propositions, and a single vector may not fully and accurately represent its complex internal structure. Moreover, since we wish for the action representation to be state-dependent, containing both the action and state information, the state embedding should include multiple vectors associated with effects of action. Therefore, the original GBFS-GNN encodes the state as a directed complete graph with strong expressiveness, where different graph components are associated with different propositions affected by effects of action.

Specifically, the directed complete graph consists of the global attributes, nodes, and edges, each with its attributes described by a feature vector. The feature vector contains three parts: features describing the previous state, the current state, and the goal. These three parts are identical in size and used to fully describe the information of a state from its past state, current state, to its goal state, hence called the history-state-goal graph (also referred to as state graph). In each part of the state graph, the original GBFS-GNN uses the global attributes to represent the set of all unique entities in the domain, such as the arm in the Blocksworld domain, whose attributes are the domain's 0-ary predicates; nodes to represent objects in the problem, whose attributes are the domain's 1-ary predicates, and each object type is also considered a 1-ary predicate; and edges to represent relationships between objects, whose attributes are the domain's 2-ary predicates.

Each feature in the feature vector is a binary value of 0 or 1. In the history part, the current state part, and the goal part of the state graph, an feature being 1 indicates that its corresponding grounded predicate is present in the history state, current state,

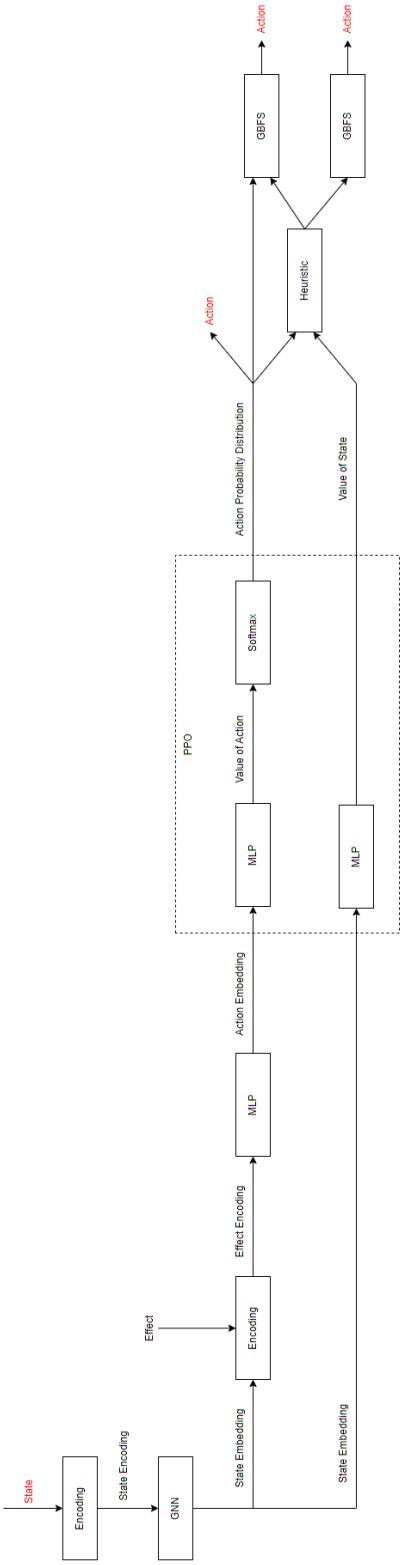


Figure 3.1: The overall training and inference process of GBFS-GNN. The red text "state" represents the input, the red text "action" represents the outputted action through different inference method.

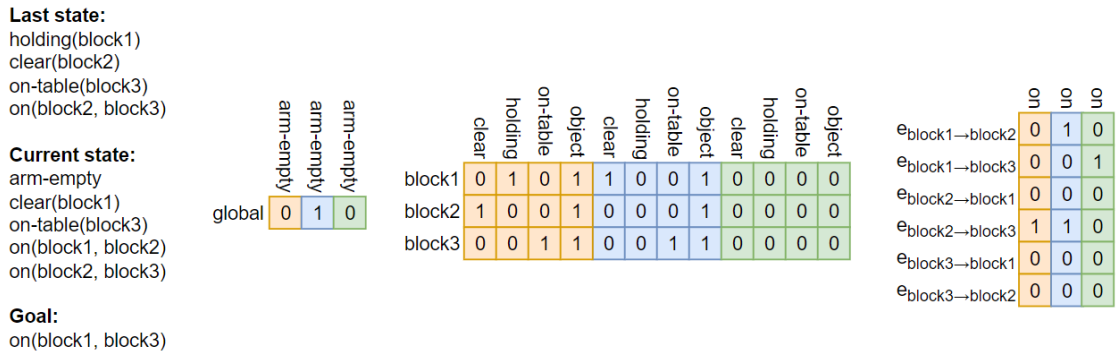


Figure 3.2: The example history-state-goal graph (state graph) of the Blocksworld domain. The left side shows the PDDL description of the history-state-goal, and the right side shows the graph representation of the same history-state-goal. Orange represents history information, blue represents state information, and green represents goal information.

or goal, respectively, and 0 indicates the opposite. An example of a state graph from the Blocksworld domain is as Figure 3.2.

### State Embedding

Due to the simplicity of the aforementioned state encoding method, its capacity for representation is limited. Furthermore, there is a lack of connectivity among global attributes, nodes, and edges, which hinders a comprehensive and accurate representation of state information. Therefore, after encoding the state into a state graph, the original GBFS-GNN employs a GNN to train it, performing message passing between the different components of the graph. This process yields the embeddings of nodes, edges, and global attributes in the latent spaces. The original GBFS-GNN utilizes two types of GNN blocks: the Graph Network block (GN block) and the Graph Network Attention block (GNAT block). These two GNN blocks share a similar methodology for message passing and update sequences:

1. The edge is updated using the original edge and the source node.
2. The node is updated using the original node, the incoming edges, and the global attributes.
3. The global attributes is updated using the original global attributes and the average of all nodes.

For domains of varying complexity, the original GBFS-GNN employs GNNs with different stacking approaches: for simpler domains, such as Blocksworld, it employs two successive GN blocks; for more complex domains, such as Satellite, it utilizes a GNAT block followed by a GN block. Moreover, the original GBFS-GNN introduces a residual connection mechanism between the two layers of GNN blocks to mitigate the vanish-

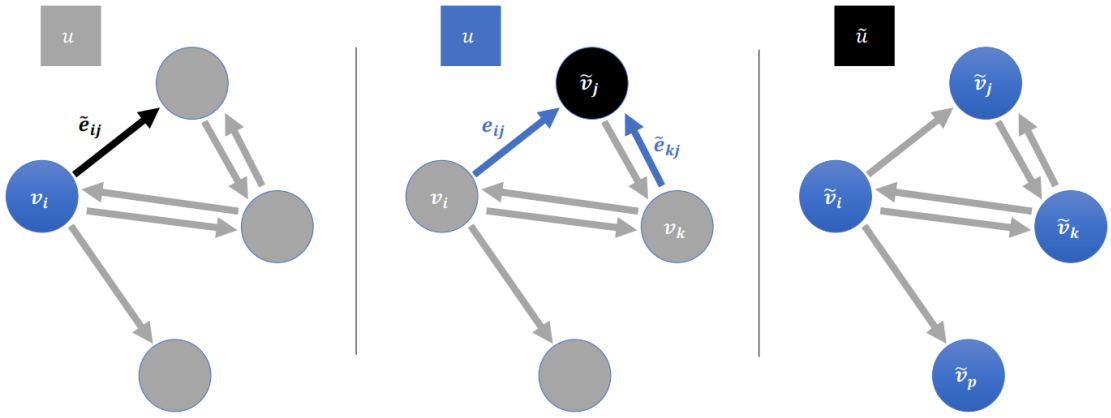


Figure 3.3: The GNN message passing process. The black elements represent the graph components updated at each step, the blue elements represent the additional graph components used to update that component, and the gray elements represent the graph components not used. On the left: updating the edges. In the middle: updating the nodes. On the right: updating the global attributes. Source: Rivlin et al. (2020)

ing gradient problem while also enabling the model to better integrate features across various levels of abstraction. Figure 3.4 illustrates these two stacking approaches.

GN block is similar to the one described in Subsection 2.2.1. Mathematically, it performs the following operations: **GN block**

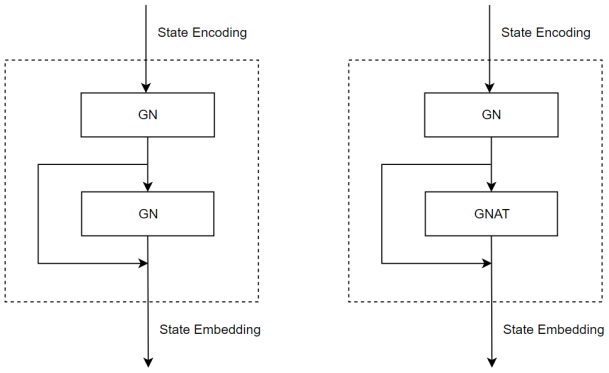


Figure 3.4: Two types of GNNs with different stacking approaches

$$
\begin{aligned}
\tilde{e}_{ij} &= \phi_e([e_{ij}, v_i]) \\
h_{ij} &= \phi_{v_1}([v_j, \tilde{e}_{ij}]) \\
m_i &= \psi(h_{ij}) \\
\tilde{v}_i &= \phi_{v_2}([m_i, u]) \\
\tilde{u} &= \phi_u(\frac{1}{|v|}\sum_{i \in v} \tilde{v}_i)
\end{aligned}
$$

where $[x, y]$ represents the concatenation operation, $\phi$ represents the MLP, and $\psi$ represents the max-pooling operation.

During the update process of a GN block, each node equally receives information from its neighbors. Although it can achieve the purpose of information transmission, it does not allow for targeted focus on key information.

**GNAT block** To address this limitation, the original GBFS-GNN proposes the GNAT block, which introduces an attention mechanism similar to that of Transformers. Mathematically, it performs the following operations:

$$
\begin{aligned}
\tilde{e}_{ij} &= \phi_e([e_{ij}, v_i]) \\
value_i &= \phi_{value}(v_i) \\
key_i &= \phi_{key}(v_i) \\
query_{ij} &= \phi_{query}(\tilde{e}_{ij}) \\
attention_{ij} &= \frac{exp(key_i \cdot query_{ij})}{\sum_{p \in v} exp(key_i \cdot query_{ip})} \\
m_i &= \varphi(attention_{ij} \odot \tilde{e}_{ij}) \\
\tilde{v}_i &= \phi_v([v_i, m_i, u]) \\
\tilde{u} &= \phi_u(\frac{1}{|v|}\sum_{i \in v} \tilde{v}_i)
\end{aligned}
$$

where $[x, y]$ represents the concatenation operation, $\phi$ represents the MLP, $\varphi$ represents the sum-pooling operation, and $\odot$ represents element-wise product operation.

During its update process, the GNAT block employs an attention mechanism that allows each node to focus on certain key information, optimizing the performance of message passing between neighboring nodes. This mechanism significantly enhances the model's expressive capability, thereby improving the performance of the GBFS-GNN.

### 3.1.2 Action Representation

At each step of the search, a successor-state generator is employed to take the current state and output a list of legal actions. Since the selection of actions is primarily based on their effects, the original GBFS-GNN uses an effect-based embedding to describe an action: encoding all effects of an action and then training them through the MLP, followed by a sum-pooling operation on the trained effect embeddings to obtain the action representation.

Specifically, an action is comprised of preconditions, add effects, and delete effects. As the legal actions are provided by the successor-state generator (i.e., actions all meet their preconditions), preconditions can be ignored, and only encoding the add and delete effects. The original GBFS-GNN uses a one-hot feature vector to represent the effect, with each feature being a ternary value of 0, -1, or 1. Each effect influences a grounded predicate in the state, hence each effect corresponds to a state graph component associated with the grounded predicate. The dimension of the effect feature vector matches that of the corresponding state graph component encoding. A feature value of 0 indicates the corresponding grounded predicate is unaffected by the effect, while 1 and -1 indicate a positive or negative impact (i.e., this effect is an add effect or del effect), respectively. To ensure the action representation is state-dependent, containing both the action and state information, the original GBFS-GNN concatenates each effect feature vector with the effect's corresponding state graph component embedding as the effect encoding to represent the impacts of this effect on the current state.

Subsequently, effect encodings are grouped based on the type of corresponding state graph component (e.g., global attributes, node, and edge) and input into three MLPs for training, yielding the effect embedding for each effect. A sum-pooling operation is then performed on all effect embeddings of an action to derive the action embedding, which includes the information of action and state, indicating the impacts of this action on the current state. Figure 3.5 illustrates the process of action embedding.

### 3.1.3 Training

The objective of GBFS-GNN is to train models using problems of a small size to obtain the generalized policy and then solve problems of a much larger size, leading to a difference between the state space of the training and testing set. Although supervised learning could train the model with the optimal solution of a problem, this approach primarily learns the inherent patterns from the data, performing well only within the state space covered by the training data and potentially failing to make optimal decisions for unseen states. Reinforcement learning, on the other hand, learns policies to solve problems through interaction with the environment, not limited by known optimal solutions, and can make optimal decisions for unseen states. This makes reinforcement learning particularly suited for planning and decision-making problems, prompting the choice to train the GBFS-GNN using reinforcement learning. The PPO model, chosen for its simplicity, broad applicability, effectiveness, and stability, serves as the foundational

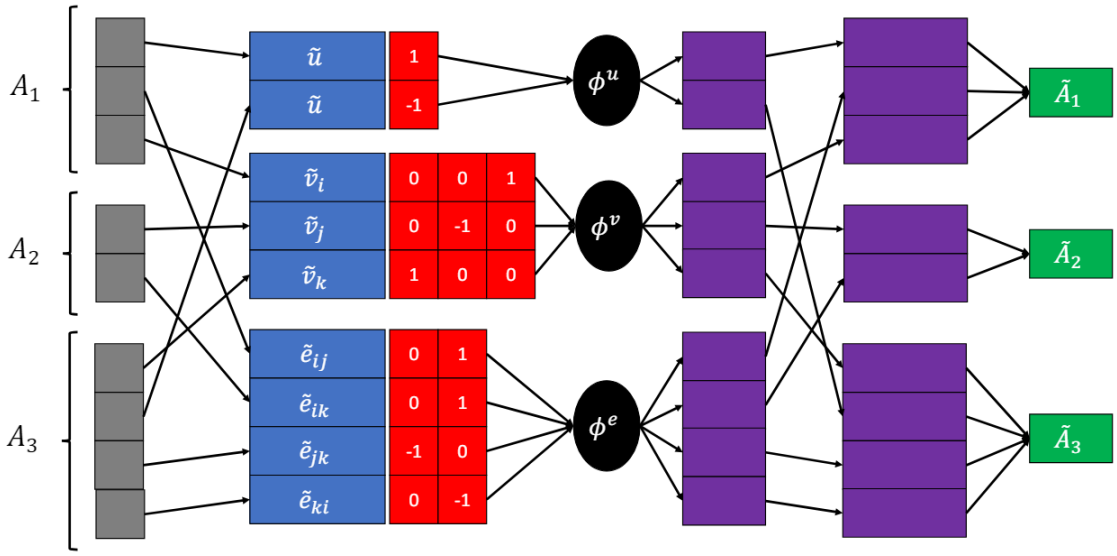


Figure 3.5: The action embedding process. From left to right: gray blocks represent action effects, blue blocks represent state graph components embeddings, red blocks represent one-hot effect feature vectors, black pieces represent MLPs, purple blocks represent effect embeddings and then scattered back to their origin actions, and green blocks represent action embeddings. Source: Rivlin et al. (2020)

reinforcement learning model for the GBFS-GNN.

Since the original GBFS-GNN focuses on finding a feasible plan to solve planning problems, it models these problems as a binary sparse reward environment. It uses five times the value of the FF heuristic for the initial state as the maximum acceptable plan length, rewarding the agent with a score of 1 if it solves the planning problem within this length; otherwise, no reward is given. The original GBFS-GNN inputs the action embeddings into the PPO's policy network (Actor) to evaluate the value of each legal action under the current state, then normalizes these action values using a softmax function to obtain an action probability distribution. At the same time, it inputs the embedding of global attributes into the PPO's value network (Critic) to acquire the current state's value. The resulting action probability distribution and state value are used for subsequent inference. Moreover, the training set includes problems whose size is small enough to be occasionally solved by a randomly initialized policy, allowing for effective feedback and learning at the initial training stage.

Several improvements to the original PPO have been implemented in the original GBFS-GNN:

1. Unlike in the original PPO, where model parameters are updated after a fixed number of rollout steps, often terminating the episode prematurely, the original GBFS-GNN performs rollout until the agent receives a reward or reaches the plan length limit, avoiding premature episode termination. This ensures the data in the rollout buffer remains complete and sequential, enhancing training effectiveness.

2. The original PPO uses Bootstrapping Value Estimation or Generalized Advantage Estimation (GAE) as the return function, whereas the original GBFS-GNN employs a modified Empirical Return function, defined as follows:

$$R_t = r_t + \gamma^{T-t+1} R_{t+1}$$

   where $R_t$ represents the cumulative discounted return at time step $t$, $r_t$ represents the reward received at time step $t$, and $\gamma$ represents the discount factor.

3. While the original PPO performs gradient descent on each batch, larger batch sizes can stabilize the training process and improve performance. Therefore, the original GBFS-GNN uses cumulative gradient descent to simulate super large batch size.

### 3.1.4 Inference

During the inference process, we can directly use the neural network for rollout. Starting from the initial state, the current state is input into the neural network to obtain the action probability distribution. Next, we sample from this distribution to select an action to interact with the environment. This process is repeated until we either obtain the reward or reach the maximum plan length. While this method is straightforward to implement, its success rate is relatively low for complex domains and large-size problems.

The original GBFS-GNN introduces a neural network-based heuristic estimation, defined as follows:

$$h(s,a) = \frac{\pi(a \mid s) \cdot V(s)}{1 + H(\pi(\cdot \mid s))}$$

where $h(s,a)$ represents the heuristic estimate for choosing action $a$ under state $s$, $\pi(a \mid s)$ is the probability of action $a$ output by the actor part of PPO, $V(s)$ is the value of state $s$ output by the critic part of PPO, and $H(\pi(\cdot \mid s))$ is the entropy of the action probability distribution. This heuristic estimation can be used with the heuristic search algorithm, such as GBFS, to solve planning problems.

Many zero-sum game solvers have significantly improved inference performance by combining search algorithms with neural network policies. For example, AlphaGo (Silver et al., 2016) combines the Monte Carlo Tree Search algorithm with a deep neural network policy to achieve remarkable results, defeating the world champion of Go. The original GBFS-GNN also uses this inference method, combining the heuristic search algorithm GBFS with a neural network policy and heuristic to decide on the next action to take. Specifically, similar to standard GBFS, each state-action pair is considered a node in the search tree, and the heuristic described above is used to navigate through the tree. Starting from the initial state node, the node with the best heuristic estimate is selected from the open list, and the neural network performs a rollout on it. If the goal state is reached within the maximum plan length, this plan is returned; otherwise, this node is expanded, and its child nodes are put into the open list. This process is repeated until the time limit is reached. Figure 3.6 illustrates this inference procedure.

### 3.1.5 Limitations of Original GBFS-GNN

The original GBFS-GNN has the following limitations:

1. The original GBFS-GNN's planning performance for large-size problems and complex domains is poor, and its training times are too long.
2. The original GBFS-GNN was trained using a small problem-size dataset, and it cannot complete training on large problem-size datasets, such as the IPC 2023 Learning Tracks dataset, within a reasonable time.
3. The original GBFS-GNN can only encode 0-ary, 1-ary, and 2-ary predicates (low-arity predicates) as global, node, and edge features in the state graph, and it is unable to solve problems in domains with higher-arity predicates.

In the next section, we address limitation 1 and 2. In Chapter 4, we address the arity limitation.

