Learning for Planning Under Uncertainty: Predicting Features of SSPs with Novel Graph Representation

A thesis submitted for the degree
Bachelor of Advanced Computing
(Research and Development) (Honours)

24 pt Honours project, S1/S2 2024

By:
Mingjun (Jerry) Zhang

Supervisor:
Dr. Felipe W. Trevizan

Australian National University

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

October 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.

October, Mingjun (Jerry) Zhang

Acknowledgements

I would like to begin by expressing my profound gratitude to my supervisor, Dr. Felipe W. Trevizan. His patience, encouragement, and unwavering support have been instrumental in guiding me through this project. The time he dedicated to discussions and answering my questions provided invaluable insights that helped me navigate many challenging moments.

My heartfelt thanks also go to my friends, especially Jiawen Wang, who has been a constant source of support and care throughout the year. Additionally, I extend my gratitude to Dillon Chen and William Shen for their exemplary work on their research theses. Their comprehensive reports have been both educational and motivating, providing me with inspiration for structuring my own work.

I am deeply grateful to the CECC and ANU for their essential support and resources, which have been foundational to my academic journey. Special thanks to Pascal Bercher, who provided the LaTeX template and section guidelines for this project.

Finally, I wish to thank my family. Their unwavering belief in me and their encouragement have long been my foundation, and I hope to make them proud someday.

It is my Honour to spend the past 4 years in ANU. And it is my Honour to have all your support along the way. Hope you will enjoy this thesis.

Abstract

Probabilistic planning has long been a complex and challenging area within AI planning. Stochastic Shortest Path Problems (SSPs) provide an effective framework for scenarios where an agent seeks to reach a goal state with minimal expected cost under uncertain action outcomes. When dead ends are unavoidable, SSPs become multi-objective optimisation problems with two conflicting objectives: maximising the probability of reaching the goal and minimising the expected cost. The Minimising Cost given Maximum Probability (MCMP) criterion addresses this challenge by finding the policy with the minimum expected cost among those with the maximum probability of reaching the goal ($p^{\max}$). Achieving this criterion first requires the accurate computation of $p^{\max}$, yet few effective approaches have been proposed. Moreover, most state-of-the-art heuristic search algorithms for $p^{\max}$ remain reliant on determinisation and classical planning heuristics, resulting in weak upper bounds for the actual probability of reaching the goal. To our knowledge, no prior work has employed machine learning to develop heuristics for estimating $p^{\max}$.

In this study, we develop and implement two novel graph representations that encode SSPs in a lifted format meanwhile incorporating probabilistic information directly. These graph representations are designed to be either directly learned by Graph Neural Networks (GNNs) or transformed into feature spaces using the Weisfeiler-Lehman (WL) algorithm, enabling effective learning through Statistical Machine Learning (SML) methods. We evaluate these representations within two experimental frameworks. The first framework assesses the expressiveness of our graph representations, demonstrating that GNN and SML models achieve higher accuracy in predicting $p^{\max}$ when using features derived from our probabilistic graph structures, compared to state-of-the-art classical planning graphs in determinised SSPs. In the second framework, our learned model serves as a heuristic in i-dual—the only heuristic search algorithm for SSPs under MCMP criterion—and we show that models trained on our probabilistic graphs offer competitive performance against current $p^{\max}$ heuristics. Furthermore, they hold the potential to significantly surpass existing heuristics by strategically trading off optimality to provide a robust lower bound on $p^{\max}$.

Table of Contents

Chapter 1

Introduction

Artificial intelligence (AI) is a field of study focused on designing systems that exhibit intelligent behaviour. These systems aim to mimic human cognitive functions such as learning, problem-solving, perception, and reasoning. One goal from the topic of AI is to create agents that perceive their environment, take actions to achieve goals, and adapt rationally to changing conditions. [Russell and Norvig, 2016].

In recent decades, breakthroughs in numerous machine learning techniques have led modern AI to diverge into two key paradigms: model-free learners and model-based solvers. Model-free learners primarily rely on data and experience to learn functions that map inputs to expected outputs, whereas model-based solvers utilise explicit models of the world to compute solutions for a variety of tasks [Geffner, 2018].

1.1 Learning for Planning Under Uncertainty

One of the most common applications of model-based solvers is in the area of AI planning, which focuses on generating a sequence of actions or decisions that an agent can follow to achieve specific goals [Ghallab et al., 2004]. In classical planning, the environment is fully observable and deterministic, where each action leads to a single predictable outcome. In contrast, probabilistic planning involves uncertainty, where each action can result in multiple possible outcomes with associated probabilities [Puterman, 2014; Russell and Norvig, 2016].

Stochastic Shortest Path Problems (SSPs) [Bertsekas and Tsitsiklis, 1991] provide a practical framework for modelling probabilistic planning scenarios. In an SSP, an agent aims to reach a goal state with the lowest possible expected cost. The solution to an SSP is a policy that maps states to actions which ensures that the goal state is reached when starting from the initial state. However, it is impossible to always reach the goal when unavoidable deads are present. In this case, SSPs become multi-objective optimisation

problems where two potentially conflicting objectives arise: maximising the probability of reaching the goal and minimising the expected cost [Trevizan et al., 2017b]. To illustrate this conflict, consider the **box delivery problem**: a simple SSP where a box must be delivered to a city using one of two actions. A plane delivers the box with a 95% success rate at a cost of \$1000, while a car delivers the box with a 90% success rate at a cost of \$100. The dead end occurs if the delivery fails and the box is destroyed. The central conflict is whether paying an additional \$900 is worth increasing the probability of delivery from 90% to 95%.

Several criteria have been developed to manage such conflict. The Finite-Penalty criterion [Kolobov et al., 2012b] assigns a fixed and finite penalty D for actions leading to a dead end. The expected cost for the plane and car under Finite-Penalty would be $0.95 \times 1000 + 0.05D$ and $0.9 \times 100 + 0.1D$ respectively. However, this criterion has the drawback of labelling any state with a cost higher than D as a dead end. The requirement for domain-specific knowledge to select an appropriate D makes it unsuitable for domain-independent planning. The Maximum Probability (Max-Prob) criterion [Kolobov et al., 2011] avoids fixed penalties by ignoring costs and focusing solely on maximising the probability of reaching the goal. In the box delivery problem, Max-Prob would always select the plane but cannot distinguish between various planes with the same probability of success but different costs (if there exist a second plane). To address this, three approaches have been proposed: S^3P [Teichteil-Königsbuch, 2012], iSSPUDE [Kolobov et al., 2012a], and the Minimising Cost given Maximum Probability (MCMP) criterion [Trevizan et al., 2017b]. MCMP is the most efficient and robust one. It considers both the cost and risk of reaching dead ends, aiming to find the policy with the minimum expected cost among those that share the maximum probability of reaching the goal ($p^{\max}$). Solving an SSP under the MCMP criterion can be divided into two stages: (1) the **Max-Prob stage**, where the SSP is solved under the Max-Prob criterion to obtain $p^{\max}$ from the optimal Max-Prob policy; and (2) the **Min-Cost stage**, where the SSP is solved by considering only policies that can achieve $p^{\max}$. Both stages are addressed by solving two specific dual Linear Programming (LP) problems, derived from the original SSP [Trevizan et al., 2017b].

Heuristic search algorithms are a type of model-based solver that allow efficient navigation through vast solution spaces by directing exploration towards the most promising areas. This is achieved with the help of a heuristic function—a function that approximates the cost of reaching the goal from a given state [Bonet, 2001]. Heuristic search algorithms have long been among the most effective approaches for solving SSPs [Hansen and Zilberstein, 2001; Bonet and Geffner, 2003; Kolobov et al., 2011; Bonet and Geffner, 2012]. Meanwhile, i-dual [Trevizan et al., 2016] remains the only heuristic search algorithm capable of efficiently solving SSPs under the MCMP criterion. To solve an SSP under the MCMP criterion, i-dual requires two heuristic functions: one for the Max-Prob stage and one for the Min-Cost stage. The performance of i-dual as a heuristic search algorithm is directly tied to the quality of these heuristic functions. Some well-informed heuristic functions such as h^{pom} and h^{roc} [Trevizan et al., 2017a], have been developed for

the Min-Cost stage to estimate the expected minimum cost. However, most state-of-the-art heuristics for the Max-Prob stage are lacking as they do not consider probabilities and often rely on determinisation or classical planning heuristics. This leads to poor estimates of $p^{\max}$, with the only exception being Max-Prob pattern databases (Max-Prob PDB) [Klößner et al., 2021]. Even so, Max-Prob PDB often fails to outperform simple classical planning heuristics like $h^{\max}$ in many SSP domains [Bonet, 2001; Klößner et al., 2021].

Machine Learning (ML) has become a cornerstone of model-free learners where systems learn directly from data without relying on explicit models of the environment [Geffner, 2018]. Traditional Statistical Machine Learning (SML) methods such as Support Vector Machines (SVMs) [Vapnik, 2013] and Gaussian Process Regression (GPR) [Williams and Rasmussen, 2006] focus on identifying patterns in data to make predictions without the need for structured domain models [Bishop and Nasrabadi, 2006]. In recent years, advancements in computing power have driven Deep Learning (DL) approaches which leverages Neural Networks (NNs) to learn complex representations from data [Goodfellow, 2016]. This has revolutionised machine learning and significantly improving performance on many tasks that previously required handcrafted features and domain expertise, especially in fields like image recognition or natural language processing [Krizhevsky et al., 2012; Vaswani, 2017; OpenAI, 2023]. NNs have further evolved into specialised forms like Graph Neural Networks (GNNs) which enable learning from graph-structured data (e.g., social networks, molecular structures) [Zhou et al., 2020].

With the development of these tools, more researchers start to explore the application of machine learning for heuristic search. For instance, Action Schema Networks (ASNets) simulate heuristic search by defining a dedicated neural network architecture that learns generalised policies for both classical and probabilistic planning problems [Toyer et al., 2018, 2020], though their fixed receptive field limits their ability to handle long chains of reasoning. Recent state-of-the-art approaches focus on using machine learning to learn heuristics through graph representations of planning problems. Approaches like STRIPS-HGN [Shen et al., 2020] and the GOOSE framework with Lifted Learning Graph (LLG) or Instance Learning Graph (ILG) [Chen et al., 2023a, 2024] parse classical planning problems into graph representations that can be used as input to machine learning models, enabling these models to learn robust heuristics for guiding existing heuristic search algorithms. However, none of these approaches support SSPs. Meanwhile to the best of our knowledge, no attempts have yet been made to apply any machine learning techniques in developing heuristics for $p^{\max}$.

Our motivation stems from the key challenges in developing robust heuristics for $p^{\max}$, which is essential for efficiently solving SSPs with unavoidable deadends under the robust MCMP criterion. The lack of effective heuristics for $p^{\max}$, coupled with the limited application of ML in SSPs, inspires us to extend the concept of "learning for planning"—specifically, learning heuristic values to guide search—from the deterministic domain to the stochastic domain. The next section provides a detailed overview of our research objectives and contributions.

1.2 Research Objective and Contributions

The primary objective of this research is to explore how both traditional SML and modern DL methods can be applied to learn $p^{\max}$ for SSPs. Additionally, we aim to investigate how predictions from these models can be leveraged as heuristics which are capable of guide existing heuristic search algorithms to efficiently solve SSPs with unavoidable dead ends in the Max-Prob stage of MCMP. This thesis presents three main contributions:

1. **Novel Graph Representation for SSPs**: Building on insights from Younes and Littman [2004] and Chen et al. [2024], we develop two novel graph representations for SSPs under lifted representation: **Probabilistic Learning Graph Small** (PLGS) and **Probabilistic Learning Graph Large** (PLGL). Both graphs directly incorporate the probabilistic information from SSPs without determinisation. PLGS captures both domain-specific and problem-specific information within SSPs efficiently, while PLGL extends PLGS by introducing additional graph layers which allow more internal relationships within SSPs to be encoded.

2. **Learning Features of SSPs through PLG**: We extend the GOOSE framework [Chen et al., 2023a] to handle SSPs, allowing graph representations of SSPs to be utilised in training both GNNs and SML models. Our extended framework supports graph representation from both PLGS and PLGL which directly encodes SSPs; as well as any other graphs representing classical planning problems by using all-outcome determinisation on SSPs. This framework enables models to learn and predict $p^{\max}$ as well as other potential SSP features. We further theoretically prove that both PLGS and PLGL are strictly more expressive in learning SSP features within our framework compared to Instance Learning Graph (ILG)—the state-of-the-art graph for learning planning problems in deterministic environments. We evaluate our graph representations through a set of experiments which compares models that learn SSP features from PLG representations with those learning from ILG representations. Results demonstrate that PLG outperforms ILG at predicting $p^{\max}$ in terms of both effectiveness and generalisability.

3. **Learning $p^{\max}$ Heuristics to Guide Search**: We train both GNN and SML models using PLG to learn $p^{\max}$ and deploy the i-dual algorithm to incorporate modified predictions from these trained models as heuristic functions, which guide the search during the Max-Prob stage. We conduct another set of experiments which evaluates the performance of these heuristics against $h^{\max}$ in SSP domains where $h^{\max}$ remains the state-of-the-art. Results indicate that the generated heuristics can provide competitive performance against $h^{\max}$ in guiding i-dual for solving Max-Prob SSPs, with the potential to significantly outperform it by trading-off optimality with a strong lower bound.

1.3 Thesis Outline

This thesis is structured to effectively present our contributions:

- **Chapter 2** establishes the theoretical foundation, defining essential concepts in probabilistic planning and machine learning, alongside a summary of key notations.
- **Chapter 3** provides a comprehensive literature review, covering Max-Prob methods and machine learning approaches in planning, culminating in a discussion that connects the research motivation to our specific objectives.
- **Chapter 4** details the methodology addressing our research objectives, introducing the novel probabilistic graph representations PLGS and PLGL as first contribution. We also exploring how the extended GOOSE framework is adapted to learn $p^{\max}$ and effectively guide search.
- **Chapter 5** presents the empirical evaluation for contributions two and three, benchmarking PLGS and PLGL representations against existing graphs by predicting $p^{\max}$ and applying the learned heuristics to i-dual's Max-Prob stage, with a focus on evaluating improvements in search efficiency and robustness.
- **Chapter 6** provides a comprehensive conclusion, summarising the findings and implications of this research, while outlining potential directions for future work.

Chapter 2

Background

The aim of this chapter is to introduce foundational concepts while providing in-depth explanations of the theoretical definitions essential for understanding the research presented in this thesis. We begin with a comprehensive overview of AI planning in Section 2.1, focusing on Stochastic Shortest Path Problems (SSPs)—a key framework within probabilistic planning. We then formalise the definition of SSPs, discuss their key features and common challenges, with a highlight on optimisation criteria such as MCMP in Section 2.2, which provide important motivation for our work. We also show how classical planning problems can be seen as a special case of SSPs and provide formal representations for both. In Section 2.3, we explore heuristic search algorithms by presenting and explaining some heuristic functions and algorithms that will be employed in our framework. Finally, in Section 2.4, we shift focus to machine learning by explaining the definitions for regression tasks, GNNs, and the WL algorithm. Section 2.5 presents a table summarising the notations used throughout this chapter, which may help the audience to recall the key concepts in the remainder of the thesis.

2.1 AI Planning

Planning is the reasoning side of action. It involves explicit deliberation to choose and organise actions based on their expected outcomes in order to achieve a particular objective as effectively as possible [Ghallab et al., 2004].

AI planning, or Artificial Intelligence planning, is a sub-field of AI that focuses on developing algorithms and computational techniques to generate sequences of actions that achieve specific goals [Russell and Norvig, 2016]. This field originates from the pursuit of automating human-like capabilities, such as thinking ahead, strategising and deciding on a series of steps to solve complex problems. AI planning extends beyond robotics and virtual agents; its principles and algorithms are applied across various domains, in-

cluding business process management, video game design, emergency response planning and autonomous driving. Its wide range of applications underscores the essential role of planning in intelligent behaviour. In this thesis, our primary focus lies on a specific subset of AI planning known as *Probabilistic Planning*.