## 3.2 Improving Training and Inference Performance

As mentioned in the previous section, there is a need to optimize the training and inference performance. Therefore, we propose four optimization methods to address these

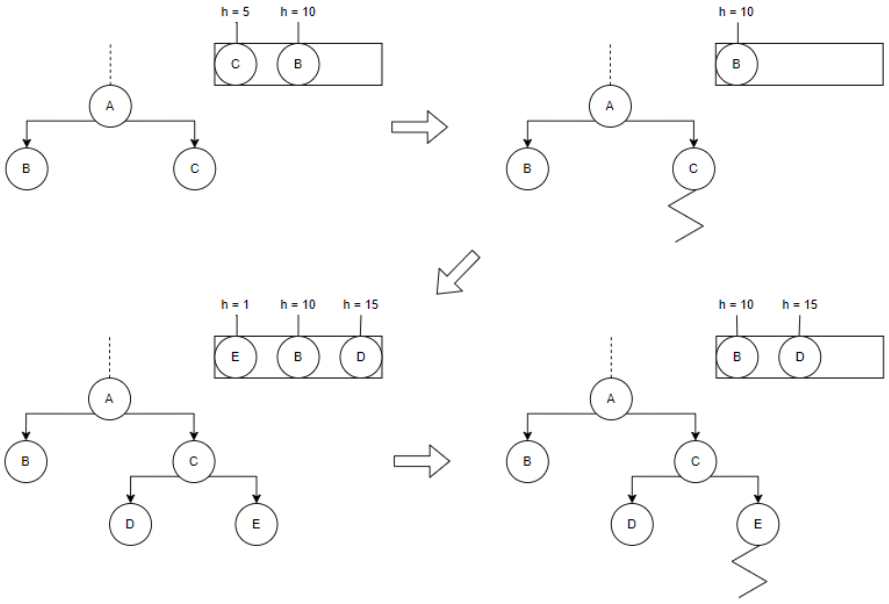


Figure 3.6: The GBFS-GNN inference procedure. Open list is represents by the nodes inside rectangle and tree represents the seatch tree. At first, there are two nodes with different heuristic estimates in the open list. We choose the node with best heuristic estimate to perform a rollout. If the goal state is not reached, this node is expanded, and its child nodes are put into the open list. Then, we repeat the above steps.

limitations: advantage normalization, selecting the most likely action, illegal edges deletion, and incremental training procedure. This section will introduce these improvements in order.

### 3.2.1 Advantage Normalization

In the PPO model, advantage estimation assesses the relative value of taking a specific action in a given state compared to the average performance of the policy. A higher advantage estimation indicates that the specific action is significantly better than the policy's average performance. The formula for calculating the advantage estimation is as follows:

$$A_t = R_t - V(s)$$

where $A_t$ represents the advantage estimation at time step $t$, $R_t$ represents the cumulative discounted return (i.e., the value of taking a specific action) at time step $t$, and $V(s)$ represents the value of state $s$ (i.e., the average performance of the policy).

However, in practical applications, advantage estimation can have a high variance due to differences in states and rewards and the stochastic nature of policies, leading to instability in training, long training times, and poor performance. To address this issue, advantage normalization can be applied to reduce its variance. The specific formula is as follows:

$$\hat{A}_t = \frac{A_t - \mu_A}{\sigma_A}$$

where $\hat{A}_t$ is the normalized advantage estimation at time step $t$, $A_t$ is the advantage estimation at time step $t$, $\mu_A$ is the mean of advantage estimation across a batch, and $\sigma_A$ is the standard deviation of advantage estimation across a batch.

By reducing the variance of advantage estimation, advantage normalization can decrease oscillations during the update steps, improving training stability. Furthermore, advantage normalization adjusts all advantage estimations to the same scale, preventing states or actions from being overemphasized or ignored due to extreme reward values. This aids in accelerating model convergence, reducing training time, and enhancing model performance.

### 3.2.2 Selecting Most Likely Action

In the inference process of the original GBFS-GNN, after obtaining the action probability distribution from the neural network, actions are chosen through sampling. This method is simple to implement and introduces randomness, providing the capability for exploration. It prevents the model from overly optimizing for states encountered during training, thereby preserving its ability to generalize to unseen states and effectively mitigating the issue of overfitting. However, this approach does not fully utilize the trained policy, and when the trained policy is close to optimal, it significantly increases the uncertainty and time required to find a solution. For instance, in a problem where any state

has 10 legal actions, and the trained policy outputs an action probability distribution where the best action has a 91% chance and all others have a 1% chance. This policy is close to optimal, but sampling has over a 50% chance of selecting a poor performance action in 7 consecutive steps.

Thus, for policies close to optimal, another method for choosing actions can be used during inference: selecting the most likely action (i.e., selecting the action with the highest probability) based on the action probability distribution obtained from the neural network. This deterministic method fully utilizes the trained policy, converging more quickly to the optimal solution and improving inference performance. Using the example above, the chance of selecting a poor performance action in 7 consecutive steps drops to 0%.

Selecting the most likely action algorithm is similar to a Depth-first search (DFS) algorithm, starting from the initial state and always choosing the highest probability action that has not yet been taken in the current state. If trapped in a loop (encountering the same state twice) or there are no unexecuted actions in the current state, it backtracks to the previous state. This process repeats until the problem is solved or the plan length limit is reached.

However, this method of choosing action also presents three limitations:

1. **Complex implementation**: This method requires identifying the highest probability action that has not been taken in the current state and employs a backtracking algorithm to avoid loops, necessitating various suitable data structures to store a large number of intermediate states.
2. **Lack of exploration**: Always choosing the optimal action leads to a lack of exploration during inference, especially when facing unknown states, which may result in missing better paths and settling for local optima.
3. **High dependency on policy performance**: When the trained policy performs poorly, it may frequently enter loops, leading to frequent backtracking and traversing a large number of useless states.

In Section 5.4, we will determine the appropriate method of choosing action.

### 3.2.3 Illegal Edges Deletion

In the original GBFS-GNN, the state is encoded as a directed complete graph, where nodes represent objects and edges denote binary relations between two objects. This encoding approach assumes the existence of binary relations among all objects. During training and inference with GNNs, messages are passed through edges between all object nodes. However, the parameters accepted by predicates are subject to type constraints, meaning that if no predicate that takes two specific types as parameters, there is no binary relation between objects of those types. For example, in the Satellite domain, there are no binary relations between objects of the “direction” type and the “mode” type.

Edges should not exist between nodes that lack a binary relation preventing messages from passing directly between them. But in the original GBFS-GNN, these unassociated nodes still engage in message passing through illegal edges in the directed complete graph, significantly increasing training and inference times and reducing inference performance.

To address this, after encoding the state as a directed complete graph, we can delete illegal edges between nodes that lack any binary relation (i.e., where no predicate exists that takes the types of the objects as parameters). This reduces training and inference times and improves model performance. Moreover, as the number of objects increases, illegal edges deletion can remove a greater number and proportion of illegal edges. For example, in an easy satellite problem containing 6 satellites, 11 instruments, 2 modes, and 7 directions, the state graph in the original GBFS-GNN contains 650 edges. After applying illegal edges deletion, the graph contains 442 edges, reducing 208 (32%) edges. For a medium satellite problem with 40 satellites, 78 instruments, 5 modes, and 30 directions, the state graph in the original GBFS-GNN contains 23,256 edges. After illegal edges deletion, the graph has only 14,400 edges, a reduction of 8,856 (38%) edges.

### 3.2.4 Incremental Training Procedure

GBFS-GNN uses reinforcement learning to train the model. Finding any plan to solve a planning problem is a sparse reward environment, where the agent receives rewards only after completely solving the problem without intermediate feedback during the planning process since action costs are ignored and only goal reachability is considered. Moreover, the initial model uses a random policy to solve the problem, and the environment is more complex for larger-size problems. These factors lead the agent to generally fail to receive rewards for large-size problems within the plan length limit in the early stage of training. As a result, it is challenging for the agent to receive effective feedback and learn on large problem-size datasets during the training. Furthermore, problems with larger sizes typically have longer plan lengths, making it difficult for the agent to learn which specific actions lead to rewards, even when rewards are occasionally granted.

Therefore, original GBFS-GNN constructs training sets with problems whose size is small enough to be occasionally solved by a randomly initialized policy to obtain sufficient rewards during the training. For example, problems containing four blocks are used as training data in the Blocksworld domain. However, the generalization target for the original GBFS-GNN is 5 - 100 blocks, while the IPC 2023 Learning Tracks dataset aims for a generalization of 5 - 500 blocks. Thus, larger problem-size training sets (e.g., 2 - 26 blocks in the Blocksworld domain) are employed in the IPC 2023 Learning Tracks dataset, where original GBFS-GNN struggles to train within a reasonable time on such a large problem-size dataset.

To overcome this limitation, we propose an incremental training procedure. First, the dataset is sorted by problem size and divided into $n$ splits. Training begins with the first split, and after $m$ epochs, the next split is merged into the current training set and

traning is resumed. This process is repeated, gradually adding splits to the training set until the entire dataset is utilized for training. The hyperparameters $n$ and $m$ can be adjusted based on the difficulty of the training set to ensure more comprehensive training on simpler data.

This approach allows the agent to rapidly learn the basic policy from small-size problems. The basic policy is then used to continue training on more complex problems, gradually learning higher-level policy until the optimal policy is finally learned using the entire dataset. By adjusting the hyperparameters $n$ and $m$, this method enables GBFS-GNN to train on large problem-size datasets within a reasonable time.

Additionally, the original GBFS-GNN trains 1000 iterations in each domain and selects the best model based on performance on the validation set. This approach can lead to wasting computational resources and training time, as training continues even after achieving optimal performance or overfitting. In our incremental training procedure, if the performance on the entire dataset does not exceed the previous best performance for $k$ consecutive validations, it means that the model performance has not been improved, and the training can be stopped at this time to save computational resources and training time. We can adjust hyperparameter $k$ to balance the expected performance and computational resources.

Chapter 4

# GBFS-GNN for Domains with Higher-arity Predicates

In the previous chapter, we introduced the original GBFS-GNN and proposed four improvement methods. However, the improved GBFS-GNN is unable to encode higher-arity predicates as nodes or edges of the state graph, which limits its ability to train and infer in domains containing higher-arity predicates.

The original GBFS-GNN obtains legal actions through a successor state generator, independent of whether higher-arity predicates are encoded. Moreover, a static predicate is not changed by any action and is used only in the precondition and not in effect. The original GBFS-GNN embeds action based on effects, therefore, the static higher-arity predicates do not affect action embeddings. Since legal actions and action embeddings are independent of whether encoding the static higher-arity predicates, for domains containing only static higher-arity predicates, like Sokoban domain, we can address the limitation of the original GBFS-GNN's inability to handle domains with higher-arity predicates by ignoring static higher-arity predicates during state encoding.

For domains containing non-static higher-arity predicates, such as Grippers and Rovers domains, there are two approaches to addressing this limitation:

1. Decompose higher-arity predicates in the domain into binary predicates, then input the decomposed domain and problem into the original GBFS-GNN for encoding, training, and inference. We will detail this solution in Section 4.1.

2. Modify the architecture of the original GBFS-GNN so it can directly encode higher-arity predicates as nodes and edges of the state graph. This solution will be explained in Section 4.2.

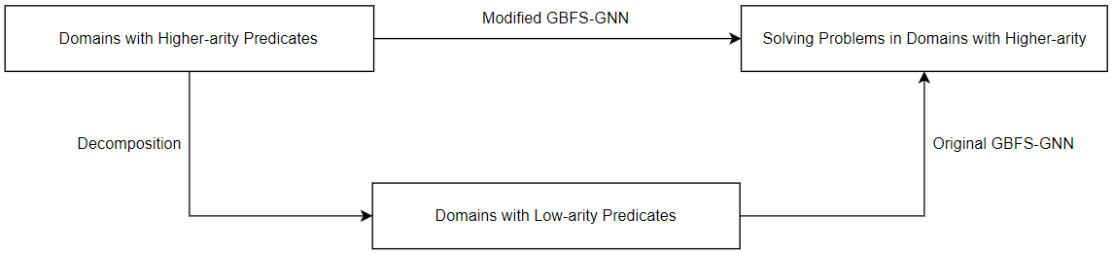


Figure 4.1: Two approaches to solving problems in domain with higher-arity predicates

## 4.1 Decomposition

We first introduce the solution of decomposing higher-arity predicates in the domain into binary predicates. There are two decomposition methods:

1. Decompose the higher-arity relationship among parameters of higher-arity predicate into multiple binary relationships by pairing parameters, i.e., decompose $p(?a, ?b, ?c)$ into $p_1(?a, ?b)$, $p_2(?a, ?c)$, and $p_3(?b, ?c)$. We refer to this method as **relational decomposition** and it will be detailed in Subsection 4.1.1.

2. Introduce an extra relation token to represent the relationship among parameters of higher-arity predicate, and decompose the higher-arity relationship into multiple binary relationships between each parameter and the relation token, i.e., decompose $p(?a, ?b, ?c)$ into $p_1(?r, ?a)$, $p_2(?r, ?b)$, and $p_3(?r, ?c)$. We refer to this method as **token-based decomposition** and it will be explained in Subsection 4.1.2.

### 4.1.1 Relational Decomposition

Higher-arity predicate $p(?a, ?b, ?c, ...)$ represents the higher-arity relationship among parameters $a, b, c, ...$, which can be decomposed into multiple binary relationships $p_1(?a, ?b)$, $p_2(?a, ?c), p_3(?b, ?c), ...$ by pairing parameters while maintaining consistency. For instance, in a traffic system, predicate $p(?c, ?s, ?d)$ denotes "car $c$ travels from source $s$ to destination $d$." This can be decomposed into $p_1(?c, ?s)$ for "car $c$ departs from source $s$", $p_2(?c, ?d)$ for "car $c$ arrives at destination $d$", and $p_3(?s, ?d)$ for "there is a route from source $s$ to destination $d$".

The specific decomposition involves:

1. Decomposing all higher-arity predicates $p(?a, ?b, ?c, ...)$ in the domain into $p_1(?a, ?b), p_2(?a, ?c), p_3(?b, ?c), ...$, i.e., generating new binary predicates by pairing all parameters of the higher-arity predicate to replace the previous higher-arity predicate.

2. Decomposing all propositions derived from higher-arity predicates, such as $p(A, B, C...)$, in the problem's initial state and goal into $p_1(A, B), p_2(A, C), p_3(B, C), ...$, i.e., generating new propositions by pairing all instances of parameters of the

higher-arity predicate to replace the previous proposition.

3. Modifying actions containing higher-arity predicates by decomposing their preconditions and effects in the same way:
   - $p(?a, ?b, ?c, ...) \rightarrow p_1(?a, ?b), p_2(?a, ?c), p_3(?b, ?c), ...$
   - $(not\ p(?a, ?b, ?c, ...)) \rightarrow (not\ p_1(?a, ?b)), (not\ p_2(?a, ?c)), (not\ p_3(?b, ?c)), ...$

Taking Rovers domain as an example, the original domain and problem:

```
(define (domain rover)
    ...
    (:types rover waypoint mode objective ...)
    (:predicates
        (can_traverse ?r - rover ?x - waypoint ?y - waypoint)
        (have_image ?r - rover ?o - objective ?m - mode)
        ...
    )
    (:action navigate
        :parameters (?x - rover ?y - waypoint ?z - waypoint)
        :precondition (and (can_traverse ?x ?y ?z) ...)
        :effect (and ...)
    )
    (:action take_image
        :parameters (?r - rover ?o - objective ?m - mode ...)
        :precondition (and ...)
        :effect (and (have_image ?r ?o ?m) ...)
    )
    ...
)
```

```
(define (problem rover-01)
    (:domain rover)
    (:objects
        rover0 - rover
        waypoint0 waypoint1 waypoint2 waypoint3 - waypoint
        ...
    )
    (:init
        (can_traverse rover0 waypoint3 waypoint0)
        ...
    )
    ...
)
```

decomposed domain and problem:

```
(define (domain rover)
    ...
    (:types rover waypoint mode objective ...)
    (:predicates
        (can_traverse_1 ?r - rover ?x - waypoint)
        (can_traverse_2 ?r - rover ?y - waypoint)
        (can_traverse_3 ?x - waypoint ?y - waypoint)
        (have_image_1 ?r - rover ?o - objective)
        (have_image_2 ?r - rover ?m - mode)
        (have_image_3 ?o - objective ?m - mode)
        ...
    )
    (:action navigate
        :parameters (?x - rover ?y - waypoint ?z - waypoint)
        :precondition (and
            (can_traverse_1 ?x ?y)
            (can_traverse_2 ?x ?z)
            (can_traverse_3 ?y ?z)
            ...
        )
        :effect (and ...)
    )
    (:action take_image
        :parameters (?r - rover ?o - objective ?m - mode ...)
        :precondition (and ...)
        :effect (and
            (have_image_1 ?r ?o)
            (have_image_2 ?r ?m)
            (have_image_3 ?o ?m)
            ...
        )
    )
    ...
)

(define (problem rover-01)
    (:domain rover)
    (:objects
        rover0 - rover
        waypoint0 waypoint1 waypoint2 waypoint3 - waypoint
        ...
    )
    (:init
        (can_traverse_1 rover0 waypoint3)
```

```
        (can_traverse_2 rover0 waypoint0)
        (can_traverse_3 waypoint3 waypoint0)
        ...
    )
    ...
)
```

The relational decomposition method allows us to convert domains containing higher-arity predicates into those with only low-arity predicates while ensuring consistency. This enables us to input the decomposed domain and problem into the original GBFS-GNN for training and inference, allowing us to solve problems in domains with higher-arity predicates without modifying the original GBFS-GNN architecture. Additionally, since the decomposition does not introduce new objects to the problem, new nodes and edges will not be introduced when encoding the state, maintaining the original complexity of the state graph.

Moreover, the number of potentially instantiated propositions after decomposition is significantly less than before decomposition. For a n-ary predicate $p(?a, ?b, ?c, ...)$ ($n \geqslant 3$), assuming the number of objects of each parameter is $m$, the number of potentially instantiated propositions would be $m^n$. If we use the relational decomposition method to decompose this n-ary predicate into $p_1(?a, ?b), p_2(?a, ?c), p_3(?b, ?c), ...$, the resulting number of potentially instantiated propositions would be $m^2 \cdot C_2^n = \frac{n(n-1)}{2} \cdot m^2$. When $m > 3$, the number of potentially instantiated propositions after decomposition is less than before decomposition, and the difference between them exponentially increases as $n$ and $m$ increase. For example, for a 3-ary predicate $p(?a, ?b, ?c)$ with 4 objects of each parameter, the number of potentially instantiated propositions is 64, while the number of potentially instantiated propositions after decomposition is 48.

However, when solving the decomposed domain and problem, the planner needs to consider the paths to each intermediate state generated by the decomposition, increasing the number of states in the search space. Considering just one n-ary predicate $p(?a, ?b, ?c, ...)$ ($n \geqslant 3$), and assuming the number of objects of each parameter is $m$, the search space after decomposition would contain $(m^2)^{C_2^n} = m^{n(n-1)}$ states, which is more than the $m^n$ states in the search space before decomposition, with the difference between them exponentially increasing as $n$ and $m$ increase. Therefore, relational decomposition expands the search space, leading to the planner potentially expanding more states, thus consuming more time and memory, and increasing the difficulty of solving the problem. For instance, for a 3-ary predicate $p(?a, ?b, ?c)$, assuming each of the parameters has 4 objects, considering only the predicate $p$, the search space after decomposition would have 4096 states, significantly more than the 64 states in the search space before decomposition.

Although relational decomposition leads to an expansion of the search space, the increase in the number of expanded states is limited, allowing for the solving of medium difficulty domains and problems within a reasonable time and memory. Table 4.1 shows the

Table 4.1: The comparison of the number of expanded states before and after relational decomposition

| | Grippers with 2 robots, 3 rooms, 4 balls | Rovers with 1 rover, 4 waypoints, 2 objectives, 1 camera |
|---|---|---|
| Original | 1665 | 1534 |
| Relational Decomposition | 1761 | 2756 |

comparison of the number of expanded states before and after decomposition in solving problems in Grippers and Rovers domains using the A* algorithm and blind heuristic in Fast Downward.

Additionally, relational decomposition results in information loss:

1. **Loss of global information**: Higher-arity predicate can describe the higher-arity relationship among objects from a global perspective. After decomposition into binary relationships, only local information between pairs of objects is retained, losing the global information among objects.

2. **Weakening of constraints**: Higher-arity predicate can precisely express complex constraints among objects. After decomposition, these constraints are simplified into weaker local constraints, leading to weakened and incomplete original constraints. This introduces a critical issue, assuming the original domain and problem:

```
(define (domain decomposition)
    ...
    (:types object)
    (:predicates (p ?a - object ?b - object ?c - object))
    (:action add
        :parameters (?a - object, ?b - object, ?c - object)
        :effect (and (p ?a ?b ?c))
    )
    (:action del
        :parameters (?a - object, ?b - object, ?c - object)
        :precondition (and (p ?a ?b ?c))
        :effect (and (not (p ?a ?b ?c)))
    )
)

(define (problem decomposition-01)
    (:domain decomposition)
```

```
    (:objects A B C D - object)
    ...
)
```

decomposed domain and problem:

```
(define (domain decomposition)
    ...
    (:types object)
    (:predicates
        (p_1 ?a - object ?b - object)
        (p_2 ?a - object ?c - object)
        (p_3 ?b - object ?c - object)
    )
    (:action add
        :parameters (?a - object, ?b - object, ?c - object)
        :effect (and
            (p_1 ?a ?b)
            (p_2 ?a ?c)
            (p_3 ?b ?c)
        )
    )
    (:action del
        :parameters (?a - object, ?b - object, ?c - object)
        :precondition (and
            (p_1 ?a ?b)
            (p_2 ?a ?c)
            (p_3 ?b ?c)
        )
        :effect (and
            (not (p_1 ?a ?b))
            (not (p_2 ?a ?c))
            (not (p_3 ?b ?c))
        )
    )
)

(define (problem decomposition-01)
    (:domain decomposition)
    (:objects A B C D - object)
    ...
)
```

The issue manifests in two cases:

1. Assume $state = \{\}$, and actions $add(A, B, C), add(A, B, D), del(A, B, C)$ are exe-

cuted in sequence:

- In the original domain, the resulting $state = \{p(A, B, D)\}$ meets the precondition $PRE(del(A, B, D))$, allowing for the further execution of action $del(A, B, D)$.
- In the decomposed domain, the resulting $state = \{p_2(A, D), p_3(B, D)\}$ does not meet the precondition $PRE(del(A, B, D))$, preventing the execution of action $del(A, B, D)$, which is inconsistent with the scenario in the original domain.