2.1.1 Probabilistic Planning: Assumptions

The problems being researched within this thesis operates under the following assumptions of probabilistic planning that distinguish them from problems other planning paradigms [Russell and Norvig, 2016]:

- **Finite:** The number of states, actions and observations are finite.
- **Static:** The world does not change on its own, only in response to the agent's actions.
- **Single Agent:** Only one agent acts in the environment, however this agent has access to multiple actions.
- **Stochastic:** Actions do not have deterministic outcomes. Instead, they result in probabilistic transitions between states represented by a probability distribution over possible successor states.
- **Fully Observable:** The agent has complete knowledge of the world's state.
- **Sequential:** The solution to a probabilistic planning problem is a policy that dictates the action to be taken in each state. The execution of such policy will result a sequence of decisions as well as a sequence of visited states.
- **Implicit Time:** Actions occur one after another instantaneously and there is no duration time.
- **Reach-Ability of Goals:** A policy is not always guaranteed to reach the goal. Each policy has an associated probability of successfully reaching the goal.
- **Cost Functions:** Actions may have associated costs and policy may be evaluated based on the total expected cost when executed from initial state.

2.1.2 Stochastic Shortest Path Problems

Stochastic Shortest Path problems (SSP) [Bertsekas and Tsitsiklis, 1991] provide a convenient mathematical framework for modeling problems under probabilistic planning assumptions as discussed in Section 2.1.1. In this thesis, we formally define SSPs following the definition from Trevizan and Veloso [2014] as shown below:

Definition 1 (Stochastic Shortest Path Problems (SSP))**.** An SSP is a tuple $\mathbb{S} = \langle S, s_0, S_g, A, P, C \rangle$, where:

- S is the finite set of states;
- $s_0 \in S$ is the initial state;
- $S_g \subseteq S$ is the non-empty set of goal states;
- A is the finite set of actions; $A(s) \subseteq A$ denotes the set of all actions applicable in state $s \in S$;
- $P : S \times S \times A \mapsto [0, 1]$ is the probabilistic transition function, where $P(s'|s, a)$ represents the probability that state $s' \in S$ is reached after applying action $a \in A$ in state $s \in S$;
- $C : S \times A \mapsto \mathbb{R}^+$ is the cost function, where $C(s, a)$ is the immediate cost incurred when applying action a in state s. This function must be defined for all s, a where there exists s' such that $P(s'|s, a) > 0$.

■

To aid in understanding SSPs, consider the **Box Delivery Problem** previously introduced in Section 1.1:

Example 1. *A box must be delivered to a city using one of two actions. A plane delivers the box with a 95% success rate at a cost of $1000, while a car delivers the box with a 90% success rate at a cost of $100. A dead end occurs if the delivery fails and the box is destroyed. The central dilemma is whether paying an additional $900 is worth increasing the delivery probability from 90% to 95%.*

We can now formalise the Box Delivery Problem as a simple SSP following Definition 1:

▷ $S = \{s_0, d, s_g\}$ where s_0 is the initial state where the box has not yet been delivered; d is the dead-end state where the box is destroyed; s_g is the goal state where the box is successfully delivered.

▷ $A = \{fly, drive\}$ represents the two possible actions.

▷ $P(s_g|s_0, fly) = 0.95$ and $P(d|s_0, fly) = 0.05$ represent the delivery outcomes by airplane.

▷ $P(s_g|s_0, drive) = 0.9$ and $P(d|s_0, drive) = 0.1$ represent the delivery outcomes by car.

▷ $C(s_0, fly) = 1000$ and $C(s_0, drive) = 100$ are the respective costs of delivery for each option in dollars.

Figure 2.1 visualises this simple SSP as a graph, where nodes represent states and edges represent actions. The edges are also labeled with the corresponding probabilities of the uncertain outcomes.

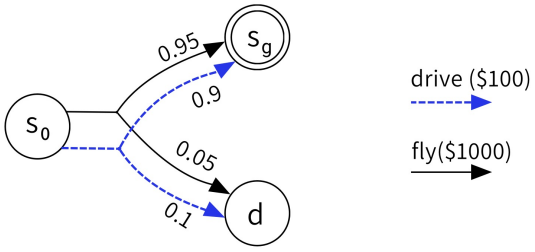

Figure 2.1: Box Delivery Problem SSP represented as a directed multi-edge graph. Where s_0, d, s_g are the initial/dead-end/goal state respectively. Edges represent two actions *drive* and fly with corresponding cost as \$100 and \$1000 respectively. The probability for each action outcome is labeled on the action edges.

In SSPs, an agent executes an action $a \in A(s)$ in discrete time steps on the current state $s \in S$. The chosen action a transitions the state from s to s' with probability $P(s'|s,a)$ and incurs a cost $C(s,a)$ is incurred. The agent always starts at s_0, and if a goal state $s_g \in S_g$ is reached, the problem is considered solved and no further actions are required. However, the agent might also reach a *dead-end state*, which occurs when it is impossible to achieve the goal from the current given state. In this setting, the agent must decide which action to apply at each step, and the solution to an SSP can be viewed as a decision strategy that selects actions based on states. For instance, in the Box Delivery Problem, should we choose to fly or drive at the initial state s_0?

We formally define such decision strategy as a policy:

Definition 2 (Solution of an SSP)**.** A solution to an SSP $\mathbb{S}$ is a policy π, which maps states to actions with corresponding probabilities [Trevizan and Veloso, 2014].

Formally, π can be viewed as either of the two functions: $\pi : S \times A \mapsto [0,1]$ or $\pi : S \to (A, [0,1])$ where:

- $\pi(s,a)$ denotes the probability that action $a \in A$ will be applied in state $s \in S$ when following policy π, with $\pi(s,a') = 0$ for all $a' \notin A(s)$.
- $\pi(s)$ represents a probability distribution over all actions in A, indicating the preference for applying actions in state s under policy π, also every action $a' \notin A(s)$ has corresponding probability of 0.
- $S^\pi \subseteq S$ denotes the domain of $\pi(s_0)$. S^π represents the set of all states reachable when following policy π from s_0.

■

Extending from Definition 2, we introduce several key terms for policies [Trevizan and Veloso, 2014] that will be used in subsequent sections:

- A policy π is **complete** if it is defined for all possible states, i.e., $S^\pi = S$.
- A policy π is **closed** if all non-goal states reachable from s_0 under π are included in S^π; otherwise, π is **partial**.
- A policy π is **deterministic** if it maps each state $s \in S^\pi$ to a single action $a \in A(s)$, i.e., for each $s \in S^\pi$, there exists one $a_0 \in A(s)$ such that $\pi(s, a_0) = 1.0$; otherwise, π is a **stochastic** policy.

Now that we have defined a policy as the solution to an SSP, we need a metric to compare the quality of different policies. Following the approach of [Trevizan and Veloso, 2014; Shen et al., 2019], we define the cost of a policy as a metric for determining the best policy. Combining Definitions 1 and 2, we formally define the cost of a policy as follows:

Definition 3 (Cost of an SSP Policy)**.** The cost of a policy π on $\mathbb{S}$ is defined through two functions:

- Equation 2.1 defines the state-value function $V^\pi : S \mapsto \mathbb{R}_{\geq 0}$, representing the expected cost of reaching a goal state from state $s \in S$ while following policy π.

$$V^\pi(s) = \begin{cases} 0 & \text{if } s \in S_g \\ \sum_{a \in A(s)} \pi(s, a) \cdot Q^\pi(s, a) & \text{otherwise} \end{cases} \tag{2.1}$$

- Equation 2.2 defines the action-state-value function $Q^\pi : S \times A \mapsto \mathbb{R}_{\geq 0}$, representing the expected cost of reaching a goal state from state $s \in S$ when action $a \in A(s)$ is applied and policy π is followed thereafter.

$$Q^\pi(s, a) = C(s, a) + \sum_{s' \in S} P(s' \mid s, a) \cdot V^\pi(s') \tag{2.2}$$

■

With the state-value function V^π defined in Definition 3, the optimal solution to an SSP can be described as a closed policy π^* that minimises the expected cost of reaching the goal from the initial state. The formal definition is as follows:

Definition 4 (Optimal Solution to an SSP)**.** The optimal solution to an SSP $\mathbb{S}$ is a closed policy π^* that minimises the expected cost of reaching the goal from the initial state. Formally: $\pi^* = \arg\min_\pi V^\pi(s_0)$ for all closed policy π on $\mathbb{S}$. ■

Note that π^* may not be unique, but at least one optimal policy is always deterministic [Trevizan et al., 2016]. Furthermore, if $\mathbb{S}$ is dead-end-free, we can extend Equations 2.3 and 2.4 to define the optimal value function V^* with the corresponding optimal action-state-value function Q^* as follows:

- Equation 2.3 defines the optimal value function $V^* : S \mapsto \mathbb{R}_{\geq 0}$, representing the minimum total expected cost to reach a goal state from any state $s \in S$ under any closed policy.

$$V^*(s) = \begin{cases} 0 & \text{if } s \in S_g \\ \min Q^*(s,a) & \text{otherwise} \end{cases} \tag{2.3}$$

- Equation 2.4 defines the optimal action-state-value function $Q^* : S \times A \mapsto \mathbb{R}_{\geq 0}$, representing the expected cost of reaching a goal state from state $s \in S$ if action $a \in A(s)$ is applied under any optimal policy π^*.

$$Q^*(s,a) = C(s,a) + \sum_{s' \in S} P(s' \mid s,a) \cdot V^*(s') \tag{2.4}$$

Equations 2.3 and 2.4 are also known as the Bellman Equations [Puterman, 2014]. We can consider the Bellman Equations as the generalised version of Equations 2.1 and 2.2 where the policy is deterministic. Moreover, any greedy policy with respect to the optimal value function V^* is an optimal policy π^* for $\mathbb{S}$. Specifically, such π^* is deterministic and can be obtained by replacing "min" with "arg min" in Equation 2.3 to select an action for each state [Trevizan et al., 2017b].

From Trevizan et al. [2016], we know that at least one optimal policy is always deterministic. Since we only require one optimal policy as the solution to the SSP, from this section onwards, we will focus solely on deterministic policies. Hence, every occurrence of π within the rest of this chapter will be treated as a deterministic policy unless explicitly stated otherwise. It is important to note that the nature of probabilistic planning is not affected in deterministic policies, since the actions still have stochastic effects.

We denote by Π the set of all (deterministic) policies for $\mathbb{S}$. Based on deterministic policies, we define the following term:

Definition 5 (Trace)**.** A trace $T_{\pi,s}$ is a sequence of states $s_1, s_2, \ldots$ visited when following $\pi \in \Pi$ from state s. We denote the i-th state of $T_{\pi,s}$ as $T^i_{\pi,s}$ and require that $P(T^{i+1}_{\pi,s} \mid T^i_{\pi,s}, \pi(T^i_{\pi,s})) > 0$ for all pairs $T^i_{\pi,s}, T^{i+1}_{\pi,s} \in T_{\pi,s}$ [Trevizan et al., 2017b].

A trace T can be either finite or infinite:

- **Finite trace:** The last state s of T is either a goal state (i.e., $s \in S_g$) or there is no applicable action in s (i.e., $A(s) = \emptyset$). For a finite trace T, its probability is $P(T) = \prod_{i=1}^{|T|-1} P(T^{i+1}|T^i, \pi(T^i)$ and its cost is $C(T) = \sum_{i=1}^{|T|-1} C(T^i, \pi(T^i))$.
- **Infinite trace:** Infinite traces occur when the execution of π enters a cycle that never reaches the goal, getting trapped indefinitely. Since action costs are strictly positive, the cost of any infinite trace is infinite. We also refer SSPs that allows such π as *cyclic* SSPs or SSPs with *circles*.

■

We denote by $\mathcal{T}_{\pi,s}$ the set of all traces of π from s. Notice that $\mathcal{T}_{\pi,s}$ may be an infinite set (e.g., if an action prescribed by π has a positive probability of looping). We also partition $\mathcal{T}_{\pi,s}$ into the set of traces that reach the goal, $\mathcal{T}^G_{\pi,s}$, and its complement, $\mathcal{T}^{\text{DE}}_{\pi,s}$, i.e., the set of traces that reach dead ends. We can again refer back to Box Delivery Problem as an illustrative example, and define a policy π_d that always chooses to drive: the set of goal traces becomes $\mathcal{T}^G_{\pi_d,s_0} = \{(s_0, s_g)\}$; while the set of dead-end traces is $\mathcal{T}^{\text{DE}}_{\pi_d,s_0} = \{(s_0, d)\}$. While every trace in $\mathcal{T}^G_{\pi,s}$ is finite (as its last state must be a goal state) $\mathcal{T}^{\text{DE}}_{\pi,s}$ can have infinite traces, i.e., traces that loop indefinitely without reaching a goal state.

With the trace notation from Definition 5, we can easily represent a dead-end-free SSP by assuming there exists a policy π such that $\mathcal{T}^{\text{DE}}_{\pi,s_0} = \emptyset$. This is a common assumption for many previous results and algorithms for solving SSPs [Hansen and Zilberstein, 2001; Kolobov et al., 2011; Bonet and Geffner, 2003; Howard, 1960]. However, when dead ends are unavoidable (for example, in the Box Delivery Problem, where there is always a non-zero probability of reaching the dead-end state d regardless of the policy), the Bellman Equations (2.3) become ill-defined.

We address this issue in Section 2.2 by introducing several criteria that have been proposed to characterise SSPs with dead ends. Before discussing these criteria, we first provide an overview of classical planning problems and the concept of domain-independent planning in the following sections.

Classical Planning

Classical planning is another essential subset of AI planning. Most assumptions for problems under classical planning remain the same as outlined in Section 2.1.1, except that actions are no longer subject to uncertain outcomes. Therefore, if a goal can be reached, the probability of success is guaranteed to be 1 [Russell and Norvig, 2016].

We can formally define classical planning problems (problems under classical planning assumptions) as a special case of SSPs as follows:

Definition 6 (Classical Planning Problems)**.** A classical planning problem is a special type of SSP $\mathbb{S}^c = \langle S, s_0, S_g, A, P_c, C\rangle$, where:

- S, s_0, S_g, A, and C are the same as in Definition 1.
- $P_c : S \times S \times A \mapsto \{0, 1\}$ is a deterministic transition function, where $P_c(s'|s, a)$ indicates whether the state $s' \in S$ is the next state after applying action $a \in A$ in state $s \in S$. For each state-action pair (s, a), if $a \in A(s)$, there is exactly one $s' \in S$ such that $P_c(s'|s, a) = 1$; otherwise, $P_c(s'|s, a) = 0$.

■

In a classical planning problem $\mathbb{S}^c$, for each state $s \in S$, every applicable action $a \in A(s)$ always leads to a single successor state. A policy π for $\mathbb{S}^c$ can therefore be simplified as

a deterministic mapping from each state-action pair (s, a) to its corresponding successor state s', where $s \in S$, $a \in A(s)$, and $P_c(s'|s, a) = 1$. Since we can continue applying the policy to each successor state until a goal is reached (if such a policy exists), a solution to a classical planning problem $\mathbb{S}^c$ can be further simplified to a sequence of n actions, $\eta_{\pi,s_0} = a_1, a_2, \ldots, a_n$, chosen by the policy π starting from s_0, which results in the trace $T_{\pi,s_0} = s_0, s_1, s_2, \ldots, s_n$ where $s_n \in S_g$. T_{π,s_0} includes only the states visited by sequentially executing η_{π,s_0} from s_0. As the solution to $\mathbb{S}^c$, η_{π,s_0} is commonly referred to as a *plan* for $\mathbb{S}^c$ [Ghallab et al., 2004]. The optimal plan η^*_{π,s_0} is a plan that minimise the total cost of transitioning from s_0 to s_g in $\mathbb{S}^c$.

One way to translate an SSP $\mathbb{S} = \langle S, s_0, S_g, A, P_c, C\rangle$ into a classical planning problem $\mathbb{S}^c = \langle S, s_0, S_g, A', P'_c, C'\rangle$ is through *all-outcome determinisation* [Teichteil-Königsbuch et al., 2011] as described below:

- S, s_0, S_g remains the same.
- for possible action outcome in $\mathbb{S}$, we create a new deterministic action in $\mathbb{S}^c$ that achieve the same outcome. Formally: for every $a \in A$ with $P(s_j|s_i, a) > 0$, we add a' to A' where $P'_c(s_j|s_i, a') = 1$ and $C'(s_i, a') = C(s_i, a)$

However, such transformed problem often completely change the scope of the original SSP. This is because the process of determinisation often oversimplify the original SSP by removing or modifying crucial probabilistic information. For example in Box Delivery Problem, all-outcome determinisation will create two actions based on the original *drive* action where: $drive_1$ will always deliver the box and $drive_2$ will always destroy the box.

The computational representation of AI-Planning problems is crucial for algorithms to effectively solve them. While the representations for SSPs in Definition 1 and classical planning problems in Definition 6 offer an intuitive and mathematically solid structure, they may not be efficient enough as input for solvers, as the state space can grow exponentially with increasing problem complexity. For example, in the Box Delivery Problem, simply adding two more boxes for delivery creates six additional states representing whether each box has been successfully delivered, resulting in a total combination of 2^3 states. To address this issue, we introduce the concept of domain-independent planning representations.

2.1.3 Domain-Independent Planning Representation

Domain-Independent Planning refers to a methodology where planning algorithms are designed to function independently of the specifics of any particular domain [Russell and Norvig, 2016]. The central idea is to abstract domain-specific details by separating environment-specific descriptions (e.g., domain settings, action types, object types) from problem-specific descriptions (e.g., initial states, goals, objects). This separation enables a formalisation that applies across different problem domains, allowing algorithms to be constructed in a domain-independent manner by taking problem inputs in such a formalisation [Ghallab et al., 2004].

We can apply a similar separation idea to the representation of classical planning problems as defined in Definition 6: the initial state s_0 and the set of goal states S_g can be viewed as problem-specific descriptions, while the rest of the information—such as the set of states, actions, transition functions, and cost functions—can be considered domain-specific. By integrating the concept of domain-independent planning with the motivation to avoid enumerating all possible instances in S, we can represent a classical planning problem in a lifted format that encodes only the first-order information of domain-specific knowledge through the use of predicates and action schemas [Haslum et al., 2019].

Definition 7 (Lifted Classical Planning Problems)**.** A lifted classical planning problem is a tuple $\mathbb{S}^{cL} = \langle \mathbb{D}, O, s_0, g \rangle$ where:

- O is the set of objects in the problem, objects are allowed to have different types.
- $\mathbb{D} = \langle \mathcal{P}, \mathcal{A} \rangle$ is the domain for $\mathbb{S}^{cL}$ where:
  - $\mathcal{P}$ is a set of first-order predicates representing the environment settings.
  - A predicate $P \in \mathcal{P}$ has a tuple of parameters $P(x_1, \ldots, x_{n_P})$ for $n_P \in \mathbb{N}$, we call such predicate an n_p-ary predicate.
  - A predicate where all the parameters are assigned with objects $o \in O$ is called a (grounded) proposition. (It is possible for a predicate to have no parameter therefore default grounded)
  - $\mathcal{A}$ is a set of action schemas.
  - An action schema $a \in \mathcal{A}$ is a tuple $\langle \Delta(a), \text{pre}(a), \text{add}(a), \text{del}(a), \text{cost}(a) \rangle$ where $\text{pre}(a) = \langle \text{pre}_+(a), \text{pre}_-(a) \rangle$ are the positive/negative preconditions. $\Delta(a)$ is a set of parameter variables. $\text{pre}_+(a), \text{pre}_-(a), \text{add}(a), \text{del}(a)$ are sets of predicates from $\mathcal{P}$ and $\text{cost}(a)$ is a cost function, they all need to be instantiated with either parameter variables or objects in $\Delta(a) \cup \mathcal{O}$. Similarly to predicates, an action schema with $n = |\Delta(a)|$ parameter variables is an n-ary action schema.
- s_0 a set of propositions representing the initial state.
- g a set of propositions representing the partial goal state.

■

In this thesis, both $\mathbb{S}^{cL}$ (Definition 7) and $\mathbb{S}^{c}$ (Definition 6) represent the same classical planning problem, but in different formats. We demonstrate this by showing how a lifted planning problem $\mathbb{S}^{cL} = \langle \langle \mathcal{P}, \mathcal{A} \rangle, O, s_0, g \rangle$ can be transformed into a classical planning problem $\mathbb{S}^{c} = \langle S', s'_0, S'_g, A', P'_c, C' \rangle$ as follows:

- **States** S': The predicate set $\mathcal{P}$ defines the set of all possible states S' with objects from O. A state $s' \in S'$ is a set composed of any valid combination of the propositions instantiated by combining predicates $P \in \mathcal{P}$ with objects $o \in O$.

- **Actions** A': The set of objects O can also instantiate action schemas $\mathcal{A}$. All possible instantiated actions form the set A'.
- **Initial State** s'_0: The initial state s_0 directly corresponds to the initial state s'_0 in the classical planning model.
- **Goal States** S'_g: The goal states S'_g comprise all superset states of the partial goal state set g.
- **Transition Function** P'_c **and Cost Function** C': In the classical model, the transition function P'_c maps a state and an action to their resulting state with a corresponding cost described by C'. If we denote $a' \in A'$ as the action instantiated from action schema $a \in \mathcal{A}$ with the corresponding parameter set $O' \subseteq (\Delta(a) \cup \mathcal{O})$, then $P'_c(s'|s, a')$ and $C'(s, a')$ can be defined where:
  - $P'_c(s'|s, a') = 1$ and $C'(s, a') = \text{cost}(a)(O')$ for all combinations of $s, s' \in S' \times S'$ where s is a superset of $\text{pre}_+(a)(O')$ not containing any element within $\text{pre}_-(a)(O')$, with the corresponding $s' = s \cup \text{add}(a)(O') \setminus \text{del}(a)(O')$, indicating that action a' can transition s to s' at a cost $C'(s, a')$.
  - $P'_c(s'|s, a') = 0$ for all other combinations of $s, s' \in S' \times S'$, indicating that action a' cannot transition s to s'.

The intuition is to view states as set of propositions, actions as instantiated action schemas. While every action schema is a mapping that transit a state satisfying the precondition into its successor state. A state satisfies the precondition if it contains all the required(positive) propositions and does not contain any banned(negative) proposition. The successor state is obtained by removing deleting(negative) effect propositions from the original state and append the adding(positive) effect propositions onto it.

Modern planning languages such as the Planning Domain Definition Language (PDDL) [Haslum et al., 2019] are defined based on the lifted format for classical planning problems (Definition 7). PDDL allows problems to be parsed as input to specific algorithms and has become the standard language for many planning competitions, such as the International Planning Competition (IPC) [Vallati et al., 2015]. The Probabilistic Planning Domain Definition Language (PPDDL) [Younes and Littman, 2004] extends PDDL to allow for the parsing of SSPs in their lifted formats. We introduce the lifted representation for SSPs in Chapter 4.1, where we also provide examples of both a lifted classical planning problem instance and a lifted SSP instance.

2.2 Criteria of Stochastic Shortest Path Problems

As briefly introduced in Section 1.1, when unavoidable dead ends are present, an SSP inherently becomes a multi-objective optimisation problem with two conflicting objectives: maximising the probability of reaching the goal and minimising the expected cost. To better align with intuition, these objectives can be thought of as **"minimising cost"**

by reducing the expected cost as much as possible; or **"minimising risk"** by increasing the probability of reaching the goal as much as we can. The Box Delivery Problem serves as an illustrative one: is it worth paying an additional $900 on airplanes to mitigate the remaining 5% risk of delivery failure from cars?

At the end of Section 2.1.2, we have further discussed that most of the algorithms developed for SSPs share one common assumption: the SSPs they take as input to solve are dead-end-free. Unfortunately, this is not the case in many real-life scenarios. For example we can never avoid the chance of destroying the box along the delivery in the Box Delivery Problem no matter what vehicle we select.

2.2.1 Finite-Penalty Criterion

One approach, called the *Finite-Penalty* method, solves this trade-off by allowing the agent to reach the goal from any state by applying a special high-cost action, as formally defined below:

Definition 8 (Finite-Penalty Criterion)**.** The Finite-Penalty criterion removes dead ends in a Stochastic Shortest Path (SSP) problem by introducing a fixed dead-end penalty $D \in \mathbb{R}^+$ and incorporating an additional "give-up" action [Kolobov et al., 2012b; Trevizan et al., 2017b]. Solving an SSP $\mathbb{S}$ under the Finite-Penalty criterion is equivalent to solving a transformed SSP $\mathbb{S}' = \langle S, s_0, S_g, A', P', C' \rangle$, where for each state $s \in S$:

- $A'(s) = A(s) \cup \{\text{give-up}\}$;
- $C'(s, \text{give-up}) = D$;
- $C'(s, a) = C(s, a)$ for $a \in A(s)$;
- $P(s_g \mid s, \text{give-up}) = 1$ for an arbitrary $s_g \in S_g$.

It is important to note that the transformed SSP $\mathbb{S}'$ contains no dead ends and can therefore be solved using any standard SSP-solving algorithm. ■

However, as briefly discussed in Section 1.1, any state where the expected cost of reaching the goal exceeds the dead-end penalty D will be considered as a dead end, regardless of the state's maximum probability of reaching the goal. For instance, consider the Box Delivery Problem. If we set $D = \$10$ and solve it using the Finite-Penalty criterion, the optimal policy would always select the "give-up" action, ignoring the options to either drive or fly. This reliance on domain knowledge to select an appropriate value of D poses a challenge, as it is not typically feasible to obtain a good D on every problem domain, whilst the requirement for manually selecting D restrict the performance of solvers to automate across various domains. To avoid such requirement for domain knowledge, an alternative approach focuses on directly maximising the probability of reaching the goal regardless of the cost.

2.2.2 Max-Prob Criterion

Unlike the Finite-Penalty criterion, the *Max-Prob* criterion addresses the conflict between "risk" and "cost" by focusing exclusively on minimising "risk."

Definition 9 (Max-Prob Criterion)**.** The Max-Prob criterion focuses on finding a policy π that maximises the probability of reaching any goal state $s_g \in S_g$; as opposed to π^* which minimises the expected cost [Kolobov et al., 2011].

- In our trace notation, this can be represented as finding a π from the set of policies $\text{argmax}_{\pi \in \Pi} P(\mathcal{T}^G_{\pi,s_0})$, where *goal probability* $P(\mathcal{T}^G_{\pi,s_0}) = \sum_{T \in \mathcal{T}^G_{\pi,s_0}} P(T)$ represent the expected probability for a policy to reach goal states from s_0.
- We denote this *maximum goal probability* as $p^{\max}$, i.e., $p^{\max} = \max_{\pi \in \Pi} P(\mathcal{T}^G_{\pi,s_0})$.
- We refer to all policies with a goal probability equal to $p^{\max}$ as *Max-Prob Policies* and use Π^{MP} to represent the set of all Max-Prob policies, i.e., $\Pi^{\text{MP}} = \left\{\pi \in \Pi \mid P(\mathcal{T}^G_{\pi,s_0}) = p^{\max}\right\}$ [Trevizan et al., 2017b].

■

The optimal solution of $\mathbb{S}$ under the Max-Prob criterion can be obtained by solving a modified problem, denoted as $\mathbb{M}$. In this modified problem $\mathbb{M}$, everything remains the same as in the original SSP $\mathbb{S}$ except that the objective shifts to maximising the probability of reaching the goal when following a policy π, i.e., $\max P(\mathcal{T}^G_{\pi,s_0})$ as shown in Definition 9. This maximisation process can be accomplished by solving a modified version of the Bellman Equations. Specifically, we adapt the Bellman Equations from Equation 2.3 and 2.4 to suit the Max-Prob SSP $\mathbb{M}$ as follows:

- Equation 2.5 defines the probabilistic value function $V^p : S \mapsto [0, 1]$, representing the maximum probability of reaching the goal from $s \in S$. It lies within the range of $[0, 1]$, even if $\mathbb{S}$ has circles.

$$V^p(s) = \begin{cases} 1 & \text{if } s \in S_g \\ \max Q^p(s, a) & \text{otherwise} \end{cases} \tag{2.5}$$

- Equation 2.6 defines the probabilistic action-state-value function $Q^p : S \times A \mapsto [0, 1]$, representing the maximum probability of reaching a goal state from $s \in S$ if action $a \in A(s)$ is applied.

$$Q^p(s, a) = \sum_{s' \in S} P(s' \mid s, a) \cdot V^p(s') \tag{2.6}$$

Similar to what we have done in Section 2.1.2 for Bellman Equations, this modification allows us to obtain a deterministic policy π^p by replacing "max" with "arg max" in Equation 2.5 to select an action for each state. If the probabilistic value function on

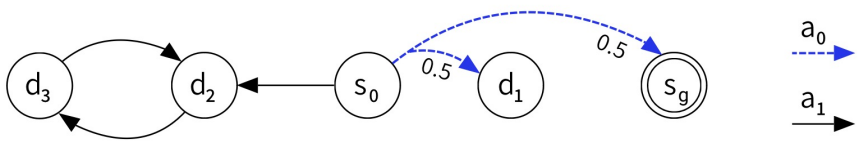

Figure 2.2: Example of an SSP represented as a directed multi-edge graph. Where s_0, s_g are the initial/goal state respectively, d_i are dead-end states. Edges represent two actions a_0 and a_1 with omitted cost. The probability for each action outcome is labeled on the action edges. $p^{\max} = 0.5$.

initial state $V^p(s_0)$ is globally optimal, i.e., $V^p(s_0) = p^{\max}$, then we have $\pi^p \in \Pi^{\text{MP}}$ as a Max-Prob policy.

Applying the above method to the Box Delivery Problem, the optimal policy will always choose the airplane to deliver the box with the optimal maximum probability $V^p(s_0) = p^{\max} = 0.95$, without considering the cost.

However, solve the Bellman equations for $\mathbb{M}$ (Equations 2.5 and 2.6) may produce several non-optimal fixed-point solutions for problems with circles. For instance, consider the SSP listed in Figure 2.2 [Trevizan et al., 2017b], where there is only one goal state s_g and three dead-end states d_1, d_2, d_3 with two applicable actions a_0 and a_1 (an edge without float label indicate deterministic outcome). It may produce the following non-optimal fixed-point solution for $\mathbb{M}$: $V^p(d_1) = 0$ and $V^p(s_g) = V^p(s_0) = V^p(d_2) = V^p(d_3) = 1$ when $V^p(d_2) = V^p(d_3) = 1$ (probabilistic value function for states within the circles are not initialised properly). Since solving Equation 2.5 under this scenario will always guide the optimal policy to select a_1 at s_0. We discuss the works that address this issue in the related works sections later in Chapter 3.

In addition to the non-optimal fixed-point solutions, the Max-Prob criterion has another drawback: all policies in Π^{MP} are equally good for Max-Prob, regardless of their expected cost since the consideration for "cost" is removed completely. To better illustrate this drawback, consider an **Extended Box Delivery Problem**:

Example 2. *An extension of the Box Delivery Problem where an additional action "cheap-fly" is introduced. This discounted airplane delivery option offers the same 95% success rate as the regular airplane but at a significantly lower cost of only \$100 (equivalent to the cost of car). For clarity, we visualise the SSP for this extended problem in Figure 2.3 following Definition 1.*

Clearly, the cheap airplane is always the best choice to select in Extended Box Delivery Problem. Yet the Max-Prob criterion fails to distinguish between a policy that always selects the normal airplane and one that always selects the cheap airplane since they both share $V^p(s_0) = p^{\max} = 0.95$. The S^3P criterion [Teichteil-Königsbuch, 2012] addresses

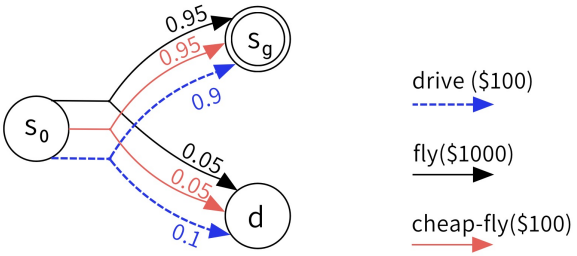

Figure 2.3: Extended Box Delivery Problem SSP represented as a directed multi-edge graph. Where s_0, d, s_g are the initial/dead-end/goal state respectively. Edges represent three actions $drive$, fly and $cheap-fly$ with corresponding cost as \$100, \$1000 and \$100 respectively. The probability for each action outcome is labeled on the action edges.

this issue by distinguishing Max-Prob policies with their expected cost across all traces that successfully reach the goal. The formal definition for S^{3}P is given below:

Definition 10 (S^{3}P Criterion)**.** The S^{3}P criterion for solving an SSP $\mathbb{S}$ focuses on finding a Max-Prob policy π that minimises the expected cost across all traces that successfully reach the goal when following π from s_0.