2. Assume $state = \{\}$, and actions $add(A, B, D), add(A, D, C), add(D, B, C)$ are executed in sequence:
   - In the original domain, the resulting $state = \{p(A, B, D), p(A, D, C), p(D, B, C)\}$ does not meet the precondition $PRE(del(A, B, C))$, preventing the execution of action $del(A, B, C)$.
   - In the decomposed domain, the resulting $state = \{p_1(A, B), p_2(A, D), p_3(B, D), p_1(A, D), p_2(A, C), p_3(D, C), p_1(D, B), p_2(D, C), p_3(B, C)\}$ meets the precondition $PRE(del(A, B, C))$, allowing for the further execution of action $del(A, B, C)$, which is inconsistent with the scenario in the original domain.

In these two cases, after executing the same action in the same initial state, the states obtained by using the original and decomposed domain are different. In the first case, the inability to execute certain legal actions may prevent finding an optimal solution or make a solvable problem unsolvable. In the second case, executing certain illegal actions results in a plan that is infeasible in the original domain. This indicates that relational decomposition cannot guarantee the correctness of the plans, suggesting that this decomposition method weakens the original constraints and cannot ensure the integrity. Therefore, it is necessary to explore a alternative decomposition approach to maintain the consistency and integrity at the same time.

### 4.1.2 Token-Based Decomposition

To address the issue present in the relational decomposition method, an alternative decomposition approach named "token-based decomposition" can be employed. This approach involves introducing extra relation token to represent the relationship among parameters of higher-arity predicate, and decomposing the higher-arity relationship into multiple binary relationships between each parameter and the relation token while ensuring consistency and integrity. For example, in an ecosystem, the predicate $p(?t, ?w, ?s)$ denotes "a tree $t$ depends on water source $w$ and soil $s$ for growth." By introducing a relation token $r$ to represent a specific growth environment, $p(?t, ?w, ?s)$ can be decomposed into $p_1(?r, ?t)$ indicating "under growth environment $r$, tree $t$ can grow", $p_2(?r, ?w)$ indicating "growth environment $r$ includes water source $w$", and $p_3(?r, ?s)$ indicating "growth environment $r$ includes soil $s$".

The specific decomposition involves:

- **Modify the domain**

  1. For each higher-arity predicate $p$, introduce a new object type $relation\_p$.

  2. Decompose all higher-arity predicates $p(?a - T_1, ?b - T_2, ...)$ in the domain into $p_1(?r - relation\_p, ?a - T_1), p_2(?r - relation\_p, ?b - T_2), ...$, i.e., generating new binary predicates by combining each parameter of the higher-arity predicate with a relation parameter to replace the previous higher-arity predicate.

  3. For each higher-arity predicate $p$, introduce a new unary predicate $available\_relation\_p(?r - relation\_p)$ to indicate whether a relation object is in use, ensuring that relation objects currently in use are not reused.

  4. Modify actions involving higher-arity predicates:

     a) For each higher-arity predicate $p$ in an action, add a parameter $?r - relation\_p$ to the action.

     b) Decompose each higher-arity predicate $p$ in the precondition and effect of action as above:

        - $p(?a, ?b, ...) \rightarrow p_1(?r, ?a), p_2(?r, ?b), ...$
        - $(not\ p(?a, ?b, ...)) \rightarrow (not\ p_1(?r, ?a)), (not\ p_2(?r, ?b)), ...$

     c) For higher-arity predicate $p$ in the add effect, add $(not\ available\_relation\_p(?r))$ to the effect; for those in the del effect, add $available\_relation\_p(?r)$ to the effect.

- **Modify the problem**

  1. For static higher-arity predicate $p$, generate a relation object $r\_p_n$ $(n = 1, 2, ...)$ for each proposition $p(A, B, ...)$ in the initial state. Here, $n$ is used to distinguish objects corresponding to different propositions derived from the same higher-arity predicate;

  2. For non-static higher-arity predicate $p$, generate a relation object $r\_p_n$ $(n = 1, 2, ...)$ for each potentially instantiated proposition derived from higher-arity predicate $p$;

  3. Decompose all propositions derived from higher-arity predicate, such as $p(A, B, ...)$, in the initial state and goal into $p_1(r\_p_n, A), p_2(r\_p_n, B), ...$, i.e., generating new propositions by combining each instance of parameter of the higher-arity predicate with a relation object to replace the previous proposition.

  4. Add propositions $available\_relation\_p(r\_p_{n+1}), available\_relation\_p(r\_p_{n+2}), ...$ for unused relation objects $r\_p_{n+1}, r\_p_{n+2}, ...$ in the initial state.

Taking Rovers domain as an example, the original domain and problem (where *can_traverse* is a static higher-arity predicate):

```
(define (domain rover)
    ...
    (:types rover waypoint mode objective ...)
    (:predicates
        (can_traverse ?r - rover ?x - waypoint ?y - waypoint)
        (have_image ?r - rover ?o - objective ?m - mode)
        ...
    )
    (:action navigate
        :parameters (?x - rover ?y - waypoint ?z - waypoint)
        :precondition (and (can_traverse ?x ?y ?z) ...)
        :effect (and ...)
    )
    (:action take_image
        :parameters (?r - rover ?o - objective ?m - mode ...)
        :precondition (and ...)
        :effect (and (have_image ?r ?o ?m) ...)
    )
    ...
)

(define (problem rover-01)
    (:domain rover)
    (:objects
        rover0 - rover
        waypoint0 waypoint1 waypoint2 waypoint3 - waypoint
        ...
    )
    (:init
        (can_traverse rover0 waypoint3 waypoint0)
        (can_traverse rover0 waypoint0 waypoint3)
        ...
    )
    ...
)
```

decomposed domain and problem:

```
(define (domain rover)
    ...
    (:types
        rover waypoint mode objective ...
```

```
        relation_can_traverse relation_have_image
    )
    (:predicates
        (can_traverse_1 ?r_decomposition - relation_can_traverse ?r - rover)
        (can_traverse_2 ?r_decomposition - relation_can_traverse ?x - waypoint)
        (can_traverse_3 ?r_decomposition - relation_can_traverse ?y - waypoint)
        (available_relation_can_traverse ?r_decomposition - relation_can_traverse)
        (have_image_1 ?r_decomposition - relation_have_image ?r - rover)
        (have_image_2 ?r_decomposition - relation_have_image ?o - objective)
        (have_image_3 ?r_decomposition - relation_have_image ?m - mode)
        (available_relation_have_image ?r_decomposition - relation_have_image)
        ...
    )
    (:action navigate
        :parameters (
            ?x - rover ?y - waypoint ?z - waypoint ?r_1 - relation_can_traverse
        )
        :precondition (and
            (can_traverse_1 ?r_1 ?x)
            (can_traverse_2 ?r_1 ?y)
            (can_traverse_3 ?r_1 ?z)
            ...
        )
        :effect (and ...)
    )
    (:action take_image
        :parameters (
            ?r - rover ?o - objective ?m - mode ... ?r_1 - relation_have_image
        )
        :precondition (and ... (available_relation_have_image ?r_1))
        :effect (and
           (have_image_1 ?r_1 ?r)
           (have_image_2 ?r_1 ?o)
           (have_image_3 ?r_1 ?m)
           (not (available_relation_have_image ?r_1))
            ...
        )
    )
    ...
)

(define (problem rover-01)
    (:domain rover)
    (:objects
```

```
        r_can_traverse_1 - relation_can_traverse
        r_can_traverse_3 - relation_can_traverse
        r_have_image_2 - relation_have_image
        r_have_image_4 - relation_have_image
        ...
    )
    (:init
        (can_traverse_1 r_can_traverse_1 rover0)
        (can_traverse_2 r_can_traverse_1 waypoint2)
        (can_traverse_3 r_can_traverse_1 waypoint1)

        (can_traverse_1 r_can_traverse_3 rover0)
        (can_traverse_2 r_can_traverse_3 waypoint3)
        (can_traverse_3 r_can_traverse_3 waypoint1)

        (available_relation_have_image r_have_image_2)
        (available_relation_have_image r_have_image_4)
        ...
    )
    ...
)
```

The token-based decomposition method allows us to convert domains containing higher-arity predicates into those with only low-arity predicates while ensuring consistency and completeness. This enables us to fed the decomposed domain and problem into the original GBFS-GNN for training and inference, allowing us to solve problems in domains containing higher-arity predicates without altering the original GBFS-GNN architecture. Moreover, this decomposition method introduces extra relation token to represent the relationship between parameters of the higher-arity predicate, preserving global information and the integrity of the original constraints. As a result, it prevents information loss and addresses the issue of relational decomposition that cannot guarantee the correctness of the plan. We use the same example as before, assuming the original domain and problem:

```
(define (domain decomposition)
    ...
    (:types object)
    (:predicates (p ?a - object ?b - object ?c - object))
    (:action add
        :parameters (?a - object, ?b - object, ?c - object)
        :effect (and (p ?a ?b ?c))
    )
    (:action del
        :parameters (?a - object, ?b - object, ?c - object)
```

```
        :precondition (and (p ?a ?b ?c))
        :effect (and (not (p ?a ?b ?c)))
    )
)

(define (problem decomposition-01)
    (:domain decomposition)
    (:objects A B C D - object)
    ...
)
```

decomposed domain and problem:

```
(define (domain decomposition)
    ...
    (:types object relation_p)
    (:predicates
        (p_1 ?r - relation_p ?a - object)
        (p_2 ?r - relation_p ?b - object)
        (p_3 ?r - relation_p ?c - object)
        (available_relation_p ?r - relation_p)
    )
    (:action add
        :parameters (
            ?a - object, ?b - object, ?c - object, ?r - relation_p
        )
        :precondition: (and (available_relation_p ?r))
        :effect (and
            (p_1 ?r ?a)
            (p_2 ?r ?b)
            (p_3 ?r ?c)
            (not (available_relation_p ?r))
        )
    )
    (:action del
        :parameters (
            ?a - object, ?b - object, ?c - object, ?r - relation_p
        )
        :precondition (and
            (p_1 ?r ?a)
            (p_2 ?r ?b)
            (p_3 ?r ?c)
        )
        :effect (and
            (not (p_1 ?r ?a))
```

```
            (not (p_2 ?r ?b))
            (not (p_3 ?r ?c))
            (available_relation_p ?r)
        )
    )
)

(define (problem decomposition-01)
    (:domain decomposition)
    (:objects
        A B C D - object
        r_p_1 r_p_2 r_p_3 r_p_4 - relation
        ...
    )
    (:init
        (available_relation_p r_p_1)
        (available_relation_p r_p_2)
        (available_relation_p r_p_3)
        (available_relation_p r_p_4)
        ...
    )
    ...
)
```

In the same two cases as above:

1. Assume $state = \{\}$, and actions $add(A, B, C), add(A, B, D), del(A, B, C)$ are executed in sequence:
   - In the original domain, the resulting $state = \{p(A, B, D)\}$ meets the precondition $PRE(del(A, B, D))$, allowing for the further execution of action $del(A, B, D)$.
   - In the decomposed domain, the resulting $state = \{p_1(r\_p_2, A), p_2(r\_p_2, B), p_3(r\_p_2, D), available\_relation(r\_p_1), available\_relation(r\_p_3)\}$ also satisfies the precondition $PRE(del(A, B, D))$, enabling the execution of action $del(A, B, D)$, which is consistent with the scenario in the original domain.
2. Assume $state = \{\}$, and actions $add(A, B, D), add(A, D, C), add(D, B, C)$ are executed in sequence:
   - In the original domain, the resulting $state = \{p(A, B, D), p(A, D, C), p(D, B, C)\}$ does not meet the precondition $PRE(del(A, B, C))$, preventing the execution of action $del(A, B, C)$.
   - In the decomposed domain, the resulting $state = \{p_1(r\_p_1, A), p_2(r\_p_1, B), p_3(r\_p_1, D), p_1(r\_p_2, A), p_2(r\_p_2, D), p_3(r\_p_2, C), p_1(r\_p_3, D), p_2(r\_p_3, B), \ p_3$

$(r\_p_3, C)\}$ also does not satisfy $PRE(del(A, B, C, r\_p_1))$, $PRE(del(A, B, C, r\_p_2)), ...$, preventing the execution of action $del(A, B, C)$, which is consistent with the scenario in the original domain.

In both scenarios described, executing the same actions from the same initial state results in the same state whether using the original or the decomposed domain. Hence, plans derived from domains and problems decomposed by the token-based decomposition method are also applicable to the original domains and problems. This demonstrates that the token-based decomposition method ensures the correctness of the plans, indicating that this decomposition approach does not result in information loss and maintains the consistency and integrity.

However, the token-based decomposition introduces a large number of new relation objects to the problem, leading to a substantial increase in nodes and edges when encoding the state. This greatly increases the complexity of the state graph, significantly extending training time and memory. Moreover, the number of potentially instantiated propositions after the token-based decomposition far exceeds before decomposition. For a n-ary predicate $p(?a, ?b, ?c, ...)$ $(n \geqslant 3)$, assuming the number of objects of each parameter is $m$, the number of potentially instantiated propositions would be $m^n$. If we use the token-based decomposition method to decompose this n-ary predicate into $p_1(?r, ?a), p_2(?r, ?b), p_3(?r, ?c), ..., available\_relation\_p(?r)$, the resulting number of potentially instantiated propositions would be $m^n \cdot m \cdot n + m^n = (mn + 1) \cdot m^n$, which is significantly higher than the count before decomposition, and the difference increases exponentially with $n$ and $m$. For example, for a 3-ary predicate $p(?a, ?b, ?c)$ with 4 objects of each parameter, the number of potentially instantiated propositions is 64, while the number of potentially instantiated propositions after decomposition is 624.

Additionally, solving the decomposed domain and problem requires the planner to consider paths to each intermediate state generated by the decomposition, which increases the number of states in the search space. Considering only one n-ary predicate $p(?a, ?b, ?c, ...)$ $(n \geqslant 3)$ and ignoring newly generated predicates $available\_relation\_p$, assuming each parameter $a, b, c, ...$ has $m$ objects, the search space after token-based decomposition contains $(m^n \cdot m)^n = m^{n(n+1)}$ states far more than the $m^n$ states before decomposition and the $m^{n(n-1)}$ states after relational decomposition, with the difference exponentially growing with $n$ and $m$. For instance, for a 3-ary predicate $p(?a, ?b, ?c)$ with each parameter having 4 objects, considering only the predicate $p$ and ignoring newly generated predicates $available\_relation\_p$, the search space after token-based decomposition contains 16777216 states, significantly more than the 64 states before decomposition and the 4096 states after relational decomposition.

Thus, compared to relational decomposition, token-based decomposition extremely expands the search space, requiring the planner to potentially explore far more states, greatly increasing the difficulty of problem-solving. Solving easy domains and problems within a reasonable time and memory becomes unfeasible after token-based decomposition. Table 4.2 shows the comparison of the number of expanded states before decom-

Table 4.2: The comparison of the number of expanded states before decomposition, after relational decomposition, and after token-based decomposition

| | Grippers with 2 robots, 3 rooms, 4 balls | Rovers with 1 rover, 4 waypoints, 2 objectives, 1 camera |
|---|---|---|
| Original | 1665 | 1534 |
| Relational Decomposition | 1761 | 2756 |
| Token-Based Decomposition | 10755387 | 281637 |

position, after relational decomposition, and after token-based decomposition in solving problems in Grippers and Rovers domains using the A* algorithm and blind heuristic in Fast Downward.

### 4.1.3 Summary

When using the relational decomposition method, the search space does not expand significantly, hence not considerably increasing the difficulty of solving the problem. Moreover, since no relation nodes and edges are introduced during decomposition, the complexity of the state graph does not increase. Therefore, for certain medium difficulty domains and problems, it remains feasible to find the solution within reasonable time and memory after decomposition. However, due to global information loss and weakened constraints during decomposition, the correctness of the plans obtained in some cases cannot be ensured, making this approach unfeasible for general domains.

On the other hand, since the token-based decomposition method prevents the global information loss and maintains the integrity of the original constraints, it ensures the correctness of the plans. In theory, this decomposition approach is feasible. However, this decomposition method significantly expands the search space, greatly increasing the difficulty of problem-solving. Furthermore, the introduction of numerous relation nodes and edges during decomposition considerably increases the complexity of the state graph. Consequently, in practice, solving easy domains and problems within reasonable time and memory becomes infeasible.

In conclusion, both decomposition methods require additional decomposition preprocessing steps and face serious issues, making such method unable to perfectly address the problem that the original GBFS-GNN cannot handle domains with higher-arity predicates.

## 4.2 Architecture Modification

Besides the decomposition method mentioned previously, another approach to addressing the limitation of GBFS-GNN's inability to handle higher-arity predicates is to modify the architecture of GBFS-GNN. This modification enables directly encoding higher-arity predicates as graph components, eliminating the need to convert domains with higher-arity predicates, allowing them to be directly inputted into GBFS-GNN for training and inference. This approach can reduce preprocessing steps, significantly enhancing the overall pipeline efficiency. Moreover, through reasonable optimization of the architecture, it is possible to ensure the correctness of the plans without increasing the search space and state graph complexity, allowing the GBFS-GNN to train and infer on domains with higher-arity predicates within reasonable time and memory, thereby addressing the issues of the decomposition method. An intuitive idea is to simulate the two decomposition methods mentioned above to encode higher-arity predicates as graph components. These two modification methods will be introduced in Sections 4.2.1 and 4.2.2, respectively.

### 4.2.1 Simulating Relational Decomposition

We can simulate the relational decomposition method discussed previously to modify the state and effect encoding method within GBFS-GNN, considering the higher-arity predicate $p(?a, ?b, ?c, ...)$ and proposition $p(A, B, C, ...)$ as $p(?a, ?b), p(?a, ?c), p(?b, ?c), ...$ and $p(A, B), p(A, C), p(B, C), ...$, respectively, to encode the state and effect. The specific modifications are as follows:

- **Modification of State Encoding Method**

  1. **Modification of edge**: Modify the illegal edges deletion method to ensure it does not remove edges between nodes of higher-arity predicate parameter types. For example, for a predicate $p(?a - T_1, ?b - T_2, ?c - T_3)$, edges between nodes of types $T_1, T_2, T_3$ are considered legal edges and should not be deleted by the illegal edges deletion method.

  2. **Modification of edge feature**

     a) For each n-ary predicate $p(?a, ?b, ?c, ...)$ $(n \geqslant 3)$, introduce new edge features $p_1, p_2, ..., p_m$ $(m = C_2^n)$. These newly added edge features correspond to the binary predicates $p_1(?a, ?b), p_2(?a, ?c), p_3(?b, ?c), ...$ decomposed from the higher-arity predicate $p(?a, ?b, ?c, ...)$ in the relational decomposition method.

     b) Simulate the decomposition of the proposition $p(A, B, C, ...)$ into $p_1(A, B), p_2(A, C), p_3(B, C), ...$ in the relational decomposition method, setting the feature $p_1$ of the edge $e_{AB}$, the feature $p_2$ of the edge $e_{AC}$, the feature $p_3$ of the edge $e_{BC}$, ..., to 1, with all others set to 0.

  3. All other aspects of the state encoding method remain the same as in the original GBFS-GNN.

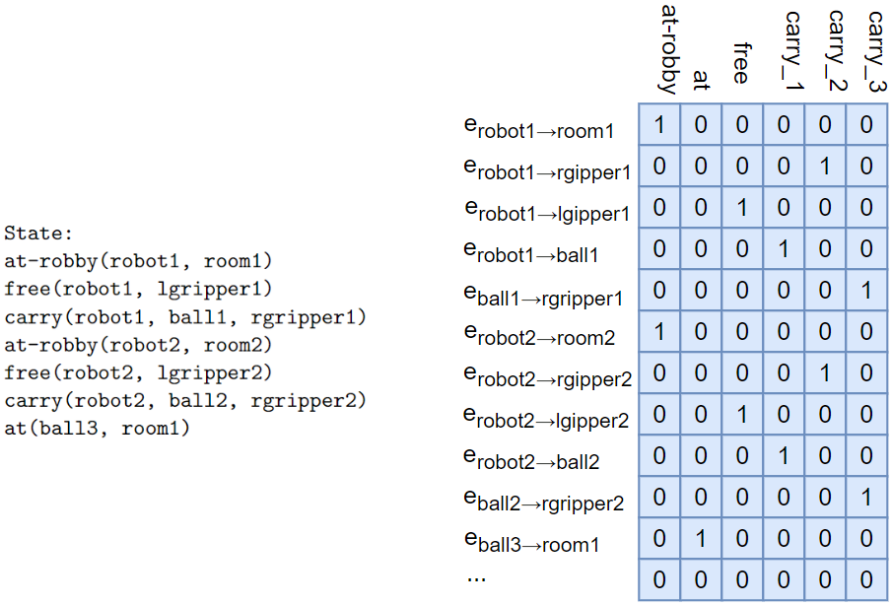


Figure 4.2: The example of part of the state graph of the Grippers domain. The left side shows the PDDL description of the state, and the right side shows the edge encodings of the same state

Taking Grippers domain as an example, the state encoding is illustrated in Figure 4.2.

- **Modification of Effect Encoding Method**

  For each n-ary effect $p(?a, ?b, ?c, ...)$ $(n \geqslant 3)$ in the action, it can be treated as $C_2^n$ binary effects $p(?a, ?b), p(?a, ?c), p(?b, ?c), ...$ for encoding, with the same effect encoding method as in the original GBFS-GNN. Taking the action $pick(robot1, ball1, room1, rgripper1)$ in the Grippers domain as an example, the effect encoding is illustrated in Figure 4.3.

This architecture modification method enables directly encoding higher-arity predicates as graph components, eliminating the need to convert domains with higher-arity predicates, allowing them to be directly inputted into GBFS-GNN for training and inference. This method has several advantages:

1. It reduces the preprocessing steps, significantly improving the efficiency of the overall pipeline.

2. It does not introduce any extra nodes and edges, keeping the complexity of the state graph unchanged, thus allowing GBFS-GNN to train and infer on domains with higher-arity predicates within reasonable time and memory.

3. Since it does not change the original domain and problem, it does not increase the

```
(:action pick
    :parameters (?r - robot ?obj - ball ?room - room ?g - gripper)
    :precondition (and
        (at ?obj ?room) (at-robby ?r ?room) (free ?r ?g)
    )
    :effect (and
        (carry ?r ?obj ?g) (not (at ?obj ?room)) (not (free ?r ?g))
    )
)
```

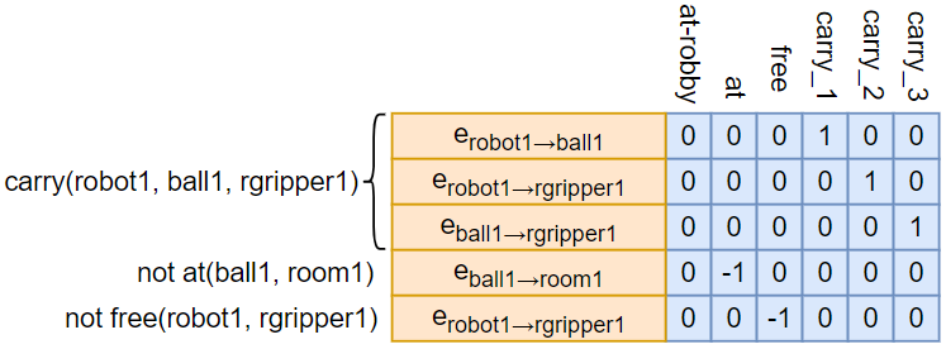


Figure 4.3: The effect encodings of the action $pick(robot1, ball1, room1, rgripper1)$. Orange blocks represent state embeddings and blue blocks represent one-hot effect feature vectors.

search space and the difficulty of solving the problem.

4. Since in GBFS-GNN, legal actions and whether goal state is achieved are determined by the successor state generator based on the original domain and state, and only the policy for selecting the action is derived from the neural network, this ensures the correctness of the resulting plan.

However, this modification method suffers from the same potential issue as relational decomposition: during state encoding and effect encoding, it implicitly decomposes a higher-arity relation into multiple binary relations. This implicit decomposition results in the model learning only local information from a set of binary relations during training, without capturing the global information inherent in a higher-arity relation. Consequently, it fails to fully and accurately represent the information within the higher-arity predicates and their groundings in states and effect.

### 4.2.2 Simulating Token-Based Decomposition

#### Introducing Relation Node for Proposition

We can also simulate the token-based decomposition method to modify the state and effect encoding method within GBFS-GNN, considering the higher-arity predicate $p(?a, ?b, ...)$ and proposition $p(A, B, ...)$ as $p(?r, ?a), p(?r, ?b), ...$ and $p(r\_p, A), p(r\_p, B), ...$, respectively, to encode the state and effect. The specific modifications are as follows:

- **Modification of State Encoding Method**

  1. **Modification of node**: Generate a node $v_{r\_p_n}$ $(n = 1, 2, ...)$ called relation node or hub node for each potentially instantiated proposition derived from higher-arity predicate $p$. These generated relation nodes correspond to the relation objects $r\_p_n$ created in the token-based decomposition method. Here, $n$ is used to distinguish nodes or objects corresponding to different propositions derived from the same higher-arity predicate.

  2. **Modification of node feature**: Introduce a new node feature $relation\_p$ for each higher-arity predicate $p$. These newly added node features correspond to the new object type $relation\_p$ introduced in the token-based decomposition method. If a node corresponds to a relation object and the type of this relation object is $relation\_p$, then set the node's feature $relation\_p$ to 1, otherwise to 0.

  3. **Modification of edge**: Modify the illegal edges deletion method to prevent the deletion of edges between relation nodes and nodes of higher-arity predicate parameter types. For example, edges between all relation nodes introduced for the predicate $p(?a - T_1, ?b - T_2, ?c - T_3)$ and nodes of types $T_1, T_2, T_3$ are considered legal edges and should not be deleted by the illegal edges deletion method.

  4. **Modification of edge feature**

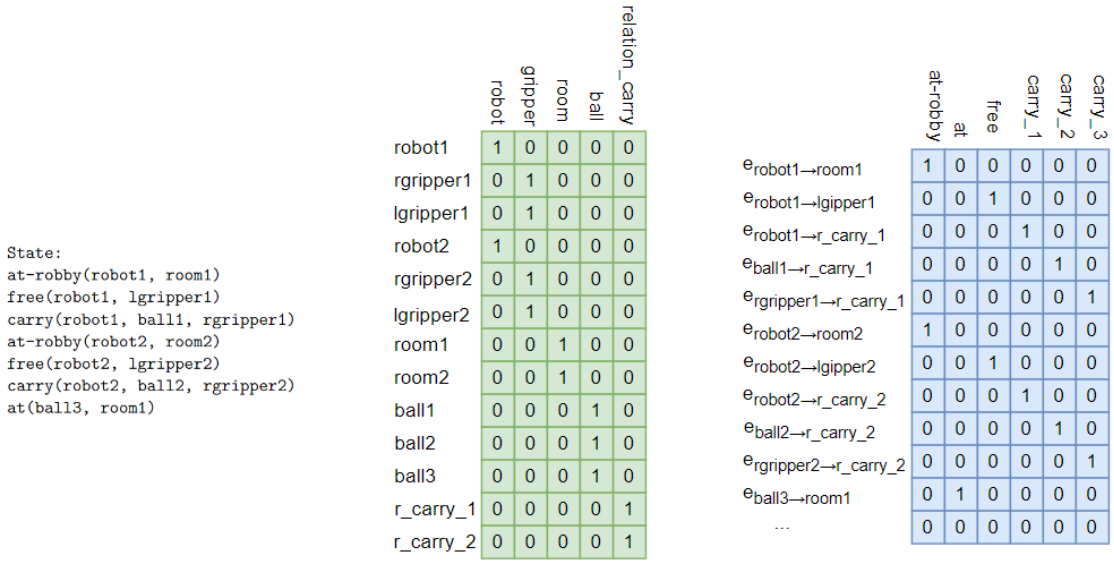


Figure 4.4: The example of part of the state graph of the Grippers domain. The left side shows the PDDL description of the state, and the right side shows the nodes and edges encodings of the same state

a) For each n-ary predicate $p(?a, ?b, ...)$ $(n \geqslant 3)$, introduce new edge features $p_1, p_2, ..., p_n$. These newly added edge features correspond to the binary predicates $p_1(?r, ?a), p_2(?r, ?b), ...$ decomposed from the higher-arity predicate $p(?a, ?b, ...)$ in the token-based decomposition method.

b) Simulate the decomposition of the proposition $p(A, B, ...)$ into $p_1(r\_p, A), p_2(r\_p, B), ...$ in the token-based decomposition method, setting the feature $p_1$ of the relation edge $e_{A,r\_p}$, the feature $p_2$ of the relation edge $e_{B,r\_p}$, ..., to 1, with all others set to 0.

5. Since the GBFS-GNN uses a successor state generator to obtain legal actions from the original domain and state, it is not necessary to simulate the predicate *available_relation_p* and its groundings introduced in the token-based decomposition method.

6. All other aspects of the state encoding method remain the same as in the original GBFS-GNN.

Taking Grippers domain as an example, the state encoding is illustrated in Figure 4.4.

- **Modification of Effect Encoding Method**

For each n-ary effect $p(?a, ?b, ?c, ...)$ $(n \geqslant 3)$ in the action, it can be treated as $n$ binary effects $p(?r, ?a), p(?r, ?b), ...$ for encoding, with the same effect encoding

```
(:action pick
    :parameters (?r - robot ?obj - ball ?room - room ?g - gripper)
    :precondition (and
        (at ?obj ?room) (at-robby ?r ?room) (free ?r ?g)
    )
    :effect (and
        (carry ?r ?obj ?g) (not (at ?obj ?room)) (not (free ?r ?g))
    )
)
```

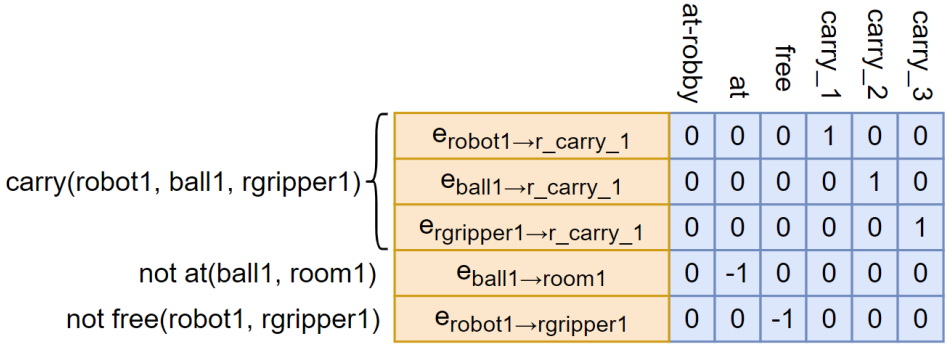


Figure 4.5: The effect encodings of the action $pick(robot1, ball1, room1, rgripper1)$. Orange blocks represent state embeddings and blue blocks represent one-hot effect feature vectors

method as in the original GBFS-GNN. Taking the action $pick(robot1, ball1, room1, rgripper1)$ in the Grippers domain as an example, the effect encoding is illustrated in Figure 4.5.

This architecture modification method enables directly encoding higher-arity predicates as graph components, eliminating the need to convert domains with higher-arity predicates, allowing them to be directly inputted into GBFS-GNN for training and inference. This method has several advantages:

1. It reduces the preprocessing steps, significantly improving the efficiency of the overall pipeline.
2. Since it does not change the original domain and problem, it does not increase the search space and the difficulty of solving the problem.
3. Since in GBFS-GNN, legal actions and whether goal state is achieved are determined by the successor state generator based on the original domain and state, and only the policy for selecting the action is derived from the neural network, this ensures the correctness of the resulting plan.
4. It has a similar advantage to the token-based decomposition method: the model can learn global information between objects of parameters of the higher-arity

predicate during training through the introduced relation nodes, enabling a more comprehensive and accurate expression of the information of higher-arity predicates and their groundings.

However, this modification approach introduces a significant number of relation nodes and edges during state encoding. For example, under the use of illegal edges deletion, a simple Grippers problem including two robots and four balls would require the addition of 32 relation nodes and 320 relation edges. More complex domains and problems might necessitate adding hundreds of relation nodes and thousands of relation edges. This could result in a state graph that is too complex for training and inference on domains with higher-arity predicates within a reasonable time and memory. Therefore, it is necessary to optimize this modification method to reduce the number of added relation nodes and edges.

### Introducing Relation Node for Predicate

The primary issue with the modification approach mentioned above is the generation of a relation node for each potentially instantiated proposition derived from higher-arity predicate, resulting in an overly complex state graph. In fact, the generation of a relation object for each potentially instantiated proposition in the token-based decomposition method is to ensure legal actions remaining consistent before and after decomposition.

For example, consider a scenario within the grippers domain where state = {carry(robot1, ball1, rgripper1), carry(robot2, ball2, rgripper2), at-robby(robot1, room1), ...}. In the original domain, the action drop(robot1, ball2, room1, rgripper1) cannot be executed since its precondition carry(robot1, ball2, rgripper1) is not satisfied. Introducing a single relation object r_carry results in a decomposition state = {$carry_1$(r_carry, robot1), $carry_2$(r_carry, ball1), $carry_3$(r_carry, rgripper1), $carry_1$(r_carry, robot2), $carry_2$(r_carry, ball2), $carry_3$(r_carry, rgripper2)}, which allows the execution of the action drop(robot1, ball2, room1, rgripper1, r_carry) since its precondition {$carry_1$(r_carry, robot1), $carry_2$( r_carry, ball2), $carry_3$(r_carry, rgripper1), at-robby(robot1, room1)} is satisfied, inconsistent with the original domain. Conversely, introducing two distinct relation objects $r\_carry_1$ and $r\_carry_2$ for the two propositions derived from higher-arity predicate within the state leads to a decomposition state = {$carry_1$($r\_carry_1$, robot1), $carry_2$($r\_carry_1$, ball1), $carry_3$($r\_carry_1$, rgripper1), $carry_1$($r\_carry_2$, robot2), $carry_2$($r\_carry_2$, ball2), $carry_3$($r\_carry_2$, rgripper2)}, which prevents the execution of the action drop(robot1, ball2, room1, rgripper1, $r\_carry_1$) or drop(robot1, ball2, room1, rgripper1, $r\_carry_2$) since their preconditions are not satisfied, consistent with the original domain.

If we can reduce the number of introduced relation nodes and edges while ensuring that the legal actions and policy for selecting the action remain consistent before and after optimising the architecture modification, it would be possible to solve complex domains and large-size problems within a reasonable time and memory. In fact, generating a relation node for each higher-arity predicate instead of one for each potentially instantiated proposition and connecting it with nodes of higher-arity predicate parameter types can

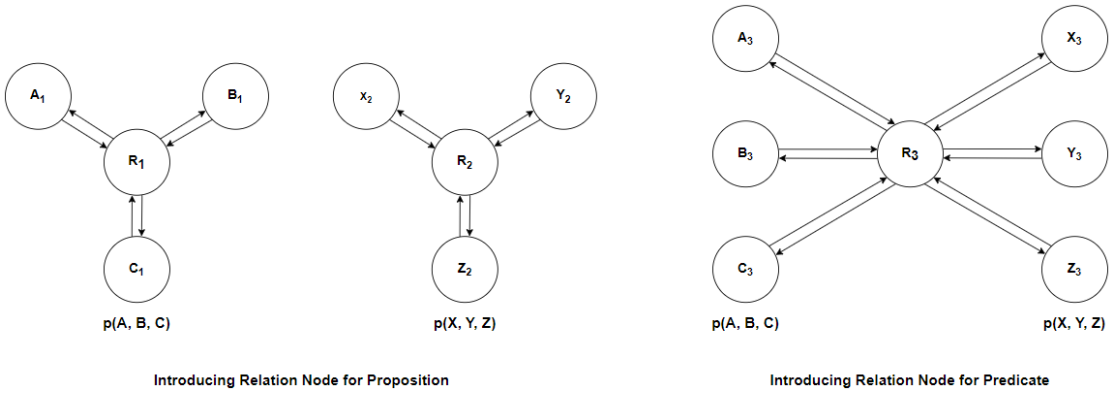


Figure 4.6: The simplified state graphs obtained using two different methods of simulating the token-based decomposition

achieve the desired outcome.

Firstly, in the GBFS-GNN, legal actions are obtained using a successor state generator based on the original domain and state, with the policy for selecting the action derived from the neural network. Therefore, as long as the state remains the same, the correct legal actions can be determined regardless of how to encode the state.

Secondly, GBFS-GNN concatenates effect feature vector and state embedding as the effect encoding, which is then input into the neural network for training to obtain the effect embedding. Subsequently, the sum-pooling operation of the embeddings of each action's effects is performed to obtain the action embedding. Suppose the domain contains the predicate $p(?a, ?b, ?c)$ and action $del(?a, ?b, ?c), PRE(del) = \{p(?a, ?b, ?c)\}, EFF^{+}(del) = \{\}, EFF^{-}(del) = \{p(?a, ?b, ?c)\}$, and $state = \{p(A, B, C), p(X, Y, Z)\}$. The simplified state graphs obtained using the modification method of introducing a relation node for proposition and predicate are as Figure 4.6. In both of these two modification methods, the initial encoding for the introduced relation nodes and edges is identical:

$$
\begin{aligned}
v_{A_1} &= v_{A_3} \\
v_{R_1} &= v_{R_3} \\
e_{R_1 A_1} &= e_{R_3 A_3} \\
e_{A_1 R_1} &= e_{A_3 R_3}
\end{aligned}
$$

Taking the two-layer GN block in GBFS-GNN as an example, in the first GN block layer:

1. According to the edge update method $\tilde{e}_{ij} = \phi([e_{ij}, v_i])$, it can be derived that
$$\tilde{e}_{R_1A_1} = \phi([e_{R_1A_1}, v_{R_1}]) = \phi([e_{R_3A_3}, v_{R_3}]) = \tilde{e}_{R_3A_3}$$
$$\tilde{e}_{A_1R_1} = \phi([e_{A_1R_1}, v_{R_1}]) = \phi([e_{A_3R_3}, v_{R_3}]) = \tilde{e}_{A_3R_3}$$

2. According to the node update method $h_{ij} = \phi([v_j, \tilde{e}_{ij}])$, $m_i = \psi(h_{ij})$, $\tilde{v}_i = \phi([m_i, u])$, it can be derived that
$$h_{R_1A_1} = \phi([v_{A_1}, \tilde{e}_{R_1A_1}]) = \phi([v_{A_3}, \tilde{e}_{R_3A_3}]) = h_{R_3A_3}$$
$$m_{A_1} = \psi(h_{R_1A_1}) = \psi(h_{R_3A_3}) = m_{A_3}$$
$$\tilde{v}_{A_1} = \phi([m_{A_1}, u]) = \phi([m_{A_3}, u]) = \tilde{v}_{A_3}$$

In the second GN block layer, according to the edge update method $\tilde{e}_{ij} = \phi([e_{ij}, v_i])$, it can be derived that

$$\hat{e}_{A_1R_1} = \phi([\tilde{e}_{A_1R_1}, \tilde{v}_{A_1}]) = \phi([\tilde{e}_{A_3R_3}, \tilde{v}_{A_3}]) = \hat{e}_{A_3R_3}$$

where $\hat{e}_{A_1R_1}$ and $\hat{e}_{A_3R_3}$ are the final embeddings of edges $A_1R_1$ and $A_3R_3$ obtained through GNN training.