Formally, it finds a Max-Prob policy π where:

$$\pi^* = \arg \min_{\pi \in \Pi^{\text{MP}}} \mathbb{E}[C(T) \mid T \in \mathcal{T}^G_{\pi,s_0}]. \tag{2.7}$$

Solving $\mathbb{S}$ under S^{3}P usually contains three stages (in some algorithms, removal of dead ends can be done on-the-fly within Max-Prob stage):

- **Max-Prob Stage:** compute $V^p(s)$ for every reachable state $s \in S$.
- **Remove dead ends Stage**: translate $\mathbb{S}$ into a dead-end-free P*-SSP with $V^p(s)$ by removing effects of actions that lead to a dead end state.
- **Min-Cost Stage:** solve P*-SSP to obtain π^* as described in Equation 2.7.

■

The S^{3}P criterion has also been proven to be equivalent to the *iSSPUDE* criterion under SSPs defined in Definition 1 [Kolobov et al., 2012a; Trevizan et al., 2017b]. Both approaches prioritise "risk" while still taking "cost" into account. However, a key limitation of both S^{3}P and iSSPUDE is their failure to account for the expected cost on dead-end traces, i.e., $T \in \mathcal{T}^{\text{DE}}_{\pi,s}$. When executing the optimal policies derived from these criteria, dead ends may still be encountered; incurring costs that were not considered in the optimisation and resulting in a gap between the computed optimal value and the actual expected cost when the policies are implemented.

One of the state-of-the-art criteria for SSPs, the *Min-Cost given Max-Prob* (MCMP) criterion, addresses this limitation by enhancing the S^{3}P criterion to also account for dead ends to some extent [Trevizan et al., 2017b].

2.2.3 Min-Cost given Max-Prob Criterion

Before introducing the MCMP criterion, we first define a truncation function $\psi : T \mapsto T$ following from Trevizan et al. [2017b]:

$$\psi(s_i s_{i+1} \dots) = \begin{cases} s_i & \text{if } |T| = 1 \text{ or } P(T^G_{\pi,s_i}) = 0 \\ s_i \psi(s_{i+1} \dots) & \text{otherwise} \end{cases} \tag{2.8}$$

Intuitively, the truncation function ψ truncates a trace after the first encountered dead-end state (if any). Notice that after truncation, $\psi(T)$ is always a finite trace since either a goal is reached; or a state s_i with zero probability of reaching the goal ($P(T^G_{\pi,s_i}) = 0$) is encountered and subsequent states $s_{i+1}s_{i+2}\dots$ are ignored. Moreover, $\psi(T) = T$ for all $T \in \mathcal{T}^G$, meaning that the truncation function does not modify traces that successfully reach the goal.

MCMP utilises the truncation function from Equation 2.8 and extends the idea from S^{3}P criterion to allow consideration of expected cost across the entire set of traces, especially, it considers the dead-end traces. The formal definition of MCMP is given below:

Definition 11 (MCMP Criterion)**.** The Minimising Cost given Maximum Probability to reach the goal (MCMP) criterion for solving an SSP $\mathbb{S}$ focuses on finding a Max-Prob policy π that minimises the expected cost across all traces when following π from s_0. For dead-end traces, the expected cost calculation only focuses on the parts before a dead-end state is encountered.

Formally, it finds a Max-Prob policy π^* where:

$$\pi^* = \arg\min_{\pi \in \Pi^{\text{MP}}} \mathbb{E}[C(\psi(T)) \mid T \in \mathcal{T}^G_{\pi,s_0}]. \tag{2.9}$$

Unlike S^{3}P, solving $\mathbb{S}$ under MCMP contains two stages:

- **Max-Prob Stage:** solve the SSP under Max-Prob criterion $\mathbb{M}$ to obtain $p^{\max}$ (derived from the solution of $\mathbb{M}$: a Max-Prob policy $\pi \in \Pi^{\text{MP}}$).
- **Min-Cost Stage:** solve Equation 2.9 with $p^{\max}$ to obtain optimal policy.

■

To illustrate the differences between S^{3}P and MCMP, recall the Extended Box Delivery Problem. If we calculate the expected cost of a policy that always selects the action *cheap-fly* on s_0 under both criteria: S^{3}P will return \$95 while MCMP will return \$100. Intuitively, S^{3}P ignores the cost of failed delivery, assuming that no cost will be incurred if the box is not delivered successfully; meanwhile MCMP aligns better with actual

expect cost by also considering the cost of failures. This difference in how paths leading to dead-ends are treated further eliminates the need for transforming the original SSP into a dead-end-free SSP before computation.

In addition to mitigating the gap between computed and actual expected cost, computation for solutions under MCMP has been proven to be up to one order of magnitude faster compared to solutions computed under S^3P and iSSPUDE criteria [Trevizan et al., 2017b]. The main reason is that MCMP avoids the need to compute goal probability V^p for all reachable states $s \in S^\pi$ as required by S^3P and iSSPUDE. Instead, it combines the technique of linear programming over dual space [d'Epenoux, 1963] with heuristic search algorithms [Pearl, 1984] to efficiently find $p^{\max}$ through heuristic search within the space of dual variables that describe SSP policies. It then performs another heuristic search on the space of dual variables describing only Max-Prob policies to obtain the final solution. In the next section, we will discuss how linear programming over dual space can be used to formulate the MCMP criterion. We will then explore heuristic search algorithms that operate on this dual linear programming formulation of MCMP in Section 2.3.

2.2.4 Linear Programming over Dual Space for MCMP

Linear Programming (LP) is a powerful optimisation method used to minimise or maximise a linear objective function, subject to a set of linear equality and inequality constraints. In this subsection, we assume the audience is familiar with the concepts of LP problems and LP problems over dual space (dual LP). For a more comprehensive understanding of LP, we refer readers to Bertsimas and Tsitsiklis [1997] and Vanderbei [1998].

Dual LPs have long been employed to solve Markov Decession Process problems (MDP) [d'Epenoux, 1963; Altman, 1999]. An SSP can be viewed as a specific type of MDP. To solve an SSP $\mathbb{S}$ with dual LP, it is firstly transformed into its dual LP representation, where the dual variables $y_{s,a}$ are known as *policy occupation measures*: $y_{s,a}$ representing the expected number of times an action $a \in A(s)$ is executed in state s. LP 1 gives an example for a dual LP representation of $\mathbb{S}$. Occupation measures y can be thought as another representation for a policy π, and an optimal policy π^* can be derived from the optimal occupation measures y^* by solving the dual LP formulation of the original SSP [Trevizan et al., 2016].

Recall from Definition 11 that solving $\mathbb{S}$ under the MCMP criterion involves two stages:

- **Max-Prob Stage:** Solve $\mathbb{M}$ to obtain $p^{\max}$.
- **Min-Cost Stage:** Solve Equation 2.9 using $p^{\max}$ to obtain the optimal policy.

We now demonstrate how two dual LP formulations, using policy occupation measures as variables, can be constructed to address both stages. Specifically, we define two LPs LP 1 and LP 2 where: solving LP 2 addresses the Max-Prob stage and solving LP 1 addresses the Min-Cost stage [Trevizan et al., 2017b].

$$
\begin{aligned}
&\min_{y} \sum_{s\in S, a\in A} y_{s,a} C(s,a) && \text{s.t. (C1) - (C6)} \\
&y_{s,a} \geq 0 && \forall s \in S, a \in A(s) \text{(C1)} \\
&in(s) = \sum_{s'\in S, a\in A(s')} y_{s',a} P(s|s',a) && \forall s \in S \text{(C2)} \\
&out(s) = \sum_{a\in A(s)} y_{s,a} && \forall s \in S \setminus S_g \text{(C3)} \\
&out(s) - in(s) \leq 0 && \forall s \in S \setminus (S_g \cup \{s_0\}) \text{(C4)} \\
&out(s_0) - in(s_0) \leq 1 && \text{(C5)} \\
&\sum_{s_g\in S_g} in(s_g) = p^{\max} && \text{(C6)}
\end{aligned}
\tag{LP 1}
$$

We denote the LP 1 [Trevizan et al., 2017b] formulation of $\mathbb{S}$ as $\mathbb{LP}_1(\mathbb{S})$. A feasible solution to $\mathbb{LP}_1(\mathbb{S})$ is a combination of dual variables y that satisfies all of its constraints, meanwhile every feasible solution y can be interpreted as a policy π of $\mathbb{S}$. The objective function of is represented as $V_{LP1}(y) = \sum_{s\in S, a\in A} y_{s,a} C(s,a)$. The optimal solution of $\mathbb{LP}_1(\mathbb{S})$ is the y^* that minimises V_{LP1} among all feasible solutions, with corresponding optimal objective value being V^*_{LP1}. Moreover, Trevizan et al. [2017b] has proved Proposition 1.

Proposition 1. *Solving $\mathbb{LP}_1(\mathbb{S})$ allow us to directly solve $\mathbb{S}$ in the Min-Cost stage of MCMP. That is, the optimal solution y^* of $\mathbb{LP}_1(\mathbb{S})$ allows us to directly derive an optimal policy π^* for $\mathbb{S}$ under MCMP criterion.*

Intuitively, LP 1 can be interpreted as a water flow problem, where each state $s \in S$ represents a sink. Each feasible action $a \in A(s)$ for state s functions as a one-to-N pipe connecting state s to other states s' for which $P(s'|s,a) > 0$. Each unit of flow through this pipe incurs a cost of $C(s,a)$. Here, $y_{s,a}$ denotes a non-negative distributive water flow (C1) leaving state s through pipe a (C3). Once it exits state s, the flow $y_{s,a}$ is distributed to each state s' connected to s by a; specifically, each state s' receives a portion of the flow $y_{s',a}P(s'|s,a)$ (C2). For each non-goal state, the sum of incoming flows must be at least as great as the sum of outgoing flows, signifying that each non-goal state sink $s \in S \setminus S_g$ allow water flow to be "stored" (C4). Meanwhile, the starting state has an inflow of 1.0 (C5) representing expected time for s_0 to happen is 1. For all goal states $s_g \in S_g$, the sum of incoming flows must total $p^{\max}$ (C6), ensuring that the expected time (probability) to reach the goal is $p^{\max}$ for any feasible y. The objective is to minimise the total incurred cost of all flows.

To provide a better intuition on the flow interpretation of 1 we introduce a simple SSP $\mathbb{S}_{flow}$ in Example 3 and visualise it in Figure 2.4.

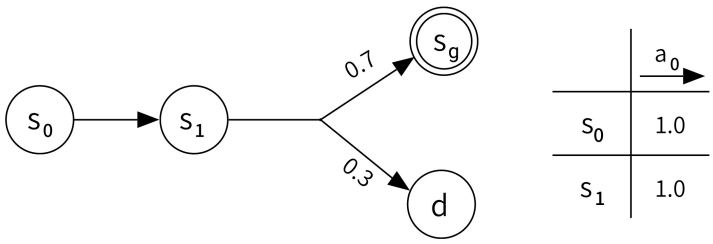

Figure 2.4: A simple SSP $\mathbb{S}_{flow}$ represented as a directed multi-edge graph. Where s_0, s_1, d, s_g are the initial/intermediate/dead-end/goal state respectively. Edges represent a single action a_0 with constant cost of 1. The probability for each action outcome is labeled on the action edges. In the water flow interpretation for LP 1 encoding of this SSP, we can consider the states as sinks and edges as pipes.

Example 3. *A simple flow problem SSP* $\mathbb{S}_{flow} = \langle S, s_0, S_g, A, P, C \rangle$ *under Min-Cost stage of MCMP, where:*

- $p^{max} = 0.7$
- *the states are* $S = \{s_0, s_1, s_g, d\}$*;*
- *there's only a single action* a_0*;*
- $P(s_1|s_0, a_0) = 1.0$ *for* s_0*.*
- $P(s_g|s_1, a_0) = 0.7$*,* $P(d|s_1, a_0) = 0.3$ *for* s_1*.*
- $C(s, a) = 1.0$ *for both* s_0 *and* s_1

Solving the Flow Problem SSP using LP 1 gives us: the optimal solution $y^*_{s_0,a_0} = y^*_{s_1,a_0} = 1.0$; the flows entering each state: $s_0 = 0.0$, $s_1 = 1.0$, $d = 0.3$, $s_g = 0.7$; the flows leaving each state: $s_0 = 1.0$; $s_1 = 1.0$; $d = 0.0$; $s_g = 0.0$; the flows remaining in each state: $s_0 = 0.0$; $s_1 = 0.0$; $d = 0.3$; $s_g = 0.7$. from the previously calculated y^*, We can observe that the policy π^* obtained from y^* is deterministic and always selects a_0 on both s_0 and s_1, meanwhile π^* has a goal probability $P(\mathcal{T}^G_{\pi^*,s_0}) = p^{\max} = 0.7$ with an expected cost of 2.0, confirming that it is indeed the optimal solution for $\mathbb{S}_{flow}$. This is achieved by (C6), which make sure no flow can be preserved in the sink s_0 and s_1. In another word, every flow is "pushed forward" in order to get a total flow of 0.7 entering s_g as required by $p^{\max}$.

One important point to note is that constraint (C6) ensures LP 1 only consider solutions that can be interpreted as Max-Prob policies, i.e., $\pi \in \Pi^{\text{MP}}$. This vastly reduce the solution space and hence vastly increase the speed for solving LP 1. However, this constraint requires the pre-computation of $p^{\max}$, which is obtained as a solution from the Max-Prob stage of MCMP. This is equivalent to solving $\mathbb{M}$ ($\mathbb{S}$ under the Max-Prob criterion) as outlined in Definitions 9 and 11.

Following the work of [Trevizan et al., 2017b; Altman, 1999], there also exists an LP formulation for solving the Max-Prob stage. This can be achieved by making slight modifications to LP 1, resulting in the construction of LP 2.

$$\begin{aligned} &\max_{y} \sum_{s_g \in S_g} in(s_g) && \text{s.t. (C1) - (C3), (C7) - (C8)} \\ &out(s) - in(s) = 0 && \forall s \in S \setminus (S_g \cup \{s_0\}) \text{(C7)} \\ &out(s_0) - in(s_0) = 1 && \text{(C8)} \end{aligned} \tag{LP 2}$$

We denote the LP 2 [Trevizan et al., 2017b] formulation of $\mathbb{S}$ as $\mathbb{LP}_2(\mathbb{S})$. A feasible solution to $\mathbb{LP}_2(\mathbb{S})$ is a combination of dual variables y that satisfies all of its constraints and a feasible y can also be interpreted as a policy π of $\mathbb{S}$. The objective function of is represented as $V_{LP2}(y) = \sum_{s_g \in S_g} in(s_g)$. The optimal solution of $\mathbb{LP}_2(\mathbb{S})$ is the y^* that maximises V_{LP2} among all feasible solutions, with corresponding optimal objective value being V^*_{LP2}

Intuitively, LP 2 does not allow flow to be kept in non-goal state sinks anymore, that is, the flow can either be stored in a goal state sink or being trapped in an infinite loop (C7, C8). LP 2 also modifies the objective to maximise the total flow entering all goal state states. Altman [1999] has proved that the objective value for a feasible solution of LP 2 y is equivalent to the goal probability of π interpreted from y. Trevizan et al. [2017b] also has proved Proposition 2.