Similarly, it can be derived that

$$\hat{e}_{B_1R_1} = \hat{e}_{B_3R_3}$$
$$\hat{e}_{C_1R_1} = \hat{e}_{C_3R_3}$$
$$\hat{e}_{X_2R_2} = \hat{e}_{X_3R_3}$$
$$\hat{e}_{Y_2R_2} = \hat{e}_{Y_3R_3}$$
$$\hat{e}_{Z_2R_2} = \hat{e}_{Z_3R_3}$$

Therefore, both modification methods get the same action embedding and policy:

$$\begin{aligned}
&embedding(del(A_1, B_1, C_1)) = sum(\\
&\quad \phi_e([\hat{e}_{A_1R_1}, ..., -1, 0, 0]),\\
&\quad \phi_e([\hat{e}_{B_1R_1}, ..., 0, -1, 0]),\\
&\quad \phi_e([\hat{e}_{C_1R_1}, ..., 0, 0, -1]),\\
&)
\end{aligned}
\qquad
\begin{aligned}
&embedding(del(A_3, B_3, C_3)) = sum(\\
&\quad \phi_e([\hat{e}_{A_3R_3}, ..., -1, 0, 0]),\\
&\quad \phi_e([\hat{e}_{B_3R_3}, ..., 0, -1, 0]),\\
&\quad \phi_e([\hat{e}_{C_3R_3}, ..., 0, 0, -1]),\\
&)
\end{aligned}$$

$$embedding(del(A_1, B_1, C_1)) = embedding(del(A_3, B_3, C_3))$$

$$\begin{aligned}
&embedding(del(X_2, Y_2, Z_2)) = sum(\\
&\quad \phi_e([\hat{e}_{X_2R_2}, ..., -1, 0, 0]),\\
&\quad \phi_e([\hat{e}_{Y_2R_2}, ..., 0, -1, 0]),\\
&\quad \phi_e([\hat{e}_{Z_2R_2}, ..., 0, 0, -1]),\\
&)
\end{aligned}
\qquad
\begin{aligned}
&embedding(del(X_3, Y_3, Z_3)) = sum(\\
&\quad \phi_e([\hat{e}_{X_3R_3}, ..., -1, 0, 0]),\\
&\quad \phi_e([\hat{e}_{Y_3R_3}, ..., 0, -1, 0]),\\
&\quad \phi_e([\hat{e}_{Z_3R_3}, ..., 0, 0, -1]),\\
&)
\end{aligned}$$

$$embedding(del(X_2, Y_2, Z_2)) = embedding(del(X_3, Y_3, Z_3))$$

$$\begin{aligned}
&policy_{introducing_relation_node_for_proposition} = softmax( \\
&\quad \phi(embedding(del(A_1, Y_1, Z_1))), \\
&\quad \phi(embedding(del(X_2, Y_2, Z_2))), \\
&\quad ... \\
&) \\
&policy_{introducing_relation_node_for_predicate} = softmax( \\
&\quad \phi(embedding(del(A_3, Y_3, Z_3))), \\
&\quad \phi(embedding(del(X_3, Y_3, Z_3))), \\
&\quad ... \\
&)
\end{aligned}$$

$$policy_{introducing\_relation\_node\_for\_proposition} = policy_{introducing\_relation\_node\_for\_predicate}$$

Therefore, by generating a relation node for each higher-arity predicate and connecting it with nodes of higher-arity predicate parameters type, we can significantly reduce the number of introduced relation nodes and edges while ensuring that the legal actions and policy for selecting the action derived from both modification methods remain consistent. Utilizing the same Grippers example as before, the state encoding is illustrated in Figure 4.7. Under the use of illegal edges deletion, the approach of introducing a relation node for predicate necessitates the addition of only 1 relation node and 10 relation edges. This is substantially less than the 32 relation nodes and 320 relation edges required when introducing a relation node for proposition.

An additional point of clarification is the choice to use the embedding of $e_{other\_node \rightarrow relation\_node}$ rather than the embedding of $e_{relation\_node \rightarrow other\_node}$ concatenated with effect feature vector as the effect encoding. Continuing with the previous example, suppose the domain contains the predicate $p(?a, ?b, ?c)$ and action $del(?a, ?b, ?c), PRE(del) = \{p(?a, ?b, ?c)\}$, $EFF^+(del) = \{\}, EFF^-(del) = \{p(?a, ?b, ?c)\}$, and $state = \{p(A, B, C), p(X, Y, Z)\}$. If the embedding of $e_{relation\_node \rightarrow other\_node}$ is used, the embeddings for actions $del(A, B, C)$ and $del(X, Y, Z)$ would be:

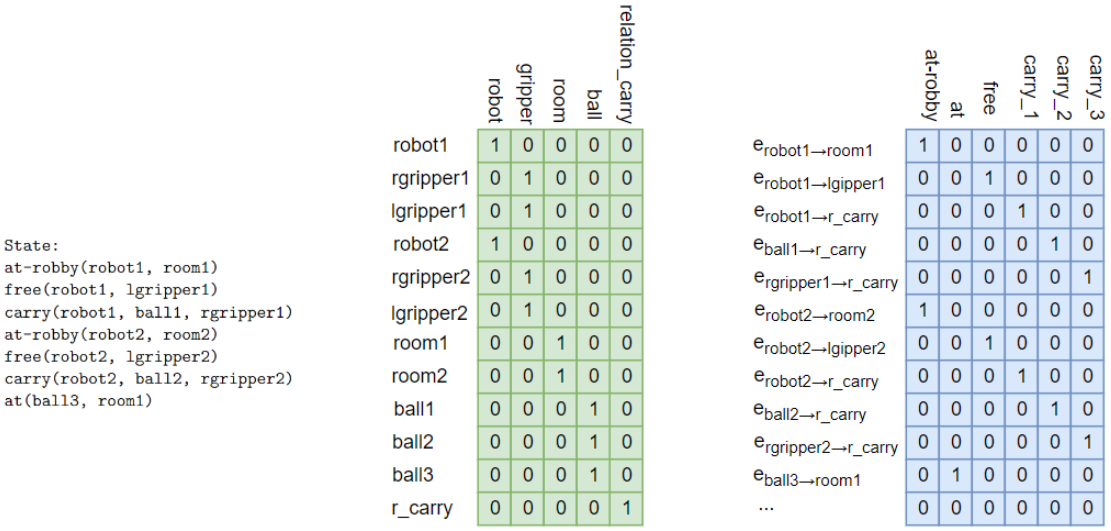


Figure 4.7: The example of part of the state graph of the Grippers domain. The left side shows the PDDL description of the state, and the right side shows the nodes and edges encodings of the same state.

$$
\begin{aligned}
&embedding(del(A,B,C)) = sum(\\
&\quad \phi_e([\hat{e}_{RA}, 0, ..., -1, 0, 0]),\\
&\quad \phi_e([\hat{e}_{RB}, 0, ..., 0, -1, 0]),\\
&\quad \phi_e([\hat{e}_{RC}, 0, ..., 0, 0, -1])\\
&)\\
&embedding(del(X,Y,Z)) = sum(\\
&\quad \phi_e([\hat{e}_{RX}, 0, ..., -1, 0, 0]),\\
&\quad \phi_e([\hat{e}_{RY}, 0, ..., 0, -1, 0]),\\
&\quad \phi_e([\hat{e}_{RZ}, 0, ..., 0, 0, -1])\\
&)
\end{aligned}
$$

During message passing in GNN, the edge update method is $\tilde{e}_{ij} = \phi([e_{ij}, v_i])$. As a result, even if the encodings of $v_A, v_B, v_C, v_X, v_Y, v_Z$ are distinct, the updated edges $\hat{e}_{RA}, \hat{e}_{RB}, \hat{e}_{RC}, \hat{e}_{RX}, \hat{e}_{RY}, \hat{e}_{RZ}$ become identical. Consequently, $embedding(del(A,B,C))$ $= embedding(del(X,Y,Z))$. This means that even though the actions $del(A,B,C)$ and $del(X,Y,Z)$ are performed on different objects, they would have identical embeddings, making it impossible for the neural network to differentiate between these two actions. This issue does not arise when using the embedding of $e_{other\_node \to relation\_node}$.

The approach of introducing relation node for predicate and proposition shares the same advantages, but only introduces a minimal number of relation nodes and edges,

maintaining the complexity of the state graph almost unchanged. Therefore, it enables the GBFS-GNN to train and infer on domains with higher-arity predicates within a reasonable time and memory, also achieving better outcomes. However, this solution has a potential issue: since each relation node is connected to nodes of higher-arity predicate parameter types, when the domain contains a large number of higher-arity predicates and associated objects, it still introduces many relation edges. For instance, in a domain with 3 higher-arity predicates, if there are 50 objects related to each predicate parameter, this would result in the addition of 150 relation edges, increasing the complexity of the state graph.

Chapter 5

# Evaluation

In this chapter, we conduct extensive experiments on the improved GBFS-GNN to evaluate the performance of our proposed improvement methods. Section 5.1 describes the dataset we used. Section 5.2 outlines the training and testing procedure and introduces the baselines used for comparison. Section 5.3 – 5.6 present the relevant experimental results and provide a thorough discussion.

## 5.1 Dataset

The original GBFS-GNN train and infer on their own ad-hoc datasets. Their experimental results depend heavily on the quality, quantity, and difficulty of the problems in the datasets, making it challenging to evaluate the models' performance fairly and accurately. Therefore, we use the IPC 2023 Learning Track dataset, an public dataset in the learning for planning field, to conduct experiments and compare the performance with other models participating in the IPC 2023 Learning Track to ensure fairness and accuracy.

Each domain in the IPC 2023 Learning Track dataset comprises of a training set and a testing set, where the training set contains 99 problems arranged from easy to hard, and the testing set includes problems categorized into easy, medium, and hard difficulties, with 30 problems in increasing order of difficulty with each category. The IPC 2023 Learning Track dataset includes many domains, such as Blocksworld, Satellite, and Rovers, but only the Rovers domain contains higher-arity predicates. Therefore, we construct the Grippers domain dataset mimicking the structure and difficulty of the IPC 2023 Learning Track dataset to test our model, which is also suitable for experiments on other models in domains with higher-arity predicates.

We evaluate our four proposed improvement methods and two architecture modification methods in the Blocksworld, Satellite, Rovers, and Grippers domains. These four

domains are classic and representative within the planning field, and we will introduce them below:

- **Blocksworld**: A robotic arm needs to rearrange a stack of blocks from an initial configuration to a specified target configuration through a series of moves. This domain is a classic in the planning field, with simple rules that are easy to understand and visualize. However, the search space grows exponentially with the number of blocks, making it an excellent test for the model's basic performance and state space search capabilities. This domain has predicates of arity no larger than 2.
- **Satellite**: Simulates the activities of satellites in orbit, requiring adjustments to their instruments' settings to perform various observation tasks. With multiple satellites with different observational capabilities and observation targets, this domain effectively demonstrates a model's capacity for coordinated planning among multiple agents in a resource-constrained environment. Additionally, the Satellite domain contains many object types and a small proportion of legal edges. This characteristic highlights the effect of illegal edges deletion. This domain has predicates of arity no larger than 2.
- **Grippers**: A Gripper variant with multiple robots and more than two rooms, where robots need to move between rooms and pick up and drop balls. This domain is simple and intuitive, effectively demonstrating the model's path planning capabilities and suitable for testing the model's basic performance on domains with higher-arity predicates. This domain includes the higher-ary predicate $carry(?r - robot, ?o - ball, ?g - gripper)$.
- **Rovers**: Describes various Mars rover exploration tasks, including moving across the Martian surface, collecting rock and soil samples, taking images, and transmitting data back to Earth. This domain involves more complex decision-making and excellently illustrates the model's planning capabilities for multi-objective, multi-constraint tasks in complex environments, suitable for testing the model's advanced decision-making abilities on domains with higher-arity predicates. It contains higher-ary predicates $can\_traverse(?r - rover, ?x - waypoint, ?y - waypoint)$ and $have\_image(?r - rover, ?o - objective, ?m - mode)$.

## 5.2 Experimental Setup

### 5.2.1 Training and Testing Pipeline

We select 90% of simpler problems from the IPC 2023 Learning Track training set as our training set and the remaining 10% of more challenging problems as our validation set, using the data from the IPC 2023 Learning Track testing set as our testing set. In each domain, we train our model using the training set, observe model convergence and choose the best model using the validation set, and test the planning performance

Table 5.1: The problem sizes of each domain's training, validation, and test sets.

| Domain | Type | Training | Validation | Testing | | |
|---|---|---|---|---|---|---|
| | | | | Easy | Medium | Hard |
| Blocksworld | blocks | 2 – 26 | 27 – 29 | 5 – 30 | 35 – 150 | 160 – 500 |
| Satellite | satellites | 1 – 10 | 10 | 3 – 10 | 15 – 40 | 50 – 100 |
| | instruments | 1 – 18 | 19 – 20 | 3 – 20 | 15 – 80 | 50 – 200 |
| | modes | 1 – 3 | 3 | 1 – 3 | 3 – 5 | 5 – 10 |
| | directions | 2 – 10 | 10 | 4 – 10 | 15 – 30 | 40 – 100 |
| Rovers | rovers | 1 – 4 | 4 | 1 – 4 | 5 – 15 | 15 – 30 |
| | waypoints | 4 – 10 | 10 | 4 – 10 | 15 – 90 | 100 – 200 |
| | objectives | 1 – 9 | 9 – 10 | 1 – 10 | 15 – 80 | 100 – 200 |
| | cameras | 1 – 4 | 4 | 1 – 4 | 5 – 50 | 60 – 100 |
| Grippers | robots | 1 – 5 | 6 – 7 | 1 – 7 | 8 – 13 | 14 – 19 |
| | rooms | 2 – 5 | 6 – 7 | 2 – 7 | 8 – 13 | 14 – 19 |
| | balls | 3 – 15 | 16 – 30 | 3 – 30 | 40 – 70 | 80 – 110 |

of the model with the testing set. For each domain, we sequentially tested the model using easy, medium, and hard testing sets. Due to the incremental difficulty of test problems, the testing stops if the model fails to solve three consecutive problems to conserve computational resources and testing time. We use Fast-downward with LAMA-first as a non-learning baseline. We run LAMA-first on each domain with a fixed time limit of 10 minutes per problem. In the Blocksworld domain, we use improved GBFS-GNN with the same time constraint, while in the Grippers and Rovers domains, we use improved GBFS-GNN with a 20-minute limit, and in the Satellite domain, a 30-minute limit per problem. The problem sizes for each domain's training, validation, and testing sets are illustrated Table 5.1.

We use the same hyperparameters across all domains: a hidden layer size of 256, ReLU activation function, a batch size of 300, a PPO clipping ratio of 0.2, a KL divergence cutoff parameter of 0.01, an entropy coefficient of 0.01, a reward discount factor of 0.99, and a learning rate of 0.0001. We trained each domain for up to 1000 iterations, rolling out 100 episodes and performing 20 gradient updates in each training iteration.

Our code is implemented in Python, utilizing PyTorch for neural network construction. We use Pyperplan, a Python based planner, as the successor state generator. All experiments are performed on a single machine with a Intel Xeon Gold 5218R CPU and a NVIDIA V100 GPU.

### 5.2.2 Evaluation Metrics

We use success rate against plan length, number of expanded states, and planning time as evaluation metrics to compare the planning performance of different models. Higher success rates under the same plan length, number of expanded states, and planning time indicate better planning performance; shorter plan lengths, fewer expanded states, and less planning time at the same success rates demonstrate better planning performance. Moreover, we compare the convergence speed of different models using training time against each epoch as a evaluation metric. Less training time per epoch indicates faster convergence. Additionally, we use the average number of actions against each epoch as a evaluation metric to assess training outcomes. Fewer actions per epoch suggest better training outcomes.

### 5.2.3 Baselines

We use Fast-downward with LAMA-first (Helmert, 2006) and models participating in the IPC 2023 Learning Track, such as GOFAI (the Winner) (Alvaro and Gnad, 2023), Muninn (Simon et al., 2023), and ASNets (Mingyu et al., 2023), as baselines for our experiments. GOFAI and Muninn directly utilized competition data from the IPC 2023 Learning Track. Since the ASNets code submitted to IPC 2023 Learning Track had a bug, we contacted the ASNets authors, who provided us with bug-free ASNets' experimental data on the IPC 2023 Learning Track dataset. Since the test environments for improved GBFS-GNN and the baselines differed and the baselines lacked experimental data in terms of the number of expanded states, we compare them only in terms of plan length, and with Fast-downward with LAMA-first in terms of the number of expanded states.

GOFAI, Muninn, and ASNets are run on a fixed time limit of 30 minutes for each problem. Due to computational constraints, we apply timeouts ranging from 10 to 30 minutes for different domains. Additionally, improved GBFS-GNN, implemented in Python and PyTorch without any inference acceleration techniques, requires more than 1 second (3 to 5 seconds in the Rovers domain) for each inference on medium and hard problems, even if excluding the time to run Python code such as encoding. In contrast, GOFAI, Muninn, and Fast-downward implemented in C++ and ASNets using compiled TensorFlow are significantly faster in inference. Based on these differences, directly comparing of improved GBFS-GNN with the baselines on all test problems can be misleading. For this reason, we also provide results considering only the subset of test problems solved by improved GBFS-GNN.

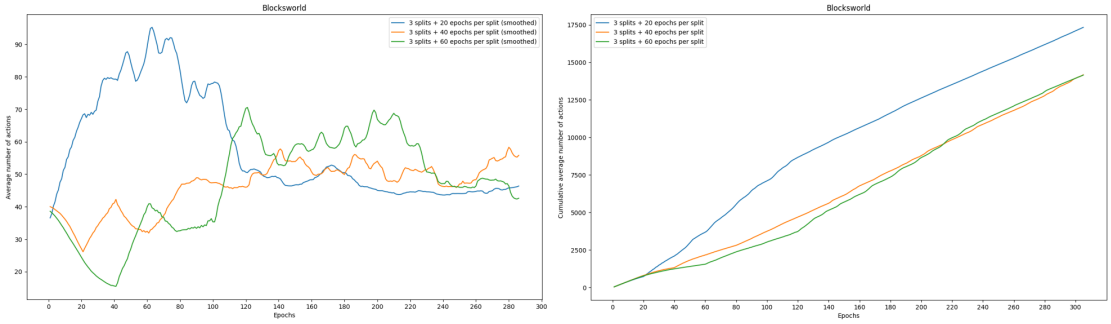


Figure 5.1: Results for the training outcomes of different epochs per split in the Blocksworld domain. On the left: the average number of actions against each epoch. On the right: the cumulative average number of actions against each epoch.

## 5.3 Results for Incremental Training Procedure

We conduct comparative experiments on incremental training procedures with different training parameters in the Blocksworld and Rovers domains to identify the optimal training parameters.

### 5.3.1 Determining Numbers of Epochs per Split

We compare three different epochs per split: 20, 40, and 60 epochs per split with 3 splits in the Blocksworld domain, with each approach training for 300 epochs. Figure 5.1 shows the average number of actions per epoch during training. The figure indicates that the training outcome of 40 and 60 epochs per split is similar and better than that of 20 epochs per split. Figure 5.2 illustrates the planning performance during testing, demonstrating that the planning performance of 60 epochs per split is better than that of 20 epochs per split, which is better than that of 40 epochs per split.

### 5.3.2 Determining Numbers of Splits

Next, we compare three different numbers of splits: 3 splits with 60 epochs per split, 5 splits with 40 epochs per split, and 10 splits with 20 epochs per split in the Blocksworld domain. We train 200 epochs for each approach, at which time all three approaches have been trained with the entire dataset to ensure the fairness and accuracy of the comparison. Figure 5.3 shows the average number of actions per epoch during training. The figure indicates that the training outcome is better as the number of splits increases. Figure 5.4 illustrates the planning performance during testing, demonstrating that the planning performance of 3 and 10 splits is almost identical and slightly better than that of 5 splits.

Given the comparable planning performance of 3 splits with 60 epochs per split and 10 splits with 20 epochs per split in the Blocksworld domain, we also experiment with

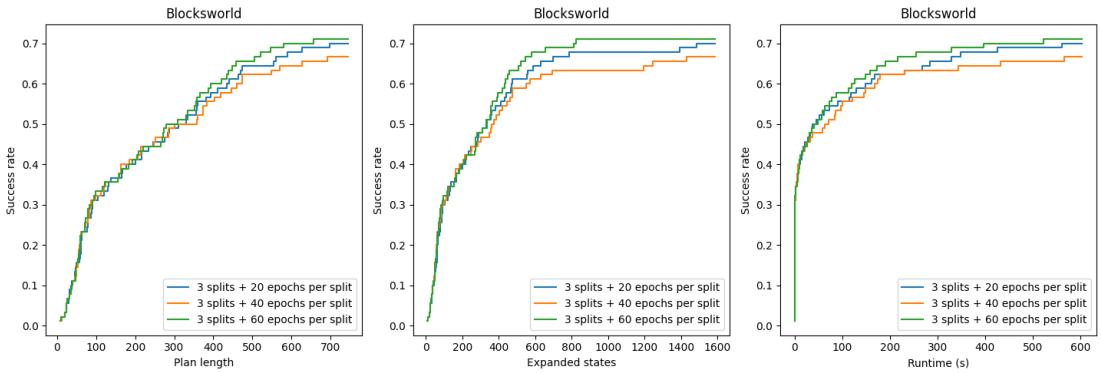


Figure 5.2: Results for the planning performance of different epochs per split in the Blocksworld domain.

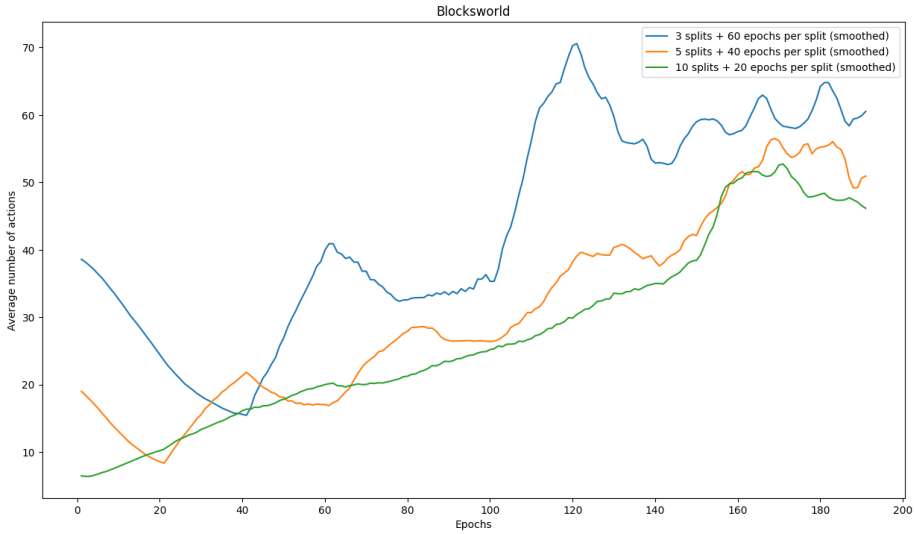


Figure 5.3: Results for the training outcomes of different numbers of splits in the Blocksworld domain.

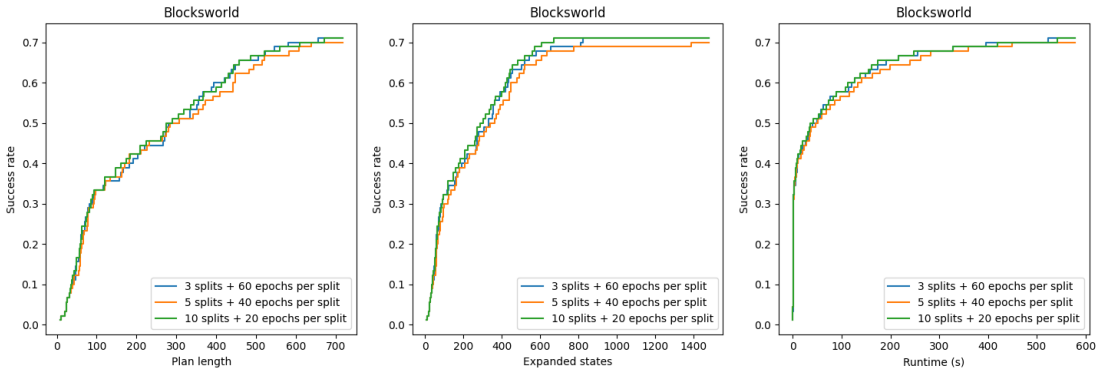


Figure 5.4: Results for the planning performance of different numbers of splits in the Blocksworld domain.

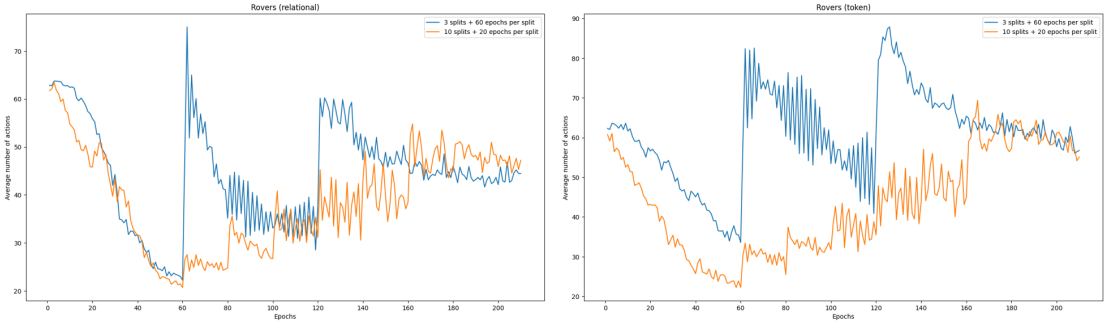


Figure 5.5: Results for the training outcomes of different numbers of splits in the Rovers domain. On the left: the results of improved GBFS-GNN (relational). On the right: the results of improved GBFS-GNN (token).

these two approaches in the Rovers domain. Figure 5.5 shows the average number of actions per epoch for improved GBFS-GNN (relational) and improved GBFS-GNN (token) during training. The figure indicates that the training outcomes of these two models are significantly better for 10 splits than for 3 splits. Figure 5.6 and Figure 5.7 illustrate the planning performance for these two models during testing, demonstrating that the planning performance of these two models is better for 10 splits than 3 splits.

### 5.3.3 Discussion

The planning performance in the Blocksworld domain is nearly identical across different numbers of splits, while in the Rovers domain, the planning performance varies more significantly. We think this is due to different domain characteristics. More splits imply a smaller range of problem difficulty per split, enabling the model to easily learn generalized policies gradually across splits. In the Blocksworld domain, there exists a straightforward generalized policy of "unstack all blocks then stack blocks according to

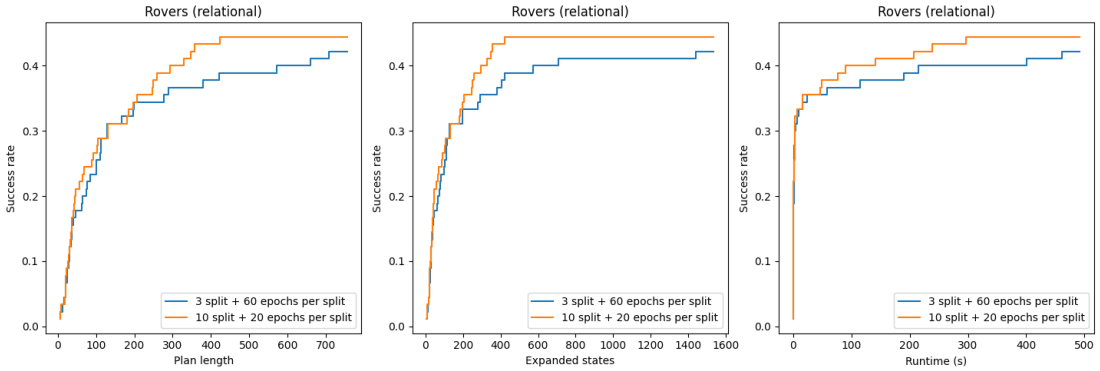


Figure 5.6: Results for the planning performance of improved GBFS-GNN (relational) for different numbers of splits in the Rovers domain.

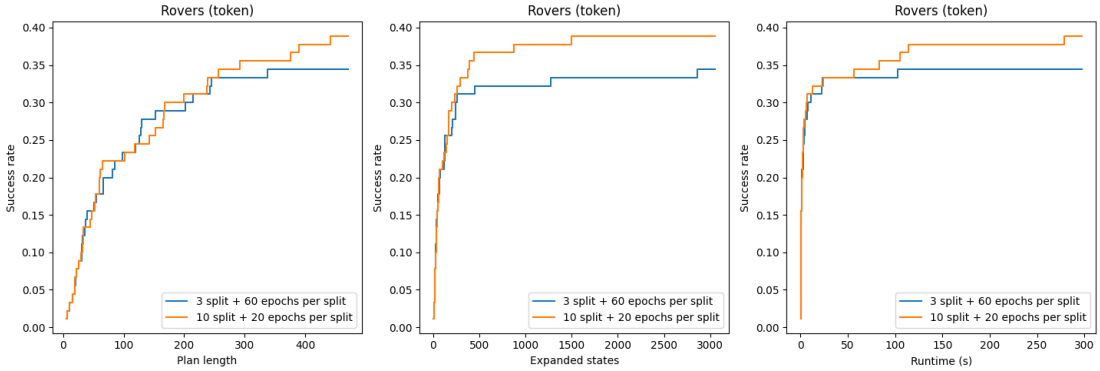


Figure 5.7: Results for the planning performance of improved GBFS-GNN (token) for different numbers of splits in the Rovers domain.

the goal," which allows the model to learn an optimal policy even with a large difficulty range in a split. Thus, the model in the Blocksworld domain is less sensitive to the number of splits. In contrast, the Rovers domain lacks such straightforward generalized policies, making it challenging for the model to learn an optimal policy with larger difficulty ranges in a split. Therefore, the model in the Rovers domain is very sensitive to the number of splits, and using more splits leads to better planning performance.

From these results, we conclude that for large problem-size datasets, compared to training directly on the entire dataset, using an incremental training procedure can significantly reduce training time and computational resources. Moreover, increasing the number of epochs per split can enhance the model's performance. In domains with straightforward generalized policies, the number of splits has a minor impact; however, in complex domains lacking such generalized policies, increasing the number of splits greatly improves model performance.

## 5.4 Comparison of Original GBFS-GNN and Improved GBFS-GNN

We compare the training time and planning performance of the original GBFS-GNN and the improved GBFS-GNN in the Satellite domain to evaluate the performance of our proposed improvement method, including advantage normalization, selecting the most likely action, and illegal edges deletion. Figure 5.8 shows the training time per epoch of the original GBFS-GNN and the improved GBFS-GNN in the Satellite domain, indicating that the improved GBFS-GNN requires less training time in almost every epoch than the original GBFS-GNN. Figure 5.9 shows the planning performance of the original GBFS-GNN and the improved GBFS-GNN in the Satellite domain, indicating that the improved GBFS-GNN outperforms the original GBFS-GNN.

From these results, we conclude that our proposed improvement methods, including advantage normalization, selecting the most likely action, and illegal edges deletion, significantly reduce the training time and accelerate convergence. Moreover, these methods considerably enhance the plan quality and planning efficiency.

## 5.5 Comparison of Improved GBFS-GNN (relational) and Improved GBFS-GNN (token)

We conduct comparative experiments with the improved GBFS-GNN (relational) and the improved GBFS-GNN (token) in Rovers and Grippers domains to explore the appropriate architecture modification method for different domains.

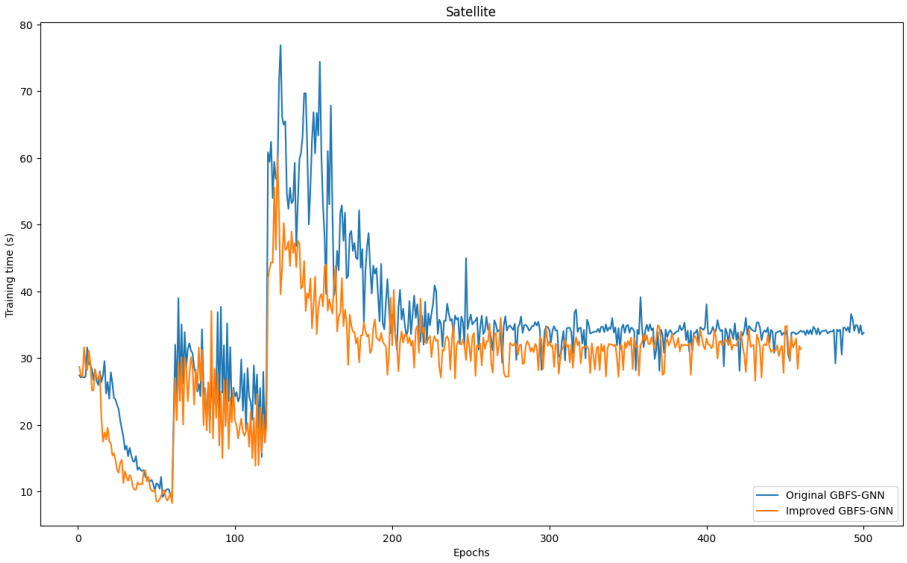


Figure 5.8: Results for the training time against each epoch of original GBFS-GNN and improved GBFS-GNN in the Satellite domain.

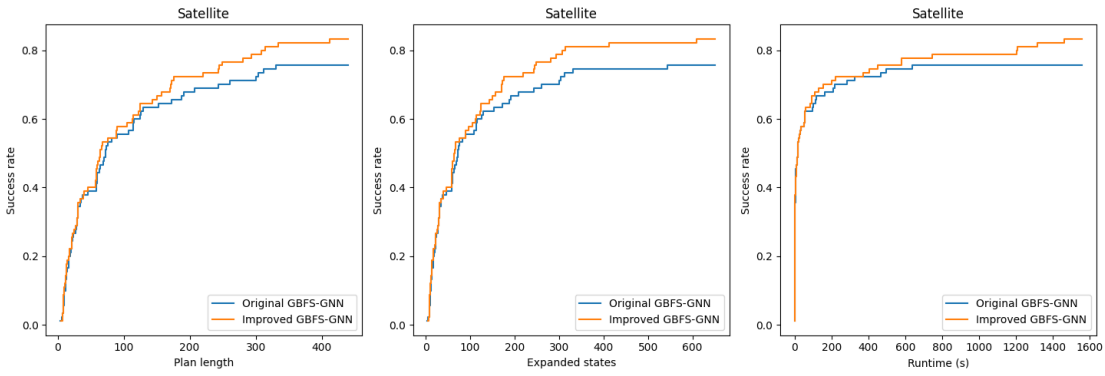


Figure 5.9: Results for the planning performance of the original GBFS-GNN and the improved GBFS-GNN in the Satellite domain.

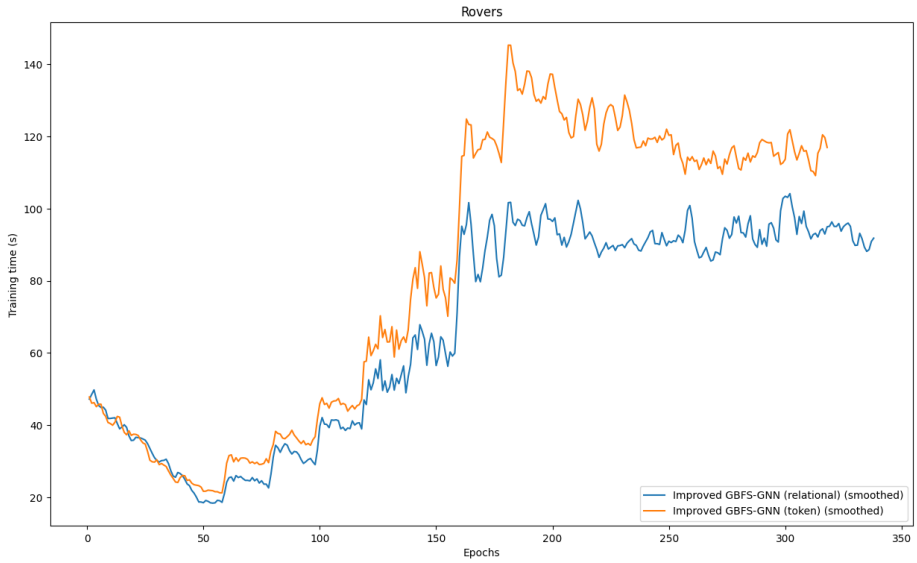


Figure 5.10: Results for the training time against each epoch of the improved GBFS-GNN (relational) and the improved GBFS-GNN (token) in the Rovers domain.

### 5.5.1 Rovers Domain

We compare training time and planning performance of the improved GBFS-GNN (relational) and the improved GBFS-GNN (token) in Rovers domain. Figure 5.10 shows the training time per epoch of these two model in the Rovers domain. The figure indicates that the improved GBFS-GNN (relational) requires less training time in almost every epoch than the improved GBFS-GNN (token). Figure 5.11 illustrates the planning performance of these two models in the Rovers domain, demonstrating that the improved GBFS-GNN (relational) significantly outperforms the improved GBFS-GNN (token).

### 5.5.2 Grippers Domain

We compare training time and planning performance of the improved GBFS-GNN (relational) and the improved GBFS-GNN (token) in Grippers domain. Figure 5.12 shows the training time per epoch of these two model in the Grippers domain. The figure indicates that the improved GBFS-GNN (token) requires less training time in almost every epoch than the improved GBFS-GNN (relational). Figure 5.13 illustrates the planning performance of these two models in the Grippers domain, demonstrating that the improved GBFS-GNN (token) outperforms the improved GBFS-GNN (relational).

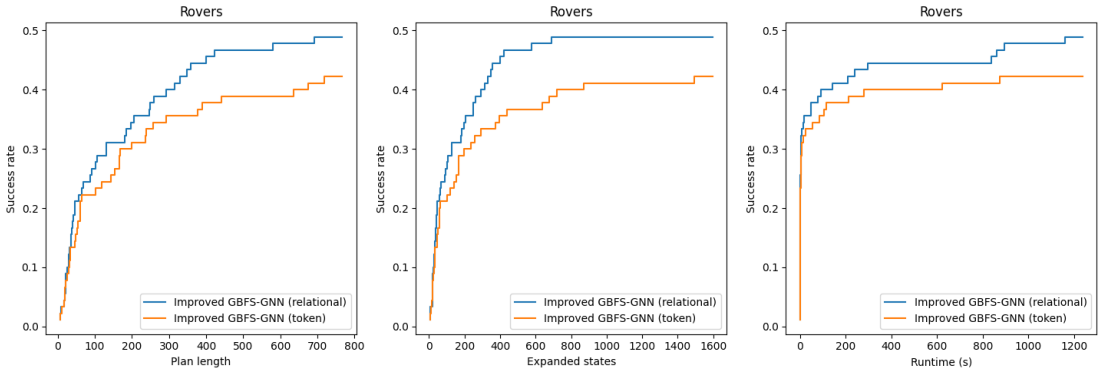


Figure 5.11: Results for the plan performance of the improved GBFS-GNN (relational) and the improved GBFS-GNN (token) in the Rovers domain.

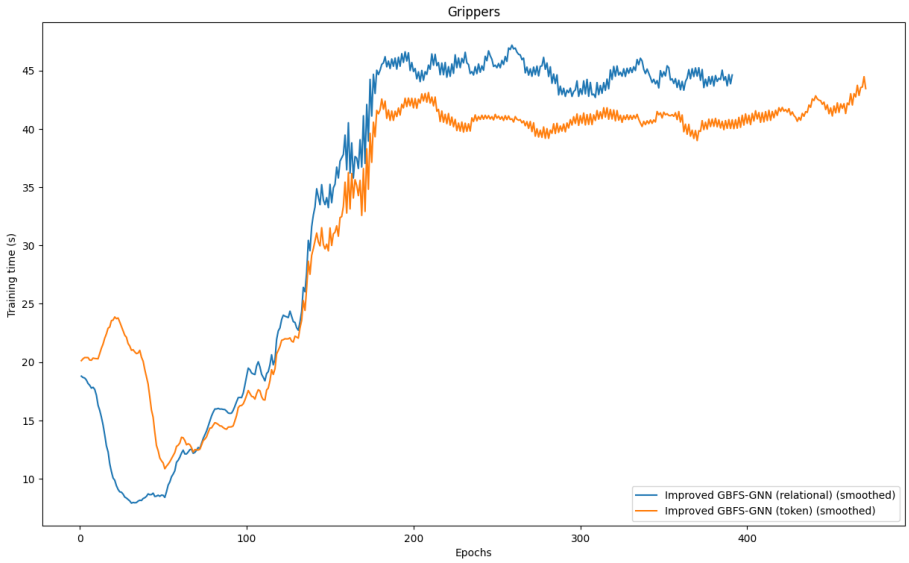


Figure 5.12: Results for the training time against each epoch of the improved GBFS-GNN (relational) and the improved GBFS-GNN (token) in the Grippers domain.

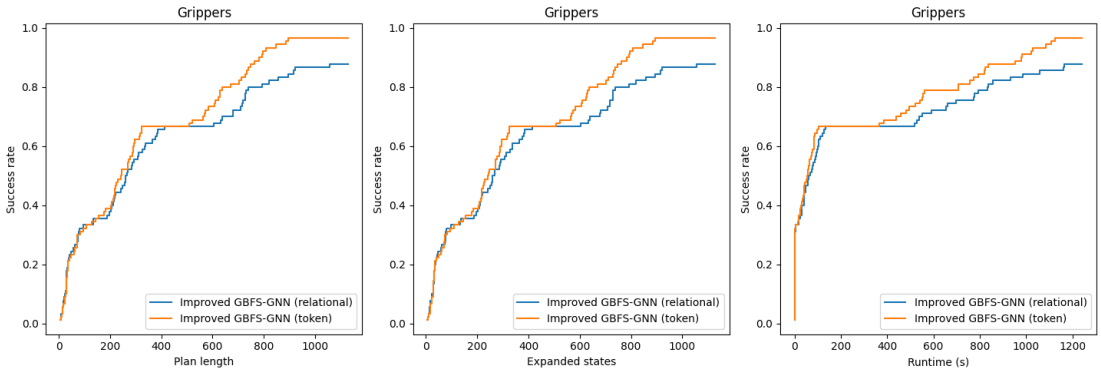


Figure 5.13: Results for the plan performance of the improved GBFS-GNN (relational) and the improved GBFS-GNN (token) in the Grippers domain.