Proposition 2. *Solving $\mathbb{LP}_2(\mathbb{S})$ is equivalent to solving $\mathbb{M}$ (therefore also Max-Prob stage of MCMP). That is:*

1. *The optimal objective value V^*_{LP2} of $\mathbb{LP}_2(\mathbb{S})$ is equivalent to p^{max} of $\mathbb{S}$.*
2. *The optimal solution y^* of $\mathbb{LP}_2(\mathbb{S})$ allows us to directly derive a Max-Prob policy $\pi \in \Pi^{MP}$.*

With Proposition 1 and 2, we can summarise the preceding background as follows: to solve an SSP with unavoidable dead ends, we must decide between minimising cost or risk. MCMP is a robust approach that prioritises risk minimisation while still considering cost. It operates in two stages: the first stage solves the SSP under the Max-Prob criterion, and the second stage uses the solution from the Max-Prob criterion to find the Max-Prob policy with Min-Cost. Proposition 2 shows that stage 1 can be achieved by solving LP 2, and Proposition 1 demonstrates that stage 2 can be achieved by solving LP 1. As a result, **the challenge of solving an SSP with unavoidable dead ends under MCMP is reduced to solving two LP problems**.

Furthermore, Trevizan et al. [2016, 2017b] propose how this process can be efficiently managed using *i-dual*, the only heuristic search algorithms available for efficiently solving SSP under MCMP criterion. In the next section, we introduce the concept of heuristic search while addressing i-dual in the end of Section 2.3.2.

2.3 Heuristic Search

In this thesis, we assume the audience to be familiar with the fundamental concepts and notations of graph theory, including the definition for graph, weighted graph, weighted multi-edge graph and the directed version of those graphs. For a more comprehensive reading, we refer the audience to Gross et al. [2018] and Trudeau [2013].

Solving a planning task through search can be understood as a process of iteratively exploring a graph, where each node represents an unexplored state and the edges denote the relationships between these states. During exploration, we keep expanding the unexplored nodes and adding new unexplored nodes into the graph until a goal state is found [Russell and Norvig, 2016].

Heuristic search, on the other hand, is a widely used method that employs heuristic evaluation functions to estimate the value of each node before formally evaluating it. This approach guides the search process more efficiently by concentrating exploration on promising areas [Bonet, 2001; Russell and Norvig, 2016]. Many state-of-the-art planners, such as (L)RTDP [Bonet and Geffner, 2003] and iLAO* [Hansen and Zilberstein, 2001], rely on heuristic search to efficiently find solutions in large and complex domains.

In this section, we first define the concept of a *heuristic* as an evaluation function. We then explaining the concept of *admissibility*, and provide two example heuristics: *Max-Cost* and *Fast-Forward*. Finally, we explore the widely adopted heuristic search algorithm $A*$ and its variant *i-dual* which allows heuristic search to be performed in dual space.

2.3.1 Heuristic Functions

Definition 12 (Heuristic Function)**.** A heuristic function $h : S \mapsto \mathbb{R}$ provides an estimate of the cost to reach the goal from a given state. This estimate is called the *heuristic value*. In this thesis, both "heuristic" and "heuristic function" refer to this definition. ■

In the rest of the section, we will explain heuristic in minimisation problem where the objective is to minimise the cost unless otherwise stated. By utilising a heuristic, search planners can prioritise paths that appear more promising (with lower heuristic values), potentially saving time by reducing the number of state expansions and improving the planning efficiency.

An important property of a heuristic is its **admissibility**. A heuristic for a minimisation problem is considered admissible if it never overestimates the true cost of reaching the goal from a given state. Formally, this is expressed as $\forall s, h(s) \leq h^*(s)$, where $h^*(s)$ is the true cost function for s in a minimisation problem. However, in a maximisation problem where the objective is to maximise instead minimise the cost (e.g. in Max-Prob SSP, we are trying to maximise the expected probability to reach the goal), an admissible heuristic never underestimate the true cost, i.e., $\forall s, h(s) \geq h^*(s)$ in maximisation

problems [Pearl, 1984]. Admissibility ensures that the heuristic provides optimistic estimates, which guarantees an optimal solution in some search algorithms, such as A* [Hart et al., 1968] or i-dual [Trevizan et al., 2016].

Conversely, inadmissible heuristics may overestimate the cost; however, they are often more informative than admissible ones therefore enables more aggressive pruning of nodes during the search. This can make them faster, though at the risk of missing the optimal solution [Geffner and Haslum, 2000]. Thus, while admissible heuristics ensure finding optimal paths in algorithms like A*, non-admissible heuristics can increase efficiency but without the same guarantees [Bonet, 2001].

Two well-known examples of these heuristics in classical planning are the admissible heuristic *Max-Cost* ($h^{\max}$) [Bonet, 2001] and the inadmissible heuristic *Fast-Forward* (h^{FF}) [Hoffmann and Nebel, 2001]. Both can be computed via a relaxed plan, which is obtained efficiently from the relaxed version of the planning problem $\mathbb{S}^{cL+}$: a $\mathbb{S}^{cL}$ where the delete effects of actions are ignored. $h^{\max}$ estimates cost by returning the highest cost required to achieve any individual proposition in the relaxed plan. In contrast, h^{FF} estimates cost by counting the number of actions in the relaxed plan [Russell and Norvig, 2016]. As we have discussed above, h^{FF} is typically more informative than $h^{\max}$ and perform better when guiding search, but does not guarantee optimal solution in some algorithms like A* or i-dual.

2.3.2 Heuristic Search Algorithms

Heuristic search algorithms use heuristic functions to estimate costs before expanding new nodes, enabling the search to focus on promising regions and thus accelerating the solving process [Russell and Norvig, 2016]. Best-First Search (BFS) is a general strategy that extends traditional breadth-first and depth-first search methods by employing a priority queue to determine the order of node expansion. The selection of nodes for expansion is typically based on an evaluation function, often referred to as the *cost* of a node [Russell and Norvig, 2016]. In BFS on a directed graph $G = \langle V, A \rangle$, with an unexpanded node set $N \subseteq V$, the node $n \in N$ selected for expansion is the one with the lowest evaluation function value, defined as $\arg\min_n f(n)$.

The search process in a classical planning problem $\mathbb{S}^c$ can be conceptualised as a path-finding task in a directed graph, where nodes represent states and edges correspond to actions. A well-known example of BFS applied to a classical planning problem $\mathbb{S}^c$ is the A* search algorithm [Hart et al., 1968]. A* uses an evaluation function defined in Equation 2.10:

$$f(s) = g(s) + h(s) \tag{2.10}$$

where $g(s)$ is the cost to reach state s from s_0, and $h(s)$ is the heuristic estimate of the cost to reach the goal from s.

The intuition behind A* is that it avoids expanding paths already known to be expensive. A* is guaranteed to terminate and is complete, meaning it will always find a plan if one

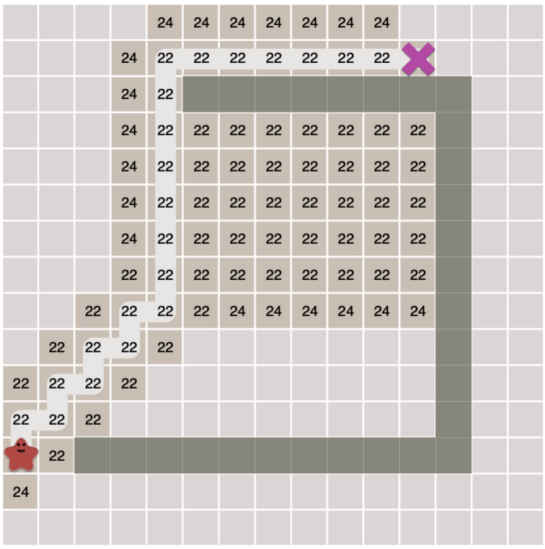

Figure 2.5: An example from Patel [2014] for a simple path finding problem solved using A* with an admissible heuristic. The objective is to move the star to the X block with minimum steps. The heuristic is the Manhattan distance with the X block. Under such heuristic, A* is guaranteed to reach optimal solution. The optimal path is highlighted in white, explored nodes (blocks) are marked with corresponding f value as defined in Equation 2.10.

exists, regardless of how uninformative the heuristic is. Furthermore, it has been proven that if the heuristic function h is admissible, then A* is guaranteed to return the optimal plan [Russell and Norvig, 2016]. Figure 2.5 provides an illustrative example where A* finds an optimal solution using Manhattan distance as the heuristic function.

One of the state-of-the-art heuristic search algorithms for solving SSPs is i-dual [Trevizan et al., 2016]. It is a variant of A* which takes input: an SSP $\mathbb{S}$, an LP formulation $\mathbb{LP}$, and a corresponding heuristic function h that estimates its objective function V_{LP}. It performs heuristic search in a manner similar to A* and solve $\mathbb{LP}(\mathbb{S})$. Meanwhile, i-dual has been proven to have the same properties as A* [Trevizan et al., 2016] as outlined below in Proposition 3 and 4.

Proposition 3. *i-dual guarantees the optimal solution V_L^* for an $\mathbb{LP}(\mathbb{S})$ if the input heuristic function h is admissible with respect to the optimal objective value V_L^* of $\mathbb{LP}(\mathbb{S})$ [Trevizan et al., 2016].*

Proposition 4. *i-dual guarantees termination and is complete, meaning it will always find a solution for $\mathbb{LP}(\mathbb{S})$ if one exists, regardless of the quality of the input heuristic [Trevizan et al., 2016].*

Recall Propositions 1 and 2 establish that solving SSPs under the MCMP criterion can be accomplished by solving LP 1 and LP 2. Trevizan et al. [2017b] further demonstrates that

this can be efficiently achieved by using i-dual to solve LP 1 and LP 2, with corresponding heuristic functions provided as stated in Proposition 5 and Theorem 1.

Theorem 1. *For an SSP $\mathbb{S}$, solving $\mathbb{LP}_2(\mathbb{S})$ using i-dual with any heuristic $h^{\mathcal{P}}$ for p^{max} provides a lower bound on p^{max}. Furthermore, if $h^{\mathcal{P}}$ is admissible, then we can directly obtain p^{max} (which solves the first(Max-Prob) stage of MCMP) [Trevizan et al., 2017b].*

Please notice that the formal proof can be found in Trevizan et al. [2017b] as a result of combining Proposition 3 and 4.

Proposition 5. *For an SSP $\mathbb{S}$, solving $\mathbb{LP}_1(\mathbb{S})$ using i-dual with p^{max} and any admissible cost heuristic h provides a solution to the second (Min-Cost) stage of MCMP [Trevizan et al., 2017b].*

To the best of our knowledge, i-dual is the only heuristic search algorithm developed so far that can solve SSPs under the MCMP criterion with unavoidable dead ends. This concludes the final part of our background discussion on model-based approaches. We will now shift our focus to machine learning techniques from a model-free perspective.

2.4 Machine Learning

Machine learning (ML) is another sub-field of AI that focuses on the development of algorithms which is able to increase accuracy on performing certain tasks through learning from input data [Theodoridis, 2015]. Tasks that ML models perform can be broadly classified into two categories:

- **Classification**: The task here is to predict discrete labels or categories. For example, determining whether an email is spam or not is a classification problem.
- **Regression**: In regression tasks, the goal is to predict continuous numerical values. An example would be predicting house prices based on features like size, location, and number of rooms.

In this thesis, we will focus on regression tasks. We first introduce what a *regression task* is, with a brief highlight on traditional models built for those tasks in Section 2.4.1. We then shift our focus to *Graph Neural Networks* and explore how they learn features from graphs in Section 2.4.2. Section 2.4.3 brings an end to background by introducing the well-known Weisfeiler-Lehman Algorithms.

2.4.1 Regression Tasks

A regression task is a type of supervised learning task where a model learns to predict a target from a set of training samples, each containing input data with its corresponding expected prediction result [Theodoridis, 2015]. We provide its formal definition below:

Definition 13 (Regression Task)**.** A regression task involves estimating a function $\mathcal{F}$ that maps input variables x (features) to a continuous output variable y (target values).

Given a set of training data $\{(x_n, y_n)\}_{n=1}^N$, where $x_n \in \mathbb{X}$ ($\mathbb{X}$ can represent any type of data) and $y_n \in \mathbb{R}$, the objective is to find a function $\mathcal{F}$ such that, for any new input feature x^* with its corresponding target y^*, the function outputs $\hat{y} = \mathcal{F}(x^*)$, where $\hat{y}$ is a good prediction of y^*. ■

Mean Squared Error (MSE) is a common metric used to evaluate the performance of a regression model. It measures the average of the squares of the differences between the true values y and the predicted values $\hat{y}$. Mathematically, it is defined as:

$$\text{MSE}(\{(x_i, y_i)\}_{i=1}^N, \mathcal{F}) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \mathcal{F}(x_i))^2 \tag{2.11}$$

where:

- $\{(x_i, y_i)\}_{i=1}^N$ is the evaluation dataset with N observations.
- x_i is the feature data for observation i, with y_i being the true target.
- $\mathcal{F}$ is the regression function to be evaluated.

MSE penalizes larger errors more than smaller ones because the errors are squared, making it sensitive to outliers. MSE is commonly applied as an evaluation strategy for quantifying the model's performance during both *training* (the process for the algorithm to learn $\mathcal{F}$ from input training data) and *testing* (the process for evaluating the learned model on new datasets). The lower the MSE, the better the model predicts the data.

Statistical Machine Learning (SML) is a subfield of machine learning that emphasizes the use of statistical models and inference to make predictions or decisions based on data [Bishop and Nasrabadi, 2006]. The core idea is to create models that generalize well to new, unseen data by understanding the underlying probability distributions and relationships within the data. Two common SML approaches for regression tasks are *Support Vector Regression* and *Gaussian Process Regression*.

Support Vector Regression (SVR) [Smola and Schölkopf, 2004] extends Support Vector Machines [Cortes, 1995] to regression tasks. SVR uses kernels to map input data into a higher-dimensional space and finds the best-fitting line (hyperplane) within a specified margin of error. It minimizes prediction errors while controlling model complexity by penalizing deviations beyond this margin. However, SVR will not be implemented in this thesis.

Gaussian Process Regression (GPR) [Williams and Rasmussen, 2006] is a non-parametric, probabilistic model used for regression tasks. It assumes that the underlying function $\mathcal{F}$ that maps input x to output y is a sample from a *Gaussian Process*. A *Gaussian Process* (GP) [Adler, 2010] is defined as a collection of random variables, any finite subset of which follows a multivariate Gaussian distribution. A GP is fully specified by its mean function $\mu(x)$ and covariance function $k(x, x')$, which captures relationships between input points. Formally, it is written as:

$$\mathcal{F}(x) \sim \mathcal{GP}(\mu(x), k(x, x')) \tag{2.12}$$

where:

- $x_i \in \mathbb{R}^d$ is a d-dimensional vector.
- $\mathcal{GP}(\mu(x), k(x, x'))$ is a GP with mean function $\mu(x)$ and covariance function $k(x, x')$.
- Both $\mu(x)$ and $k(x, x')$ are often defined using kernel functions [Bishop and Nasrabadi, 2006].

Given a set of training data (x, y), the GP regression model uses Bayesian inference to learn the distribution of $\mathcal{F}$ that is most likely to have generated the data. This involves computing the posterior distribution of $\mathcal{F}$ given the data, which is defined as:

$$p(\mathcal{F} \mid x, y) = \frac{p(y \mid x, \mathcal{F})p(\mathcal{F})}{p(y \mid x)}$$

where $p(y \mid x, \mathcal{F})$ is the likelihood of the data given the function $\mathcal{F}$; $p(\mathcal{F})$ is the prior distribution of $\mathcal{F}$; and $p(y \mid x)$ is the marginal likelihood of the data.

Once the posterior distribution of $\mathcal{F}$ has been learned, the model can make predictions at new test points x^* by computing the posterior predictive distribution $\mathcal{F}^*$ as below:

$$p(\mathcal{F}^* \mid x^*, y, x) = \int p(\mathcal{F}^* \mid x^*, \mathcal{F})p(\mathcal{F} \mid y, x)\, d\mathcal{F}$$

where the mean and variance of the final predictive distribution $\mathcal{F}^*$ provide the predicted target $\hat{y}$ for x^* and the estimate of uncertainty respectively. In this thesis, we will use $\mathcal{FG}_\Theta(x)$ to denote a GPR model with parameters Θ that performs regression task on vector x.