### 5.5.3 Discussion

From these results, we conclude that both improved GBFS-GNN (relational) and improved GBFS-GNN (token) can solve problems in domains with higher arity. Improved GBFS-GNN (token) is more suited for simpler domains, while improved GBFS-GNN (relational) is better suited for more complex domains.

We think that the performance of improved GBFS-GNN (relational) is roughly the same in both the Rovers and Grippers domains, while improved GBFS-GNN (token) performs better in the simpler Grippers domain and worse in the more complex Rovers domain. This may be due to the Grippers domain having fewer predicates (4 predicates) than the Rovers domain (22 predicates). Each predicate represents a dimension of information and the more predicates in a domain, the greater the amount of information each node contains. A relation node represents the relations among connected nodes, and information it needs to learn and express is proportional to the sum of the information of the connected nodes. If a relation node in these two domains is connected to the same number of nodes, then a relation node in the Rovers domain would need to learn and express 5.5 times more information as in the Grippers domain. Thus, a relation node might perfectly learn all the information of its connected nodes in the Grippers domain and fully express their relations. In contrast, in the Rovers domain, a relation node might struggle to learn all the information of its connected nodes nor completely express their relations.

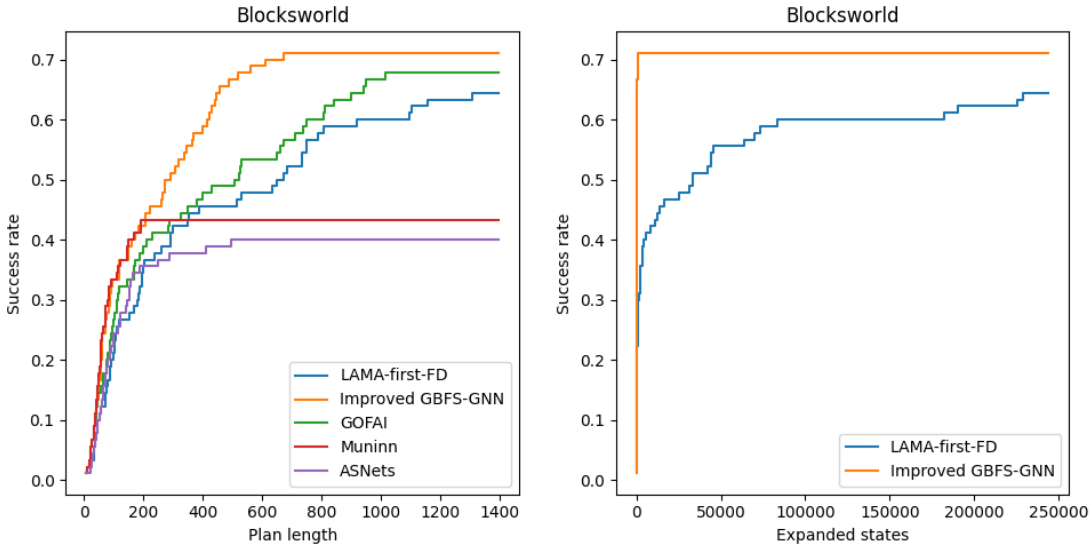


Figure 5.14: Results for the planning performance of improved GBFS-GNN and baselines in the Blocksworld domain.

## 5.6 Comparison with Baselines

### 5.6.1 Domains with Low-arity Predicates

#### Blocksworld Domain

We compare the improved GBFS-GNN with baselines in the Blocksworld domain to evaluate the performance of our proposed improvement methods. The result is shown in Figure 5.14 and indicate that improved GBFS-GNN significantly outperforms other baselines.

#### Satellite Domain

We compare the improved GBFS-GNN with baselines in the Satellite domain to evaluate the performance of our proposed improvement methods.

The left of Figure 5.15 shows the plan length of the improved GBFS-GNN and baselines for all test problems. The improved GBFS-GNN exhibits higher plan quality than Muninn and ASNets, is comparable to GOFAI, but is below Fast-downward with LAMA-first. The right of Figure 5.15 illustrates the plan length of these models for the subset of test problems solved by improved GBFS-GNN, indicating that at the same success rate, the plan quality of the improved GBFS-GNN and Fast-downward with LAMA-first are roughly equivalent.

Figure 5.16 shows the number of expanded states of the improved GBFS-GNN and baselines for all test problems and the subset of test problems solved by the improved

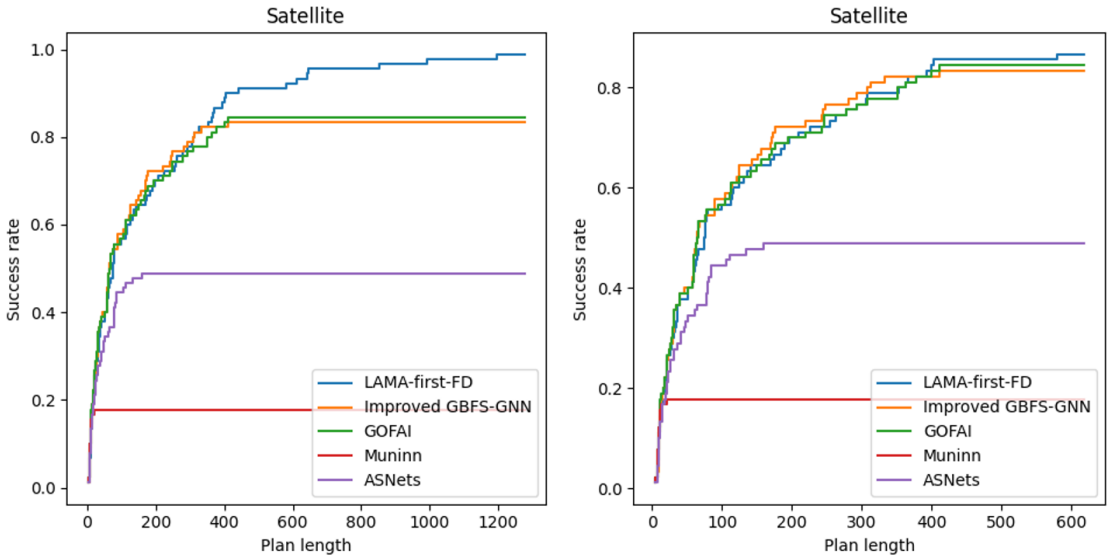


Figure 5.15: Results for the plan length of improved GBFS-GNN and baselines in the Satellite domain. On the left: the results of all test problems. On the right: the results considering only the subset of test problems solved by improved GBFS-GNN.

GBFS-GNN in the Satellite domain. The figure demonstrates that improved GBFS-GNN exhibits a significantly higher planning efficienc than Fast-downward with LAMA-first.

#### Discussion

From these results, we conclude that for domains with low-arity predicates, the plan quality of the improved GBFS-GNN surpasses or equals that of the best performance learning model GOFAI and the traditional heuristic search algorithm LAMA-first. In the meantime, the planning efficiency of improved GBFS-GNN is much higher than that of LAMA-first.

### 5.6.2 Domains with Higher-arity Predicates

#### Rovers Domain

We compare the improved GBFS-GNN (relational), improved GBFS-GNN (token), and baselines in the Rovers domain to evaluate the performance of our proposed architecture modification methods.

The left of Figure 5.17 shows the plan length of the improved GBFS-GNN (relational), improved GBFS-GNN (token), and baselines for all test problems. The improved GBFS-GNN (token) exhibits higher plan quality than Muninn and ASNets, while the plan quality of improved GBFS-GNN (relational) is below that of GOFAI and Fast-downward

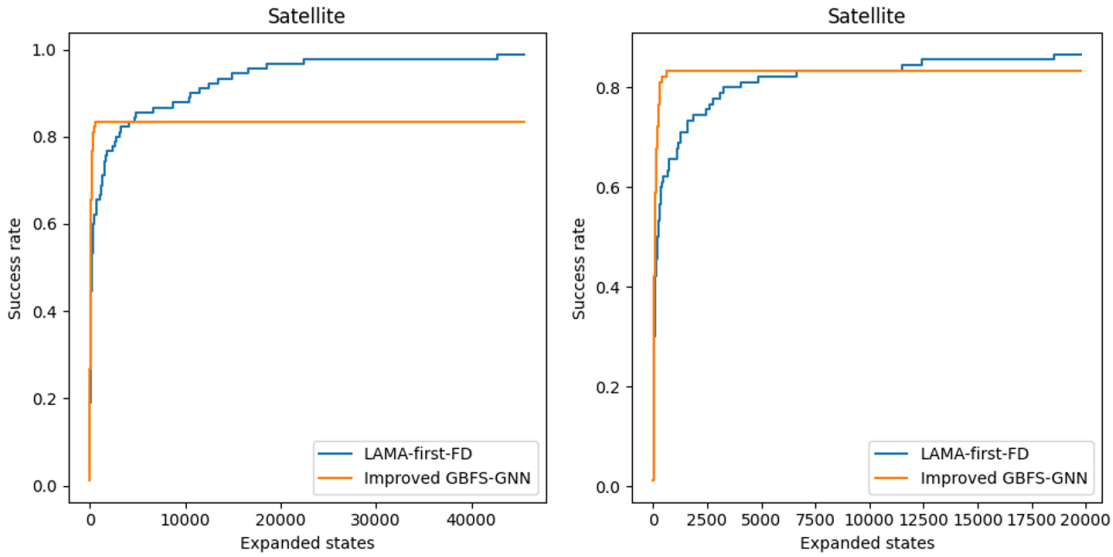


Figure 5.16: Results for the number of expanded states of improved GBFS-GNN and baselines in the Satellite domain. On the left: the results of all test problems. On the right: the results considering only the subset of test problems solved by improved GBFS-GNN.

with LAMA-first. The right of Figure 5.17 illustrates the plan length of these models for the subset of test problems solved by the improved GBFS-GNN (relational), indicating that at the same success rate, the plan quality of the improved GBFS-GNN (relational), GOFAI, and Fast-downward with LAMA-first are roughly equivalent.

Figure 5.18 shows the number of expanded states of the improved GBFS-GNN (relational), improved GBFS-GNN (token), and baselines for all test problems and the subset of test problems solved by the improved GBFS-GNN (relational) in the Rovers domain. The figure demonstrates that improved GBFS-GNN (relational) and improved GBFS-GNN (token) exhibit a significantly higher planning efficienc than Fast-downward with LAMA-first.

### Grippers Domain

We compare the improved GBFS-GNN (relational), improved GBFS-GNN (token), and baselines in the Grippers domain to evaluate the performance of our proposed architecture modification methods.

The right of Figure 5.19 shows the number of expanded states of the improved GBFS-GNN (relational), improved GBFS-GNN (token), and Fast-downward with LAMA-first for all test problems in the Grippers domain. The figure demonstrates that improved GBFS-GNN (relational) and improved GBFS-GNN (token) exhibit a significantly higher planning efficienc than Fast-downward with LAMA-first. The left of Figure 5.19 show

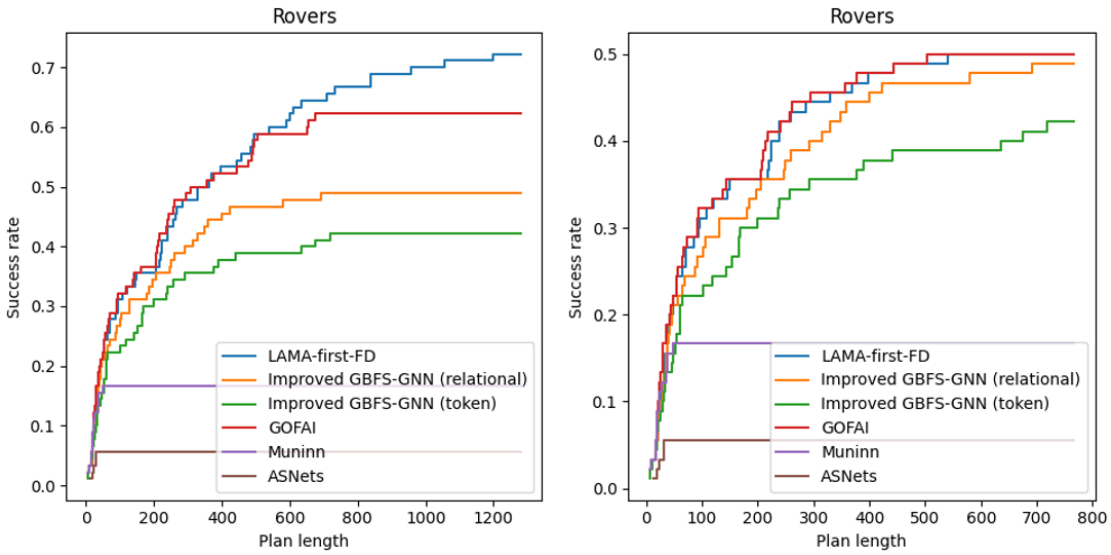


Figure 5.17: Results for the plan length of improved GBFS-GNN (relational), improved GBFS-GNN (token), and baselines in the Rovers domain. On the left: the results of all test problems. On the right: the results considering only the subset of test problems solved by improved GBFS-GNN (relational).

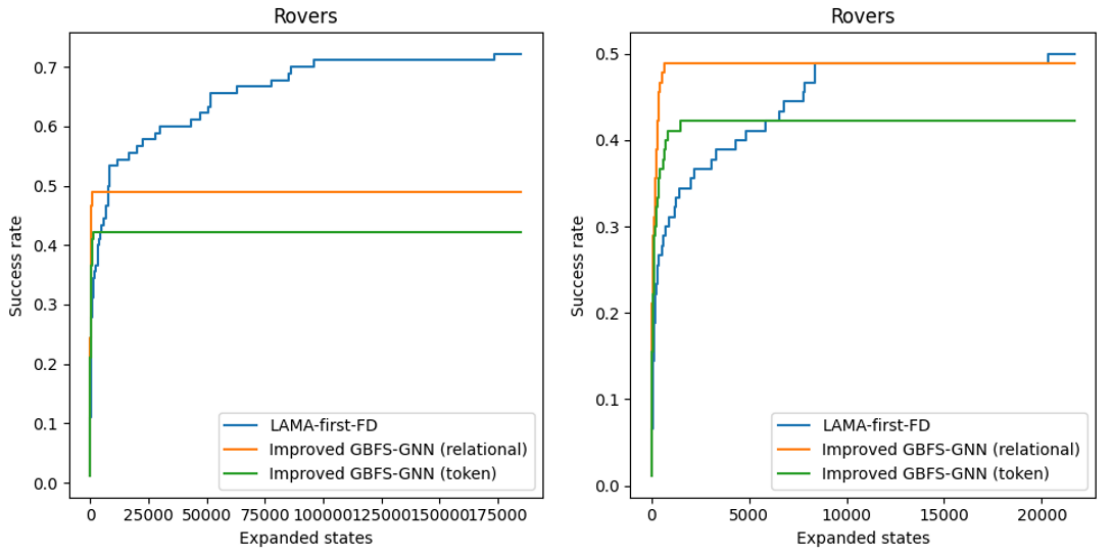


Figure 5.18: Results for the number of expanded states of improved GBFS-GNN (relational), improved GBFS-GNN (token), and baselines in the Rovers domain. On the left: the results of all test problems. On the right: the results considering only the subset of test problems solved by improved GBFS-GNN (relational).

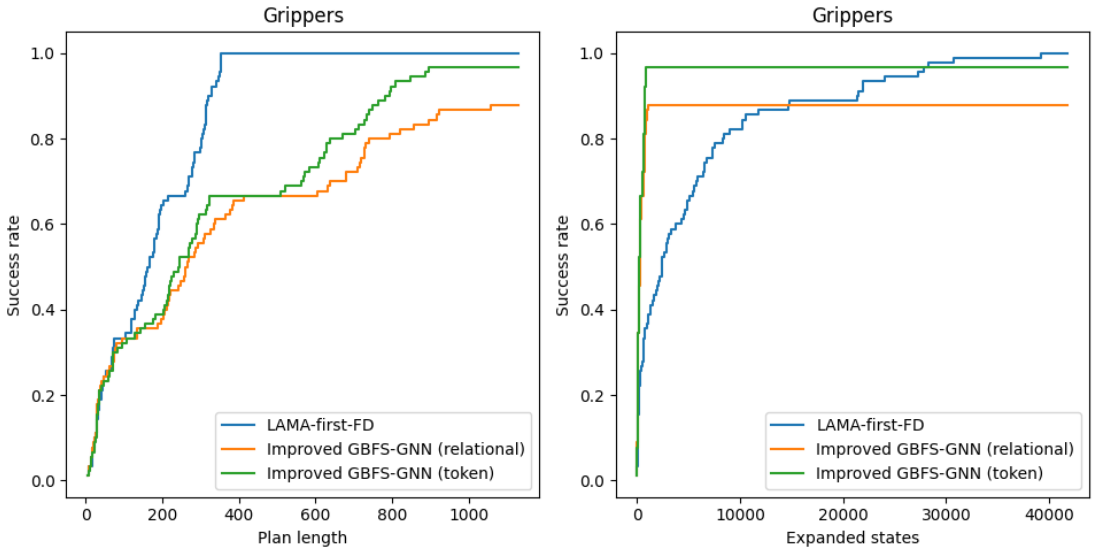


Figure 5.19: Results for the plan performance of improved GBFS-GNN (relational), the improved GBFS-GNN (token), and Fast-downward with LAMA-first in the Grippers domain.

the plan length of these three models for all test problems, indicating that the plan quality of improved GBFS-GNN (relational) and improved GBFS-GNN (token) is below that of Fast-downward with LAMA-first.

### Discussion

From these results, we conclude that for domains with higher-arity predicates, the planning efficiency of improved GBFS-GNN (relational) and improved GBFS-GNN (token) is much better than the traditional heuristic search algorithm LAMA-first, and their plan quality is significantly higher than Muninn and ASNets, but slightly lower than GOFAI and LAMA-first.

Chapter 6

# Conclusion and Future Work

In this chapter, we provide a summary of our main contributions in this thesis and propose several future research directions for GBFS-GNN.

## 6.1 Contributions

The main objective of this thesis is to improve and address the limitations of GBFS-GNN, enabling GBFS-GNN to train on domains with arbitrary arity predicates and large problem-size datasets within a reasonable time while enhancing model performance and reducing the training time. The contributions of this thesis include:

#### Improving Training and Inference Performance

The original GBFS-GNN's planning performance for complex domains and large-size problems is poor, and its training times are too long. Additionally, the original GBFS-GNN cannot complete training on large problem-size datasets within a reasonable time. Therefore, we replicate the original GBFS-GNN and introduce four optimization methods to address these limitations:

- **Advantage Normalization**: We use the advantage normalization technique to standardize advantage estimation in the reinforcement model, reducing its variance, enhancing training stability, accelerating model convergence, and improving planning performance.
- **Selecting Most Likely Action**: Instead of sampling the action, we select the action with the highest probability during inference. This method fully utilizes the trained policy, converging more quickly to the optimal solution and enhancing planning performance.

- **Illegal Edges Deletion**: We remove illegal edges between nodes with no binary relations in the original state graph to reduce training and inference time and enhance planning performance.
- **Incremental Training Procedure**: We sort the dataset by problem size and divide it into multiple splits. Starting training with the first split, after several epochs, the next split is merged into the current training set and training is resumed. This process is repeated until the full dataset is utilized for training. Training is stopped if there is no improvement in performance on the validation set after several consecutive evaluations. This method allows GBFS-GNN to train on large problem-size datasets within a reasonable time while saving computational resources and training time.

### Handling Domains with Higher-arity Predicates

The original GBFS-GNN encodes low-arity predicates as features in the state graph. However, it is unable to solve problems in domains with higher-arity predicates. To address this limitation, we introduce two types of solutions:

- **Decomposition**: We decompose higher-arity predicates in the domain into binary predicates. Then, the decomposed domain and problem are input into the original GBFS-GNN for training and inference. We propose two decomposition methods, relational decomposition and token-based decomposition, and compare their characteristics.
- **Architecture Modification**: We modify the architecture of the original GBFS-GNN to directly encode higher-arity predicates as the graph components. We develop two modification methods: (1) simulating the relational decomposition by representing higher-arity predicates as binary predicates to encode states and effects; (2) simulating the token-based decomposition by introducing the hub nodes in the state graph representing the relation between parameters of higher-arity predicates to encode states and effects.