2.4.2 Graph Neural Networks

In this thesis, we assume the audience to be familiar with the basis of graph theory. However for clarification purpose, we will provide definition for three different types of undirected graphs that is going to be used along the rest of the thesis [Gross et al., 2018].

Definition 14 (Graph)**.** An undirected *graph* $G = \langle V, E \rangle$ is a mathematical structure consisting of:

- V, a finite set of *nodes*;
- $E \subseteq V \times V$, a finite set of *edges*.

We denote n as the number of nodes $|V|$ and m as the number of edges $|E|$. ■

Definition 15 (Weighted Graph)**.** A *weighted graph* $G = \langle V, E, \omega_V, \omega_E \rangle$ is an undirected graph where each node and edge is associated with corresponding weight. The node weight and edge weight are defined by two mapping functions: $\omega_V : V \mapsto \mathbb{X}$ and $\omega_E : E \mapsto \mathbb{X}$ where $\mathbb{X}$ can be any type. ■

A weighted graph allows nodes and edges to carry attributes, providing additional information for many practical applications [Gross et al., 2018]. Notice that in this thesis, we will mainly be using a simplified version of it with only node features.

Definition 16 (Weighted Multi-Edge Graph)**.** A *weighted multi-edge graph* or *weighted multi-graph* $G = \langle V, E, \omega_V, \omega_E, \kappa, \Sigma \rangle$ is an undirected graph where each node is associated with a weight and each edge is associated with a weight and a colour, where:

- $E \subseteq V \times V$, a finite **list** of *edges*;
- The node weight and edge weight are defined by two mapping functions: $\omega_V : V \mapsto \mathbb{X}$ and $\omega_E : E \mapsto \mathbb{X}$ where $\mathbb{X}$ can be any type;
- The edge colour (or edge type) is defined by a mapping function $\kappa : E \mapsto \Sigma$, where Σ is a set of colours (usually represented as a set of integers).

■

In the remainder of this thesis, we will use term *multi-graph* to refer to a weighted multi-edge graph. A multi-graph allows both nodes and edges to carry attributes, enhancing expressiveness. It is commonly used in graph-related machine learning tasks. A key difference between multi-graphs and general graphs is that multi-graphs allow multiple edges with different colours between the same pair of nodes. The *coloured neighbourhood* of a node $v \in V$ in a multi-graph G, under edge colour ι, is the set of all neighbours of v connected through an edge with colour ι, denoted as $\mathcal{N}_\iota(v) = \{u \in V \mid e = \langle u, v \rangle = \langle v, u \rangle \in E, \kappa(e) = \iota\}$. The *total neighbourhood* of a node $v \in V$ in a multi-graph G is the set of all neighbours of v, denoted as $\mathcal{N}(v) = \bigcup_{\iota \in \Sigma} \mathcal{N}_\iota(v)$.

A *Graph Neural Network* (GNN) is a type of neural network designed to operate on graph-structured data. Unlike traditional neural networks that work on regular data structures like grids (images) or sequences (text), GNNs are tailored to capture the complex, non-Euclidean relationships present in graphs. It can also be viewed as an optimisable transformation on all attributes of the graph that preserves graph symmetries. GNNs are highly applicable given that many structures can be represented as graphs [Zhou et al., 2020]. In this subsection, we assume that the readers are familiar with mainstream deep learning concepts such as neural networks and their operation.

A *Message Passing Neural Network* (MPNN) is a specific type of GNN [Gilmer et al., 2017]. Given a weighted graph from Definition 15 $G = \langle V, E, \omega_V, \omega_E \rangle$, where $\omega_V : V \mapsto \mathbb{R}^d$ and $\omega_E : E \mapsto \mathbb{R}^e$ are feature mapping functions that map each node and edge to their corresponding features, an MPNN iteratively updates node embeddings of G locally in one-hop neighbourhoods with the general message-passing (Equation 2.13) defined as following:

$$h_u^{(t+1)} = \varphi^{(t)} \left(h_u^{(t)}, \mathrm{agg}_{v \in \mathcal{N}(u)}^{(t)} f^{(t)} \left(h_u^{(t)}, h_v^{(t)}, e_{v,u}^{(t)} \right) \right) \tag{2.13}$$

where in the t-th iteration or layer of the network:

- $h_u^{(t)} \in \mathbb{R}^{F(t)}$ is the embedding of node u, with $h_u^{(0)} = \omega_V(u)$;

- $e_{v,u}^{(t)} \in \mathbb{R}^{D(t)}$ is the embedding of the edge (v, u), with $e_{v,u}^{0} = \omega_E(e_{v,u})$;

Here, $\varphi^{(t)}$ and $f^{(t)}$ are arbitrary almost everywhere differentiable functions, and $\text{agg}^{(t)}$ is typically a differentiable permutation-invariant function acting on sets of vectors, such as sum, mean, or component-wise max [Chen et al., 2023a].

For graphs without edge features $G = \langle V, E, \omega_V \rangle$ (which is going to be the major focus within this research), equation 2.13 can be simplified to equation 2.14:

$$h_u^{(t+1)} = \varphi^{(t)} \left(h_u^{(t)}, \text{agg}_{v \in \mathcal{N}(u)}^{(t)} f^{(t)} \left(h_u^{(t)}, h_v^{(t)} \right) \right). \tag{2.14}$$

To retain the "neural network" component within an MPNN, it is typical for φ or f to incorporate learnable parameters; for example, φ can be modelled by a feed-forward network. To generate a graph representation for an input, MPNNs commonly aggregate all node embeddings after a series of message-passing updates using a graph readout function Φ, which is typically a differentiable, permutation-invariant function. Specifically, for an MPNN with T total iterations and message-passing equation 2.14, the final graph embedding is given by $h_G = \Phi_{u \in V}(h_u^{(T)})$. To perform a regression task, this graph embedding h_G is often passed through one or more fully connected layers to produce the final output [Chen et al., 2023a]. In this thesis, we will use $\mathcal{FM}_\Theta(G)$ to denote a MPNN regression model with parameters Θ that performs regression task on graph G.

Before moving onto the next section, we also introduce *one-hot encoding*. One-hot encoding is a widely used technique in ML where categorical variables are converted into binary vectors. Each category is represented by a vector containing all zeros except for a single 1 at the index corresponding to that category [Rumelhart et al., 1986]. In GNNs, one-hot encoding is frequently applied in the node feature weight function ω_V, allowing the network to effectively distinguish and process features of nodes from different categories.

2.4.3 Weisfeiler-Lehman Algorithms

In graph theory, two graphs G_1 and G_2 are said to be *isomorphic* if there exists a one-to-one correspondence (or bijection) between their vertex sets that preserves adjacency. This means that the structure of connections between vertices in G_1 can be perfectly mapped onto G_2, making them structurally identical. Formally, graphs $G_1 = \langle V_1, E_1 \rangle$ and $G_2 = \langle V_2, E_2 \rangle$ are isomorphic if there exists a bijection $\varphi : V_1 \mapsto V_2$ such that $(u, v) \in E_1$ if and only if $(\varphi(u), \varphi(v)) \in E_2$ [Gross et al., 2018].

The k-Weisfeiler-Lehman (k-WL) algorithm was originally developed to provide tests for graph isomorphism. It iteratively refines node colours by considering tuples of k nodes and their neighbouring structures, allowing the algorithm to distinguish between an increasing number of non-isomorphic graphs as k increases. The $k+1$-WL algorithm subsumes the k-WL algorithm, meaning it can distinguish a broader class of graphs

[Weisfeiler and Leman, 1968]. However, the complexity of k-WL grows exponentially with k, making it less efficient for large graphs [Cai et al., 1992].

The 1-WL algorithm, also known as the colour refinement algorithm, is a special case of the k-WL algorithm, operating on individual nodes. Algorithm 1 illustrates the process of the 1-WL algorithm on a simplified weighted graph $G = \langle V, E, \omega_V \rangle$, following Definition 15, where edges are unweighted and $\omega_V : V \mapsto \Sigma_V$ is a *node colouring function* that assigns each node a specific colour. Such a graph G is referred to as a *Coloured Graph*. Additionally, we use $\{\!\{\ldots\}\!\}$ to denote a multiset, which allows multiple occurrences of the same element.

Algorithm 1: 1-WL Algorithm

Input : Coloured Graph $G = \langle V, E, \omega \rangle$ where $\omega : V \mapsto \mathbb{N}$ is the node colouring function; A max iteration number J

Output: A multiset of node colours seen over all iterations.

1 **Procedure** `1-WLAlgorithm`(G, J)

2 $\quad \omega^0(v) \leftarrow \omega(v), \forall v \in V$ *// Initial coloring for all nodes*

3 $\quad$ **for** $j = 1, \ldots, J$ **do**

4 $\quad\quad$ **for** $v \in V$ **do**

5 $\quad\quad\quad \omega^j(v) \leftarrow \text{hash}(\omega^{j-1}(v), \{\!\{\omega^{j-1}(u) \mid u \in \mathcal{N}(v)\}\!\})$

6 $\quad$ **return** $\bigcup_{j=0,\ldots,h} \{\!\{\omega^j(v) \mid v \in V\}\!\}$

The 1-WL algorithm begins by taking the graph G and a predefined maximum number of iterations J as input. It iteratively updates each node's colour by hashing the current colour of the node with the multiset of its neighbours' colours, continuing this process until the colours stabilise or the maximum number of iterations is reached (Lines 2–5). The algorithm returns a multiset of node colours seen over all iterations (line-6). If two different multisets are produced for two graphs G_1 and G_2, then the graphs are non-isomorphic. However, if the algorithm outputs the same multisets for two graphs, we cannot conclusively determine whether they are isomorphic.

The 1-WL algorithm is computationally efficient, requiring at most $|V|-1$ iterations for a graph with node set V [Weisfeiler and Leman, 1968]. One of its key applications is in the construction of graph kernels, where the final multiset of node colours is transformed into a feature vector, enabling the graph to be used in machine learning tasks with statistical machine learning (SML) models [Shervashidze et al., 2011; Chen et al., 2024]. This approach has been shown to be effective, as **MPNNs have proved to be are at most as powerful as colour refinement** [Xu et al., 2018]. In this thesis, we employ a variant of the 1-WL algorithm to perform regression tasks on graphs, which will be detailed later in Chapter 4.

This concludes the last part of background. In the next section, we provide a summary of all the notations used throughout the entire chapter in Table 2.1.

2.5 Summary of Notations

Table 2.1: Summary of notations.

Notation	Description
$\mathbb{S} = \langle S, s_0, S_g, A, P, C \rangle$	A Stochastic Shortest Path Problem (SSP)
S	A set of states of an SSP
$s_0 \in S$	An initial state of an SSP
$S_g \subseteq S$	A non-empty set of goal states of an SSP
A	A set of actions of an SSP
$A(s)$	All actions applicable in state $s \in S$
$P : S \times S \times A \mapsto [0, 1]$	A probabilistic transition function
$C : S \times A \mapsto \mathbb{R}^+$	A cost function
$\pi : S \times A \mapsto [0, 1]$	One policy function as a solution to an SSP
$S^\pi \subseteq S$	The domain of $\pi(s_0)$
$V^\pi : S \mapsto \mathbb{R}_{\geq 0}$	The state-value function
$Q^\pi : S \times A \mapsto \mathbb{R}_{\geq 0}$	The action-state-value function
$\pi^* = \arg\min_\pi V^\pi(s_0)$	Optimal solution to an SSP
V^*	Optimal state-value function
Q^*	Optimal action-state-value function
Π	The set of all (deterministic) policies for an SSP
Π^{MP}	The set of all Max-Prob policies
$T_{\pi,s}$	A trace from s following $\pi \in \Pi$
$\mathcal{T}_{\pi,s}$	The set of all traces of π from s
$\mathcal{T}^G_{\pi,s}$	The set of all traces that reach the goal
$\mathcal{T}^{DE}_{\pi,s}$	The complement of $\mathcal{T}^G_{\pi,s}$, i.e., the set of all traces that reach dead ends
$\mathbb{S}^c = \langle S, s_0, S_g, A, P_c, C \rangle$	A classical planning problem
$\mathbb{S}^{cL} = \langle \mathbb{D}, O, s_0, g \rangle$	A lifted classical planning problem
$\mathbb{D} = \langle \mathcal{P}, \mathcal{A} \rangle$	The domain of a lifted SSP
$\mathbb{S}^{cL+}$	Relaxed version of the planning problem $\mathbb{S}^{cL}$
$p^{\max} = \max_{\pi \in \Pi} P(\mathcal{T}^G_{\pi,s_0})$	Maximum goal probability in Max-Prob Criterion
$V^p : S \mapsto [0, 1]$	The probabilistic value function under Max-Prob with modified Bellman
$Q^p : S \times A \mapsto [0, 1]$	The probabilistic action-state-value function under Max-Prob with modified Bellman
$\psi : T \mapsto T$	A truncation function
$\mathbb{LP}(\mathbb{S})$	The LP formulation of an SSP
V_{LP}	Objective function of an LP
y	Dual variables of a LP
y^*	Optimal dual variables of an LP

Continued on next page

Notation	Description
$G = \langle V, E \rangle$	A graph G with the set of vertices V and edges E
$\omega_V : V \mapsto \mathbb{X}$	A node weight mapping function of graph
$\omega_E : E \mapsto \mathbb{X}$	An edge weight mapping function of graph
$\kappa : E \mapsto \Sigma$	An edge color mapping function of graph
$G = \langle V, E, \omega_V, \omega_E \rangle$	A weighted graph with the set of vertices V and edges E, the node weight mapping ω_V, and the edge weight mapping ω_E
$G = \langle V, E, \omega_V, \omega_E, \kappa, \Sigma \rangle$	A weighted multi-edge graph G with the set of vertices V, list of edges E, node weight mapping ω_V, edge weight mapping ω_E, edge color mapping κ, and a set of colors Σ
$G = \langle V, A \rangle$	A directed graph G with the set of vertices V and arcs A
$\mathcal{N}(v)$	Neighborhood of v in G
$\mathcal{N}_l(v)$	Colored neighborhood of v in G connected by edge with color l
$h : S \mapsto \mathbb{R}$	A heuristic function
$h^{\max}$	Max-Cost heuristic
h^{FF}	Fast-Forward heuristic
$\mathcal{F}$	A regression task function
$\mathcal{GP}(\mu(x), k(x, x'))$	A Gaussian Process
$\mu(x)$	A mean function of a GP
$k(x, x')$	A covariance function of a GP

Chapter 3

Related Work

The aim of this chapter is to review the works closely related to our research, providing a concise discussion on the challenges they face and their limitations. We divide the related works into two sections: Section 3.1 addresses the area relates to solving the Max-Prob criterion; while Section 3.2 focuses on the area of learning for planning. Section 3.3 concludes the chapter with an discussion on how our motivation stems from the challenges in these two areas, as well as an emphasis on the research objectives that align with those motivations. By further elucidating our motivation and research objectives, we expect to offer the audience a conceptual understanding of the significance and novelty of our contribution, which are designed to address these challenges. Consequently, this facilitates a smooth and logical transition to the next chapter.

3.1 Solving Max-Prob

The Max-Prob criterion for SSPs, as defined in Definition 9, involves finding a policy that maximizes the probability of reaching a goal in an SSP. It also serves as the necessary preliminary stage for the more robust MCMP criterion (Definition 11). Several approaches have been proposed to solve Max-Prob problems in the past decade.

3.1.1 Algorithms for Solving Max-Prob SSPs

Before discussing the algorithms for solving Max-Prob SSPs, we first introduce some algorithms that are commonly used to solve standard SSPs. Numerous existing algorithms are available for solving SSPs without dead ends including: Value Iteration (VI) [Howard, 1960], which computes the solution for Bellman Equations (Equation 2.5, 2.6) in an iterative manner; linear programming [Puterman, 2014], which utilises constraints to represent transition probabilities; and the well-established heuristic search algorithms such as LAO* [Hansen and Zilberstein, 2001] and (L)RTDP [Bonet and Geffner, 2003].