### Standardized Evaluation

The original GBFS-GNN trains and infers on their ad-hoc datasets. Their experimental results depend heavily on the quality, quantity, and difficulty of the problems in the datasets, making it challenging to fairly and accurately evaluate the models' performance. Therefore, we utilize the IPC 2023 Learning Track dataset to conduct experiments and compare the performance with other models participating in the IPC 2023 Learning Track to ensure fairness and accuracy.

Additionally, since only the Rovers domain in the IPC 2023 Learning Track dataset contains higher-arity predicates, we construct the Grippers domain dataset that mimics the structure and difficulty of the IPC 2023 Learning Track dataset to evaluate our

model. This dataset can also be used for experiments with other models in domains containing higher-arity predicates.

## 6.2 Future work

Although our proposed improved GBFS-GNN achieved good experimental results, there are still many areas for future exploration, which can be categorized into three themes: improving performance, improving the training procedure, and extending to more complex domains. We will discuss them in detail in the next sections.

### 6.2.1 Improving Performance

#### Optimizing Implementation

Our code is implemented in Python, using the Python-based Pyperplan as the successor state generator and PyTorch for constructing neural networks. However, Python's performance is considerably slower than C++ and Pyperplan is primarily designed as a teaching tool rather than a high-performance planner. Moreover, we do not use any accelerated inference techniques for PyTorch. Consequently, the computational performance of improved GBFS-GNN is lacking.

In the future, we could refactor our code in C++ and use Fast-downward, a highly optimized classical planner, as the successor state generator. We could also implement JIT compilation or static runtime compilation for PyTorch and employ high-performance inference engines like TensorRT and ONNX Runtime. Additionally, we can draw inspiration from Asynchronous Advantage Actor-critic (A3C) (Mnih et al., 2016) to perform parallel training and inference on multiple CPU cores and GPUs. We anticipate that these optimizations could significantly enhance the training and inference speed of improved GBFS-GNN.

#### Combining Sampling Action and Selecting Most Likely Action

As mentioned in Subsection 3.2.2, sampling the action and selecting the most likely action each have advantages and disadvantages, suitable for different models, domains, and problems. In the future, we can design a method that combines these two approaches: for the action probability distribution, when the highest probability or the difference between the highest probability and the second highest probability reaches a certain threshold, it indicates high confidence in the most likely action, thereby, we select the most likely action to quickly converge to the optimal plan. Otherwise, we sample the action to reduce backtracking and maintain exploration.

#### Combining Supervised Learning and Reinforcement Learning

Supervised learning can learn the inherent patterns from large datasets and converge quickly, but it may not make the optimal decision in state spaces not covered by the

training data. On the other hand, reinforcement learning learns problem-solving policies through interaction with the environment, enabling it to make good decisions in unseen states. However, reinforcement learning struggles to obtain reward signals in the early stages of training and converges slowly.

AlphaGo (Silver et al., 2016) achieved great success by combining these two approaches: First, it uses supervised learning with professional Go game records to learn local strategies. Then, reinforcement learning is used to learn global strategies such as situation assessment. In the future, we could use this method of combining supervised learning and reinforcement learning. In the early stages of training, we can use traditional planning algorithms to acquire optimal solutions of simple problems and use supervised learning with these data to train the GBFS-GNN, enabling it to quickly learn the optimal foundational policy. Then, we could use reinforcement learning to train the GBFS-GNN to explore and learn advanced policy. At this stage, GBFS-GNN has a better starting point and avoids aimless exploration and trial-and-error in large state spaces. We expect this approach to accelerate the convergence rate and yield superior final policy.

### 6.2.2 Improving Train Procedure

#### Adaptive Incremental Training Procedure

Our proposed incremental training procedure needs to adjust hyperparameters like the number of epochs per split and the number of splits according to the characteristics and difficulty of different domains and datasets. In the future, we can design an adaptive incremental training procedure: starting training with the simplest problem, the next more difficult problem is merged into the training set when accuracy converges to a certain threshold. This process is repeated until training on the entire dataset. In the meantime, the threshold should be adjusted automatically based on training progress. We expect that this approach could enable the incremental training procedure to be applied to different domains and datasets without adjusting hyperparameters.

#### Never-end Learning

Most learning for planning models learn generalized policies during the training phase, and only utilize the generalized policies solve the planning problem during the testing phase without using the test data to enhance the model performance. In practical industrial scenario, this leads to a significant waste of valuable data, preventing the model from self-evolving with feedback. In contrast, human learning is a continuous process. We learn basic knowledge from courses and continuously gain practical experience to enhance our problem-solving abilities.

Mitchell et al. (2018) proposes the concept of never-ending learning and presents Never-Ending Language Learner (NELL), which continuously learns from the data on the Internet 24 hours a day to enrich and optimize its knowledge base. We can draw inspiration from never-ending learning: during the testing phase, for unsolved problems

or problems with poor plan quality, we could simplify them until the model can occasionally solve them. These simplified problems are then used for training until the model can efficiently solve them. Then, we increase the complexity of these problems until the model can occasionally solve them again and continue to train the model with these adjusted problems. This process is repeated until the model can efficiently solve the original problems. We expect this approach will enable models to self-evolve and continually learn stronger generalized policies from new test data.

### 6.2.3 Extending to More Complex Domains

#### Probabilistic Problem

In the probabilistic domain, actions can lead to stochastic outcomes, each with a different probability. Probabilistic planning aims to generate a plan that reaches a goal state with 100% probability and minimal cost. For instance, the Triangle Tireworld domain is a typical probabilistic domain simulating a vehicle navigating a triangular grid map from an initial location to a target location. The vehicle has a certain probability of experiencing a flat tire with each move, and without a spare tire, it cannot proceed. Spare tires are only available at certain locations on the map. Planners must consider these locations with spare tires to address potential flat tires while finding the shortest path from the start to the destination.

Determinization relaxes a probabilistic planning problem to a deterministic problem by converting probabilistic actions into one or more deterministic actions (Bonet and Geffner, 2003). This method ignores the probability of outcomes and undesirable impacts of probabilistic actions, performing poorly in domains like Triangle Tireworld, which have avoidable dead ends. The non-determinization heuristics $h^{pom}$ and $h^{roc}$ (Trevizan et al., 2017a) achieve good performance across various probabilistic domains. However, these heuristics are difficult to construct and have high computational complexity.

ASNets (Toyer et al., 2018, 2020) could solve problems in probabilistic domains and have learned optimal generalized policy in the Triangle Tireworld domain. ASNets enable the model to automatically learn possible outcomes during training instead of encoding the probabilities of outcomes into feature vectors. However, this approach may miss some outcomes during training if preconditions for certain probabilistic actions are hard to meet or the probability of undesirable outcomes is low. For example, in a medical domain, a surgical action of a rare disease has a 5% probability of resulting in patient death, which may never occur during training. Moreover, not all practical scenarios strictly require a 100% probability of solving a problem. For instance, there may be a 0.01% chance of a traffic accident on the way to work, yet we still choose to go to work every day. Therefore, developing a model capable of making trade-off decisions is also critical.

In the future, we could redesign GBFS-GNN architecture to encode the probabilities of outcomes into the action representation and redesign the reward function to penalize the actions that lead to dead ends. The training set should also be specifically designed

to include all possible situations. We expect these optimizations to effectively address problems in probabilistic domains.

### Constrained Problem

A constrained planning problem is a deterministic or probability planning problem with secondary constraints, for instance, maximum fuel allowed to be used and sequence of states that cannot happen or must happen with low probability. Several planning models exist for this class of problems based on the type of constraints being modeled, for instance, C-SSPs (Trevizan et al., 2017b), C-SSPS with PLTL constraints (Baumgartner et al., 2018; Mallet et al., 2021), and MO-Models (Geisser et al., 2022; Chen et al., 2023). All the algorithms to solve these models are much more computationally expensive than algorithms for classical planning. At the same time, the heuristics for such models are not as informative, resulting in less efficient heuristic search, that is, much more nodes expanded to find a solution. In the future, we could redesign GBFS-GNN architecture to encode the constraints into state and action representations and redesign the reward function to penalize agents for constraint violations. The search algorithm used in GBFS-GNN should also be redesigned to correct constraint violations by the neural network. We hope these improvements could provide an efficient alternative to compute sub-optimal solutions to some of these models.

# Bibliography


Aeronautiques, C.; Howe, A.; Knoblock, C.; McDermott, I. D.; Ram, A.; Veloso, M.; Weld, D.; Sri, D. W.; Barrett, A.; Christianson, D.; et al., 1998. Pddl— the planning domain definition language. *Technical Report, Tech. Rep.*, (1998). [Cited on page 8.]

Alvaro, T. and Gnad, D., 2023. Gofai. *Tenth International Planning Competition (IPC-10) Learning Track: Planner Abstracts*, (2023). [Cited on page 70.]

Arfaee, S. J.; Zilles, S.; and Holte, R., 2010. Bootstrap learning of heuristic functions. In *Proceedings of the International Symposium on Combinatorial Search*, vol. 1, 52–60. [Cited on page 19.]

Bajpai, A. N. and Garg, S., 2018. Transfer of deep reactive policies for mdp planning. *Advances in Neural Information Processing Systems*, 31 (2018). [Cited on page 21.]

Battaglia, P. W.; Hamrick, J. B.; Bapst, V.; Sanchez-Gonzalez, A.; Zambaldi, V.; Malinowski, M.; Tacchetti, A.; Raposo, D.; Santoro, A.; Faulkner, R.; et al., 2018. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, (2018). [Cited on pages 14 and 15.]

Baumgartner, P.; Thiébaux, S.; and Trevizan, F., 2018. Heuristic Search Planning With Multi-Objective Probabilistic LTL Constraints. In *Proc. of 16th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR)*. [Cited on page 90.]

Blum, A. L. and Furst, M. L., 1997. Fast planning through planning graph analysis. *Artificial intelligence*, 90, 1-2 (1997), 281–300. [Cited on page 1.]

Boddy, M. S.; Gohde, J.; Haigh, T.; and Harp, S. A., 2005. Course of action generation for cyber security using classical planning. In *ICAPS*, 12–21. [Cited on page 1.]

Bonet, B. and Geffner, H., 2001. Planning as heuristic search. *Artificial Intelligence*, 129, 1-2 (2001), 5–33. [Cited on page 2.]

Bonet, B. and Geffner, H., 2003. Labeled rtdp: Improving the convergence of real-time dynamic programming. In *Proc. of 13th Int. Conf. on Automated Planning and Scheduling (ICAPS)*. [Cited on page 89.]

Brown, T.; Mann, B.; Ryder, N.; Subbiah, M.; Kaplan, J. D.; Dhariwal, P.; Neelakantan, A.; Shyam, P.; Sastry, G.; Askell, A.; et al., 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33 (2020), 1877–1901. [Cited on page 2.]

Chen, D.; Thiébaux, S.; and Trevizan, F., 2024. Learning Domain-Independent Heuristics for Grounded and Lifted Planning. In *Proc. of 38th AAAI Conference on Artificial Intelligence*. [Cited on page 20.]

Chen, D.; Trevizan, F.; and Thiébaux, S., 2023. Heuristic Search for Multi-Objective Probabilistic Planning. In *Proc. of 37th AAAI Conference on Artificial Intelligence*. [Cited on page 90.]

Chien, S.; Rabideau, G.; Knight, R.; Sherwood, R.; Engelhardt, B.; Mutz, D.; Estlin, T.; Smith, B.; Fisher, F.; Barrett, T.; et al., 2000. Aspen-automating space mission operations using automated planning and scheduling. In *SpaceOps 2000*. AIAA Press. [Cited on page 1.]

Dalal, N. and Triggs, B., 2005. Histograms of oriented gradients for human detection. In *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*, vol. 1, 886–893. Ieee. [Cited on page 2.]

Do, M. B.; Ruml, W.; and Zhou, R., 2008. Planning for modular printers: Beyond productivity. In *ICAPS*, 68–75. [Cited on page 1.]

Elman, J. L., 1990. Finding structure in time. *Cognitive science*, 14, 2 (1990), 179–211. [Cited on page 2.]

Ferber, P.; Geisser, F.; Trevizan, F.; Helmert, M.; and Hoffmann, J., 2022. Neural Network Heuristic Functions for Classical Planning: Bootstrapping and Comparison to Other Methods. In *Proc. of 32nd Int. Conf. on Automated Planning and Scheduling (ICAPS)*. [Cited on page 19.]

Fikes, R. E. and Nilsson, N. J., 1971. Strips: A new approach to the application of theorem proving to problem solving. *Artificial intelligence*, 2, 3-4 (1971), 189–208. [Cited on page 8.]

Garg, S.; Bajpai, A.; et al., 2019. Size independent neural transfer for rddl planning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 29, 631–636. [Cited on page 21.]

Garrett, C. R.; Kaelbling, L. P.; and Lozano-Pérez, T., 2016. Learning to rank for synthesizing planning heuristics. *arXiv preprint arXiv:1608.01302*, (2016). [Cited on page 19.]

Geffner, H. and Bonet, B., 2013. A concise introduction to models and methods for automated planning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 8, 1 (2013), 1–141. [Cited on pages 1 and 7.]

Geisser, F.; Haslum, P.; Thiébaux, S.; and Trevizan, F., 2022. Admissible Heuristics for Multi-Objective Planning. In *Proc. of 32nd Int. Conf. on Automated Planning and Scheduling (ICAPS)*. [Cited on page 90.]

Geisser, F.; Poveda, G.; Trevizan, F.; Bondouy, M.; Teichteil-Königsbuch, F.; and Thiébaux, S., 2020. Optimal and Heuristic Approaches for Constrained Flight Planning under Weather Uncertainty. In *Proc. of 30th Int. Conf. on Automated Planning and Scheduling (ICAPS)*. [Cited on page 1.]

Geissmann, C., 2015. *Learning heuristic functions in classical planning*. Ph.D. thesis, Master's thesis, University of Basel, Switzerland. [Cited on page 19.]

Grammarly, 2024. Grammarly: Online writing assistant. https://www.grammarly.com. Used for grammar revision and sentence polish. [Not cited.]

Groshev, E.; Goldstein, M.; Tamar, A.; Srivastava, S.; and Abbeel, P., 2018. Learning generalized reactive policies using deep neural networks. In *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 28, 408–416. [Cited on page 21.]

Gupta, S. K.; Bourne, D. A.; Kim, K.; and Krishnan, S., 1998. Automated process planning for sheet metal bending operations. *Journal of Manufacturing Systems*, 17, 5 (1998), 338–360. [Cited on page 1.]

Hao, M.; Trevizan, F.; Thiébaux, S.; Ferber, P.; and Hoffmann, J., 2024. Guiding GBFS through Learned Pairwise Rankings. In *Proc. of 33rd Int. Joint Conf. on AI (IJCAI)*. [Cited on page 20.]

Helmert, M., 2006. The fast downward planning system. *Journal of Artificial Intelligence Research*, 26 (2006), 191–246. [Cited on page 70.]

Hoffmann, J. and Nebel, B., 2001. The ff planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14 (2001), 253–302. [Cited on pages 2 and 11.]

Kautz, H. A.; Selman, B.; et al., 1992. Planning as satisfiability. In *ECAI*, vol. 92, 359–363. Citeseer. [Cited on page 1.]

Kipf, T. N. and Welling, M., 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, (2016). [Cited on page 14.]

LeCun, Y.; Bottou, L.; Bengio, Y.; and Haffner, P., 1998. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86, 11 (1998), 2278–2324. [Cited on pages 2 and 14.]

Liu, Z.; Lin, Y.; Cao, Y.; Hu, H.; Wei, Y.; Zhang, Z.; Lin, S.; and Guo, B., 2021. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF international conference on computer vision*, 10012–10022. [Cited on page 2.]

Lowe, D. G., 1999. Object recognition from local scale-invariant features. In *Proceedings of the seventh IEEE international conference on computer vision*, vol. 2, 1150–1157. Ieee. [Cited on page 2.]

Mallet, I.; Thiébaux, S.; and Trevizan, F., 2021. Progression Heuristics for Planning with Probabilistic LTL Constraints. In *Proc. of 35th AAAI Conference on Artificial Intelligence*. [Cited on page 90.]

Mingyu, H.; Ryan, W.; Sam, T., Toyer Felipe; Sylvie, T.; and Lexing, X., 2023. Action schema networks – ipc version. *Tenth International Planning Competition (IPC-10) Learning Track: Planner Abstracts*, (2023). [Cited on page 70.]

Mitchell, T.; Cohen, W.; Hruschka, E.; Talukdar, P.; Yang, B.; Betteridge, J.; Carlson, A.; Dalvi, B.; Gardner, M.; Kisiel, B.; et al., 2018. Never-ending learning. *Communications of the ACM*, 61, 5 (2018), 103–115. [Cited on page 88.]

Mnih, V.; Badia, A. P.; Mirza, M.; Graves, A.; Lillicrap, T.; Harley, T.; Silver, D.; and Kavukcuoglu, K., 2016. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, 1928–1937. PMLR. [Cited on page 87.]

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; et al., 2015. Human-level control through deep reinforcement learning. *nature*, 518, 7540 (2015), 529–533. [Cited on page 17.]

OpenAI, 2024. Chatgpt: A large language model. https://chat.openai.com/chat. Used for grammar revision and sentence polish. [Not cited.]

Rivlin, O.; Hazan, T.; and Karpas, E., 2020. Generalized planning with deep reinforcement learning. *arXiv preprint arXiv:2005.02305*, (2020). [Cited on pages 3, 21, 27, and 30.]

Rumelhart, D. E.; Hinton, G. E.; and Williams, R. J., 1986. Learning representations by back-propagating errors. *nature*, 323, 6088 (1986), 533–536. [Cited on pages 12 and 16.]

Scarselli, F.; Gori, M.; Tsoi, A. C.; Hagenbuchner, M.; and Monfardini, G., 2008. The graph neural network model. *IEEE transactions on neural networks*, 20, 1 (2008), 61–80. [Cited on pages 3 and 14.]

Schulman, J.; Levine, S.; Abbeel, P.; Jordan, M.; and Moritz, P., 2015. Trust region policy optimization. In *International conference on machine learning*, 1889–1897. PMLR. [Cited on page 18.]

Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; and Klimov, O., 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, (2017). [Cited on page 18.]

Shen, W.; Trevizan, F.; and Thiébaux, S., 2020. Learning domain-independent planning heuristics with hypergraph networks. In *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 30, 574–584. [Cited on pages 2 and 20.]

Shen, W.; Trevizan, F.; Toyer, S.; Thiébaux, S.; and Xie, L., 2019. Guiding Search with Generalized Policies for Probabilistic Planning. In *Proc. of 12th Annual Symp. on Combinatorial Search (SoCS)*. [Cited on page 21.]

Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; Van Den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; et al., 2016. Mastering the game of go with deep neural networks and tree search. *nature*, 529, 7587 (2016), 484–489. [Cited on pages 2, 32, and 88.]

Simon, S.; Blai, B.; and Geffner, H., 2023. Muninn. *Tenth International Planning Competition (IPC-10) Learning Track: Planner Abstracts*, (2023). [Cited on page 70.]

Sutton, R. S.; McAllester, D.; Singh, S.; and Mansour, Y., 1999. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12 (1999). [Cited on page 18.]

Toyer, S.; Thiébaux, S.; Trevizan, F.; and Xie, L., 2020. ASNets: Deep Learning for Generalised Planning. *Journal of Artificial Intelligence*, 68 (2020), 1–68. [Cited on pages 21 and 89.]

Toyer, S.; Trevizan, F.; Thiébaux, S.; and Xie, L., 2018. Action Schema Networks: Generalised Policies with Deep Learning. In *Proc. of 32nd AAAI Conference on Artificial Intelligence*. [Cited on pages 21 and 89.]

Trevizan, F.; Thiébaux, S.; and Haslum, P., 2017a. Occupation measure heuristics for probabilistic planning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 27, 306–315. [Cited on page 89.]

Trevizan, F.; Thiébaux, S.; Santana, P.; and Williams, B., 2017b. I-dual: Solving Constrained SSPs via Heuristic Search in the Dual Space. In *Proc. of 26th Int. Joint Conf. on AI (IJCAI)*. [Cited on page 90.]

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, Ł.; and Polosukhin, I., 2017. Attention is all you need. *Advances in neural information processing systems*, 30 (2017). [Cited on page 2.]

Velickovic, P.; Cucurull, G.; Casanova, A.; Romero, A.; Lio, P.; Bengio, Y.; et al., 2017. Graph attention networks. *stat*, 1050, 20 (2017), 10–48550. [Cited on page 15.]

Yoon, S.; Fern, A.; and Givan, R., 2008. Learning control knowledge for forward search planning. *Journal of Machine Learning Research*, 9, 4 (2008). [Cited on page 19.]