Collectively, these methods are referred to as primal-space algorithms, as they directly compute the optimal solution for SSPs. Aside from primal-space algorithms, policy iteration [Howard, 1960] solves dead-end-free SSPs by alternating between policy evaluation and policy improvement; while dual linear programming [d'Epenoux, 1963] solves SSPs by solve their dual LP representations.

As outlined in Section 9, SSPs under the Max-Prob criterion can be reformulated as standard SSPs by shifting the objective from minimising expected cost to maximising the probability of reaching the goal. The modified Bellman equations (2.5) provide one approach for solving such Max-Prob SSPs with algorithms designed for standard SSPs (e.g. VI). However, when the SSPs contain cycles, directly solving the modified Bellman may return non-optimal fixed-point solutions. In order to solve Max-Prob SSPs with cycles, a few methods have been developed by extending the ideas from algorithms that solves standard SSPs. For instance, VI on the modified Bellman equations can still be applied by initiating the algorithm with a value of 1 for all goal states and 0 elsewhere. Another approach for solving Max-Prob SSPs with circles involves heuristic search algorithms like FRET [Kolobov et al., 2011] and FRET-π [Steinmetz et al., 2016], both of which alternate between: finding a fixed-point solution for the maximum goal probability using an optimal heuristic search algorithm for SSPs; and post-processing the solution to remove cycles. Since both FRET and FRET-π are heuristic search algorithms, they require a heuristic function that estimates the maximum probability of reaching the goal, $p^{\max}$.

The only other approach for solving Max-Prob SSPs is i-dual, which achieves this by solving the first stage of MCMP as described in Theorem 1. This method is our primary focus, as it has been shown to be more efficient than the above approaches, even when using an average-level heuristic function for $p^{\max}$ adapted from classical planning problems [Trevizan et al., 2017b, 2016]. Therefore, i-dual has potential for further performance gains, especially with improvements in heuristic functions for $p^{\max}$.

3.1.2 Heuristic Functions for Max-Prob

A major advantage of heuristic search algorithms, especially in the Max-Prob context, is their capacity to leverage well-designed heuristic functions to improve search efficiency. However, only limited efforts have been made to design effective heuristic functions for $p^{\max}$, and most existing attempts lack probabilistic encoding [Klößner et al., 2021]. For instance, E-Martín et al. [2014]; Keyder and Geffner [2008] explore determinisation by first converting SSPs into classical planning problems and then generating heuristic functions based on the maximum-likelihood sequential plan. As discussed in Section 2.1.2, this process often disregards crucial probabilistic information, oversimplifying the original SSP and significantly altering the problem's nature. Consequently, heuristic functions derived from determinisation are often less informative. Trevizan et al. [2017a] introduce two admissible heuristics for the Min-Cost stage that consider probabilities: the projection occupation measure heuristic, h^{pom}, and the regrouped operator-counting heuristic, h^{roc}. Both are domain-independent and admissible, demonstrating strong per-

formance across multiple domains for Min-Cost tasks. However, they are not designed for Max-Prob SSPs and therefore underperform when applied to Max-Prob SSPs [Klößner et al., 2021].

Most other heuristic functions for Max-Prob rely on combining determinisation with classical planning heuristics. These functions apply classical planning heuristic functions, which estimate the cost to reach the goal on the determinised SSPs to obtain the heuristic value for $p^{\max}$. The resulting heuristic function for $p^{\max}$ returns either 0 or 1, where 0 is returned only if the determinised SSP is unsolvable (with an infinite cost estimate to reach the goal) and 1 otherwise. This 0-or-1 heuristic is admissible but offers little information, particularly when dealing with intermediate states with partial goal probabilities [**?**Trevizan et al., 2017b; Klauck et al., 2020]. The sole exception is Max-Prob Pattern Databases (MPDBs) [Klößner et al., 2021], which effectively guide search in Max-Prob tasks by abstracting the problem into smaller subproblems that preserve probabilistic transitions. MPDBs provide admissible heuristics that upper-bound the real $p^{\max}$, encoding probabilistic information directly through unique probabilities for abstract transitions. Nevertheless, MPDBs still fail to outperform 0-or-1 heuristics generated from $h^{\max}$ (referred to hereafter as "$0-1$ $h^{\max}$") in many domains, especially cyclic, resource-unconstrained environments, which are the focus of our study. In conclusion, despite the significant loss of information, $0-1$ $h^{\max}$ remains the best-performing heuristic for predicting $p^{\max}$ when applied to heuristic search algorithms on our testbeds, highlighting the need for more robust heuristic functions for $p^{\max}$.

3.2 Learning for Planning

The integration of ML into planning has evolved rapidly, driven by advances in DL and its capacity to automatically learn complex relationships and patterns from data. Two primary research directions have emerged in this area: learning generalised policies and learning heuristics, with the latter being most relevant to our work.

3.2.1 Learning Generalised Policies

We begin by examining prior work on learning for planning through generalised policies. A generalised policy is one that can be applied across all possible problems within a given domain, rather than being limited to a specific instance. Similar to a heuristic function, a generalised policy guides the agent's actions, but instead of estimating costs and relying on an explicit heuristic search algorithm to choose the next action, a model with a learnt generalised policy selects the next action directly.

Action Schema Networks (ASNets) [Toyer et al., 2020] represent a significant development in learning generalised policies for planning. ASNets leverage the relational structure of planning problems encoded in (P)PDDL to learn generalised policies without requiring handcrafted input features. They employ a specialised neural network architecture comprising alternating action and proposition layers, sparsely connected based

on the action schemas defined in (P)PDDL problems. This weight-sharing mechanism provides theoretical guarantees for generalisation across problems of varying sizes within a domain. However, ASNets are limited by their fixed receptive field, which restricts their capacity for handling long chains of reasoning.

Other approaches to generalised policy learning use hand-crafted domain-specific translators to convert states into representations before feeding them into GCNs [Groshev et al., 2018]. ToRPIDo [Bajpai and Garg, 2018] and TraPSNet [Garg et al., 2019] apply standard GCN and Graph Attention Networks [Velickovic et al., 2017] to RDDL-defined planning problems [Sanner et al., 2010]. While effective in domain-dependent settings, these models impose limitations such as unary actions (actions with a single argument) and binary non-fluents (static propositions with two arguments). In contrast, SymNet [Garg et al., 2020] and SymNet2 [Sharma et al., 2022] extend generalised policy learning for RDDL to handle more expressive domains without these restrictions.

As discussed in [Shen et al., 2020], however, learning heuristics is often a more robust approach than learning actions from generalised policies. While combining a model that learns generalised policies with a search algorithm is possible, as [Shen et al., 2019] demonstrated by integrating Monte-Carlo Tree Search with ASNets. Using a model that directly learns heuristics within a heuristic search algorithm provides formal guarantees. For example, A* search is complete, meaning it will always find a solution if one exists, regardless of the heuristic's accuracy. This makes learned heuristics ideal for critical applications, as the heuristic search algorithm can compensate for any inaccuracies or misleading information in the learned heuristic.

3.2.2 Learning Heuristics

Learning heuristics for planning has become essential because it enables planners to manage large-scale, complex, and dynamic environments more effectively. This approach reduces reliance on domain expertise, enhances scalability, and integrates well with modern ML techniques, making it possible to address problems that would otherwise be infeasible with manually designed heuristics. This has significant implications for real-world applications where both performance and adaptability are essential [Bonet, 2001; Geffner, 2018; Salhi and Thompson, 2022].

Over the years, various methods have been proposed, focusing on both domain-dependent and domain-independent heuristics. Early work such as Yoon et al. [2008] introduced learning domain-dependent heuristics by predicting the difference between the optimal heuristic h^* and easily computed relaxed heuristics like h^{FF}. Similarly, Samadi et al. [2008] employed neural networks to learn either admissible or optimal heuristics for each domain, using different loss functions and representing states as vectors of pre-computed heuristics. However, these methods are limited by their heavy reliance on existing heuristics. Arfaee et al. [2011] proposed a bootstrapping method that iteratively improves a weak heuristic by generating training samples from search results. Although this approach incrementally strengthens the learned heuristic, it remains domain-dependent

and is computationally expensive due to its iterative nature, without guaranteeing optimality in a reasonable timeframe.

STRIPS-Hypergraph Networks (STRIPS-HGN) [Shen et al., 2020] marked a major shift by introducing a deep learning approach to learning domain-independent heuristics from scratch using a hypergraph representation of planning instances. Unlike previous approaches, STRIPS-HGN does not rely on pre-computed heuristics. However, it has limitations: the model omits delete lists, lacks permutation invariance in its update functions, imposes constraints on action preconditions and effects, and requires the construction of the entire grounded hypergraph. These limitations result in performance bottlenecks in complex domains. Nevertheless, STRIPS-HGN demonstrates better performance than $h^{\max}$ on certain domains.

The Graphs Optimised fOr Search Evaluation (GOOSE) [Chen et al., 2023a, 2024] is the state-of-the-art architecture for learning heuristics that addresses STRIPS-HGN's limitations. GOOSE introduces two graph-based representations for planning tasks: Lifted Learning Graphs (LLG) [Chen et al., 2023a] and Instance Learning Graphs (ILG) [Chen et al., 2024], enabling heuristic learning as a feature of the planning task for both SML and GNN models. In the GOOSE framework, graphs can be input directly to GNN models or transformed into a feature space for SML models using a variant of the Weisfeiler-Lehman algorithm [Weisfeiler and Leman, 1968]. Both graphs are represented in a lifted form. LLG enables the learning of domain-independent heuristics and outperforming all previous approaches including STRIPS-HGN. ILG, on the other hand, refocuses on learning robust domain-dependent heuristics. The domain-dependent heuristics learned from ILG achieve state-of-the-art performance when guiding heuristic search, surpassing both h^{FF} and heuristics learned from LLG.

In summary, GOOSE has introduced a new phase in learning for planning: separating model learning from problem encoding. This allows various existing models, such as GCN [Kipf and Welling, 2016], RGCN [Schlichtkrull et al., 2018], SVM [Cortes, 1995], and GPR [Williams and Rasmussen, 2006], to learn the features from any problem set with a certain representation without modifying the model's structure. Chen et al. [2023a] also provide extensive theoretical and empirical analysis that demonstrates the framework's efficiency and effectiveness over previous approaches, marking GOOSE a significant advancement in heuristic learning for planning in deterministic environments. However, GOOSE enables only the learning of heuristics for classical planning problems, as its graph encoding does not support SSPs. To the best of our knowledge, no research has yet been undertaken to learn any heuristics for SSPs, particularly for learning heuristics for $p^{\max}$.

3.3 Discussion

As discussed in Section 2.2, when unavoidable dead ends are present, it becomes impossible to always reach the goal. In such cases, an SSP transforms into a multi-objective optimisation problem, with conflicting goals of either maximising the probability of reaching the goal or minimising the cost of doing so. Unavoidable dead ends are common in real-world SSPs; however, most SSP solvers rely on applying a finite-penalty criterion to transform the SSP into a dead-end-free version before solving it, which is often impractical in many domains. MCMP has been shown to be one of the most advanced criteria for both efficiency and robustness when handling SSPs with unavoidable dead ends. Extending the Max-Prob criterion by estimating cost from all possible traces, MCMP achieves a closer alignment with actual costs compared to previous criteria, such as S^3P and iSSPUDE. Furthermore, SSPs under MCMP can be formulated as two LPs, enabling i-dual—the only heuristic search algorithm to operate on SSPs under MCMP—to solve them far more efficiently than other criteria.

Since i-dual is a heuristic search algorithm, its performance is directly influenced by the quality of heuristic functions that guide its search. In Section 3.1, we discussed the development of robust heuristics, such as h^{roc} and h^{pom}, which efficiently guide i-dual in the second (Min-Cost) stage of MCMP. However, the Max-Prob stage lacks an effective heuristic capable of guiding the search by accurately estimating $p^{\max}$. Moreover, prior efforts to create effective heuristics for $p^{\max}$ have not incorporated machine learning techniques, and most of them lack the encoding of probabilistic information. This leads to our **first research objective**: to develop an effective heuristic for $p^{\max}$ using machine learning techniques to allow consideration of probabilistic information, which can be applied in the Max-Prob stage of i-dual to enhance its performance relative to current state-of-the-art heuristics for $p^{\max}$.

In Section 3.2, we showed that learning for planning has advanced rapidly over the past decade, with the state-of-the-art approach being GOOSE, which learns heuristic functions in deterministic domains. GOOSE's robustness aligns naturally with our first research objective, while its limitation to classical planning problems motivates our **second research objective**: extending GOOSE to handle SSPs, enabling various models to learn heuristic functions for $p^{\max}$. Since GOOSE separates model learning from problem representation, this can be achieved by designing a novel graph representation beyond ILG and LLG that captures SSPs without losing valuable probabilistic information; and by upgrading the framework to support model training on these new graph representations.

Our research is centred around these two objectives, leading to the contributions discussed in Section 1.2. In the next chapter, we detail our methodology.

Chapter 4

Learning for Planning Under Uncertainty

The aim of this chapter is to present our approach that address learning for planning under uncertainty, specifically detailing our implementation to achieve the two primary research objectives: developing an effective heuristic for $p^{\max}$ using machine learning techniques that incorporate probabilistic information within SSPs; and extending GOOSE to accommodate SSPs by creating novel graph representations, thus enabling the learning of SSP features including $p^{\max}$.

We begin in Section 4.1 by formalising lifted SSPs, establishing the foundation and inspiration for our graph-based methods. Here, we also provide examples of a lifted SSP and its determinised version, which serve as illustrative references for the graph structures introduced in the remainder of the chapter. Section 4.2 explores the structural design of our novel graph representations, PLGS and PLGL, along with a comprehensive analysis of how they address ILG's limitations in learning-for-planning with SSPs, particularly in terms of expressiveness and generalisability. In Section 4.3, we turn to the development of learning frameworks, detailing how we accomplish the second objective by extending GOOSE to enable both GNN and SML models to be trained to learn features such as $p^{\max}$ of SSPs throughout various graph representations. We also demonstrate that our graph representations are strictly more expressive than ILG therefore enhancing their effectiveness within these learning frameworks. Finally, Section 4.4 concludes the chapter by explaining how we achieve the first research objective by transforming the predictions from models learning $p^{\max}$ into a suitable Max-Prob heuristic.

4.1 Lifted Planning Problem Representations

Recall from Section 2.1.3 that we defined $\mathbb{S}^{cL}$ (Definition 7) as the lifted representation of a classical planning problem $\mathbb{S}^{c}$ (Definition 6). PDDL is a language that describes $\mathbb{S}^{cL}$ in a

format compatible with planning solvers. Younes and Littman [2004] extended PDDL to PPDDL, which describes a lifted representation for SSPs, thus enabling solvers to accept SSPs as input. In this section, we present a formal lifted representation for an SSP, $\mathbb{S}^L$, which defines action schemas in Unary Nondeterminism Normal Form [Rintanen, 2003] and can be expressed in PPDDL.

Definition 17 (Lifted SSPs)**.** A lifted SSP is a tuple $\mathbb{S}^L = \langle \mathbb{D}, O, s_0, g \rangle$ where:

- O is the set of objects in the problem, objects are allowed to have different types.
- $\mathbb{D} = \langle \mathcal{P}, \mathcal{A} \rangle$ is the domain for $\mathbb{S}^L$ where:
  - $\mathcal{P}$ is a set of first-order predicates representing the environment.
  - A predicate $P \in \mathcal{P}$ has a tuple of parameters $P(x_1, \ldots, x_{n_P})$ for $n_P \in \mathbb{N}$, which we call an n_p-ary predicate.
  - A predicate with all parameters assigned to objects $o \in O$ is called a (grounded) proposition. (A predicate can have no parameters, in which case it is grounded by default).
  - $\mathcal{A}$ is a set of action schemas.
  - An action schema $a \in \mathcal{A}$ is a tuple $\langle \Delta(a), \text{pre}(a), \textbf{outcomes}(a), \text{cost}(a) \rangle$ where $\text{pre}(a) = \langle \text{pre}_+(a), \text{pre}_-(a) \rangle$ are the positive/negative preconditions; $\textbf{outcomes}(a) = [\text{outcome}(a_1), \text{outcome}(a_2), \ldots, \text{outcome}(a_{n_J})]$ is a list of $n_J \in \mathbb{N}$ outcomes. Each $\text{outcome}(a_j) = \langle prob_{a_j}, \text{add}(a_j), \text{del}(a_j) \rangle$ is represented by a probability $prob_{a_j}$ for its occurrence when action a is applied, along with corresponding add and delete effects $\text{add}(a_j), \text{del}(a_j)$. $\Delta(a)$ is a set of parameter variables; $\text{pre}_+(a), \text{pre}_-(a), \text{add}(a_j), \text{del}(a_j)$ are sets of predicates from $\mathcal{P}$; $\text{cost}(a)$ is a cost function, all instantiated with parameter variables or objects in $\Delta(a) \cup \mathcal{O}$. Like predicates, an action schema with $n = |\Delta(a)|$ parameter variables is an n-ary action schema.
- s_0 is a set of propositions representing the initial state.
- g is a set of propositions representing the partial goal state.

■

As discussed in Section 2.1.3, lifted SSPs enable domain-independent planning by providing a more efficient and compact representation. In this thesis, both $\mathbb{S}^L$ (Definition 17) and $\mathbb{S}$ (Definition 1) represent the same SSP but in different formats. We demonstrate this by showing how a lifted SSP $\mathbb{S}^L = \langle \langle \mathcal{P}, \mathcal{A} \rangle, O, s_0, g \rangle$ can be transformed into an SSP $\mathbb{S} = \langle S', s'_0, S'_g, A', P', C' \rangle$ as follows:

- **States** S': The predicate set $\mathcal{P}$ defines the set of all possible states S' with objects from O: a state $s' \in S'$ is a set formed by valid combinations of propositions instantiated by pairing predicates $P \in \mathcal{P}$ with objects $o \in O$.

- **Actions** A': The set of objects O can also instantiate action schemas $\mathcal{A}$. All possible instantiated actions form the set A'.
- **Initial State** s'_0: The initial state s_0 directly corresponds to the initial state s'_0 in the classical planning model.
- **Goal States** S'_g: The goal states S'_g comprise all superset states of the partial goal state set g.
- **Transition Function** P' **and Cost Function** C': In $\mathbb{S}$, the probabilistic transition function P' maps a state and an action to their resulting states with a probability distribution and a single cost described by C'. If we denote $a' \in A'$ as the action instantiated from action schema $a \in \mathcal{A}$ with parameters $O' \subseteq (\Delta(a) \cup \mathcal{O})$, then $P'(s'|s, a')$ and $C'(s, a')$ can be defined as follows:

  For each outcome$(a_j)(O')$, $P'(s'|s, a') = prob_{a_j}$ and $C'(s, a') = \text{cost}(a)(O')$ for all combinations of $s, s' \in S' \times S'$ where s is a superset of $\text{pre}_+(a)(O')$ containing no elements from $\text{pre}_-(a)(O')$, with the corresponding $s' = s \cup \text{add}(a_j)(O') \setminus \text{del}(a_j)(O')$. This specifies that action a' can transition s to s' with probability $prob_{a_j}$ and cost $C'(s, a)$.

This process mirrors that discussed in Section 2.1.3, with the key difference being that we consider all possible outcomes for action schemas rather than a single deterministic outcome. Throughout this thesis, all our graph representations will be based on lifted problem representations. To illustrate and differentiate the various graphs introduced in the following section, we represent the Box Delivery Problem in both lifted SSP and lifted classical planning problem formats (using all-outcome determinisation) as below:

Example 4. $\mathbb{S}^L_{box} = \langle\langle \mathcal{P}, \mathcal{A}\rangle, O, s_0, g\rangle$ *is the lifted SSP representation (Definition 17) for the Box Delivery Problem, where:*

▷ $O = \{b - Box, c - City\}$ contains two objects where b is a box of type "Box" (we use B to represent a variable of type "Box") and c is a city of type "City" (we use C to represent a variable of type "City").

▷ $\mathcal{P} = \{no - destroy(B), arrived(B, C)\}$ where $no - destroy(B)$ indicates whether a box B is intact, and $arrived(B, C)$ indicates whether a box B has reached city C.

▷ $\mathcal{A} = \langle fly, drive\rangle$ where both $fly(B, C)$ and $drive(B, C)$ are action schemas.

▷ fly is an action schema with parameters $\Delta(fly) = \{B, C\}$; positive precondition $\text{pre}_+(fly) = \{no - destroy(B)\}$; negative precondition $\text{pre}_-(fly) = \{arrived(B, C)\}$; and $\text{cost}(fly) = \$1000$. This represents the action of delivering B to C by plane at a cost of \$1000, only if B is intact and has not reached C. fly has two possible effects:

  – $\text{outcome}_1(fly) = \langle 0.05, \{\}, \{no - destroy(B)\}\rangle$, indicating a 5% chance that B is destroyed in transit.

  - $\text{outcome}_2(fly) = \langle 0.95, \{arrived(B, C)\}, \{\}\rangle$, indicating a 95% chance that B is successfully delivered to C.

▷ *drive* is another action schema with the same parameters and preconditions as fly, but with a lower cost of \$100. It represents the action of delivering B to C by car at a cost of \$100, provided B is intact and has not reached C. *drive* also has two possible effects:

  - $\text{outcome}_1(drive) = \langle 0.1, \{\}, \{no-destroy(B)\}\rangle$, indicating a 10% chance that B is destroyed in transit.
  - $\text{outcome}_2(drive) = \langle 0.9, \{arrived(B, C)\}, \{\}\rangle$, indicating a 90% chance that B is successfully delivered to C.

▷ $s_0 = \{no - destroy(b)\}$, $g = \{no - destroy(b), arrived(b, c)\}$ indicates the objective is to deliver a box b to city c without it being destroyed.

Example 5. $\mathbb{S}^{cL}_{box} = \langle\langle \mathcal{P}, \mathcal{A}\rangle, O, s_0, g\rangle$ *is the lifted classical planning problem representation (Definition 7) for the Box Delivery Problem after all-outcome determinisation (see Sections 2.1.2 and 4.3.3), where:*

▷ $\mathcal{P}, O, s_0, g$ are as defined in $\mathbb{S}^{L}_{\textbf{box}}$.

▷ $\mathcal{A} = \langle fly - 1, fly - 2, drive - 1, drive - 2\rangle$ are the determinised action schemas.

▷ $fly-1$ and $fly-2$ have the same parameters, preconditions, and cost as fly in $\mathbb{S}^{L}_{\textbf{box}}$, but each represents a single deterministic effect, i.e., $\text{del}(fly-1) = \{no-destroy(B)\}$ and $\text{add}(fly - 2) = \{arrived(B, C)\}$.

▷ $drive - 1$ and $drive - 2$ have the same parameters, preconditions, and cost as *drive* in $\mathbb{S}^{L}_{\textbf{box}}$, but each represents a single deterministic effect, i.e., $\text{del}(drive - 1) = \{no - destroy(B)\}$ and $\text{add}(drive - 2) = \{arrived(B, C)\}$.

4.2 Representations as Learning Graphs

As discussed in Chapter 3, the state-of-the-art learning-for-planning framework GOOSE translates lifted classical planning problems into graphs, enabling models to learn features from these graphical representations. In Section 4.2.1, we begin by illustrating the *Instance Learning Graph* from GOOSE, discussing its structure and potential limitations when being applied to SSPs. We then introduce two novel graph representations for SSPs in Sections 4.2.2 and 4.2.3 with a discussion on how they address limitations of ILG.

We use the lifted representations for the box delivery example ($\mathbb{S}^{L}_{\textbf{box}}$ and $\mathbb{S}^{cL}_{\textbf{box}}$) throughout this section to illustrate the graphs and provide insight into their structural design. All graph representations are given as (undirected) weighted multi-edge graphs (Definition 16), with omitted edge weights, denoted as $G = \langle V, E, \omega, \kappa, \Sigma\rangle$. Here, $\omega : V \mapsto \mathbb{R}^d$ represents the weight mapping function, which assigns each node a corresponding feature

vector in $\mathbb{R}^d$. This format directly aligns with the expected input structure for GNNs, as discussed later in Section 4.3.1. We will further show how G can be easily transformed into a colour graph, enabling SML models to process it as input in Section 4.3.2.

4.2.1 Instance Learning Graph

The Instance Learning Graph (ILG) is a graph representation that enables models to learn features from classical planning problems with state-of-the-art accuracy through the GOOSE framework [Chen et al., 2024]. It can be formally defined as follows:

Definition 18 (Instance Learning Graph)**.** An Instance Learning Graph (ILG) representation of a lifted classical planning problem $\mathbb{S}^{cL} = \langle\langle \mathcal{P}, \mathcal{A}\rangle, O, s_0, g\rangle$ is the graph $G^{ILG} = \langle V, E, \omega, \kappa, \Sigma\rangle$ where

- $V = O \cup s_0 \cup g$ is an ordered set of nodes.
- $E = \bigcup_{p=P(o_1,\dots,o_{n_p})\in s_0\cup g}\{(p, o_1), \dots, (p, o_{n_p})\}$ is the set of all edges.
- $\omega : V \to \mathbb{R}^d$ where $d = 1 + 3 \times |\mathcal{P}|$. For feature vector $\omega(v) \in \mathbb{R}^d$, $v \in V$ we use $\omega(v)[i]$ to denote the i-th element of $\omega(v)$. We have the following:
  - $\omega(v)[1] = 1$ if v is an object node, i.e., $v \in O$;
  - $\omega(v)[3 \times j - 1] = 1$ if v is the j-th element of the ordered set $s_0 \cup g$ and v is an "achieved goal proposition" (ag), i.e., $v \in s_0 \cap g$.
  - $\omega(v)[3 \times j] = 1$ if v is the j-th element of the ordered set $s_0 \cup g$ and v is an "achieved non-goal proposition" (ap), i.e., $v \in s_0 \setminus g$.
  - $\omega(v)[3 \times j + 1] = 1$ if v is the j-th element of the ordered set $s_0 \cup g$ and v is an "unachieved goal proposition" (ug), i.e., $v \in g \setminus s_0$.
- $\Sigma = \{1, 2, \dots, \arg\max_{P\in\mathcal{P}} n_P\}$: the number of colours is equivalent to the maximum number of parameters within any single predicate $P \in \mathcal{P}$.
- $\kappa : E \mapsto \Sigma$ is the colour function where $\kappa(p, o_i) = i$.

■

Figure 4.1 shows the ILG representation of $\mathbb{S}^{cL}_{\mathbf{box}}$ as an illustrative example. Intuitively, ILG only selects problem-specific information s_0, g, O to create the graph nodes, so every node is either an object node or a proposition node. Since ILG was originally designed with node features represented by different colours, we use one-hot encoding to convert colours into feature vectors as in Chen et al. [2024]. All object nodes share the same feature, while proposition nodes are categorised as one of *ag, ap, ug* combined with their predicate type; this creates a total of $1+3\times|\mathcal{P}|$ distinct features under one-hot encoding. The edges link each proposition to its parameters, and the colour of each edge represents the index of the object as parameter within the proposition connects to it.

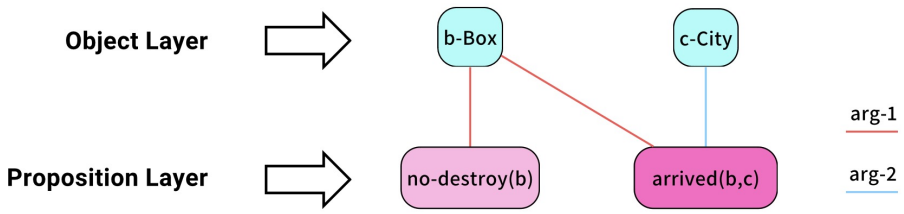

Figure 4.1: ILG representation of $\mathbb{S}^{cL}_{\mathbf{box}}$. Different colours represent different node features. The initial state *no-destroy*(b) represents an achieved goal for predicate *no-destroy*(B), whereas the goal state *arrived*(b, c) corresponds to an unachieved goal for predicate *arrived*(B, C). Edges connect proposition nodes and their parameters (object nodes) with colours indicating the parameter indices, labelled as "arg-i".

The ILG's use of problem-specific information alone is due to its focus on learning domain-dependent features from lifted classical planning problems [Chen et al., 2024]. This compact design makes ILG both efficient (quick to train and predict) and effective (state-of-the-art performance in learning domain-specific heuristic values). Moreover, ILG's encoding can be easily adapted to lifted SSPs since $\mathbb{S}^{cL}$ and $\mathbb{S}^{L}$ share the same s_0, g, O. However, applying ILG to lifted SSPs reveals some limitations:

1. ILG assigns the same feature to all objects, regardless of their types. This can limit expressiveness in domains where objects vary by type.
2. The current one-hot encoding for ILG's node feature vector in $\mathbb{R}^d$ where $d = 1 + 3 \times |\mathcal{P}|$ is a direct translation from colour to feature vector, which is not very efficient as GNN inputs due to the excessive length of the feature vector.
3. ILG is designed specifically for classical planning problems and thus does not encode probabilistic information, as probabilities reside within $\mathcal{A}$, which is domain-specific. This design stems from the assumption that classical planning problems are relatively straightforward and that relationships between predicates and action schemas are not overly complex, enabling model learning meaningful features from solely problem-specific information. However, this assumption does not hold for SSPs, which are often more challenging to solve, with problem domains encoding crucial probabilistic relationships, particularly in SSP domains that may contain cycles. The absence of domain-specific encoding in ILG can significantly reduce its generalisability in learning-for-planning tasks with SSPs, especially when training data is sparse. In such cases, the objects and propositions present across initial and goal states in the training data are likely to represent only a small subset of all possible objects and propositions in the domain, thereby limiting the model's ability to generalise effectively to new, unseen problems.

To address these limitations, we design two graphs that represent lifted SSPs at different levels of detail. In the following sections, we introduce our novel graph representations and discuss how they overcome these limitations.

4.2.2 Probabilistic Learning Graph Small

To address the limitations of ILG when applied to lifted SSPs, we extend ILG to create our first novel graph, the *Probabilistic Learning Graph Small* (PLGS), which, to the best of our knowledge, is the first graph structure specifically designed to represent lifted SSPs in a learning-for-planning task.

Definition 19 (Probabilistic Learning Graph Small)**.** A Probabilistic Learning Graph Small (PLGS) representation of a lifted SSP $\mathbb{S}^L = \langle \langle \mathcal{P}, \mathcal{A} \rangle, O, s_0, g \rangle$ is the graph $G^{PLGS} = \langle V, E, \omega, \kappa, \Sigma \rangle$ where

- $V = O \cup s_0 \cup g \cup \mathcal{P} \cup N(\mathcal{A})$ is an ordered set of nodes where:
  - $N(\mathcal{A}) = \mathcal{A} \cup \{a_i \mid a \in \mathcal{A}, \text{outcome}(a_i) \in \text{outcomes}(a)\}$ is the set containing both action schema nodes and their corresponding outcome nodes.
- $E = E_{param} \cup E_{out} \cup \bigcup_{e \in \{add,del\}} E_e \cup \bigcup_{f \in \{pre_+,pre_-\}} E_f \cup E_{ground}$ is a set of edges where:
  - $E_{param} = \bigcup_{p=P(o_1,\ldots,o_{n_p}) \in s_0 \cup g} \{(p, o_1), \ldots, (p, o_{n_p})\}$ is the set of parameter edges that link object nodes with proposition nodes, as in ILG.
  - $E_{out} = \{(a, a_i) \mid a \in \mathcal{A}, \text{outcome}(a_i) \in \text{outcomes}(a)\}$ is the set of outcome edges that link each action schema node with its outcome nodes.
  - $E_e = \bigcup_{P \in e(a)} \{(a_i, P) \mid a \in \mathcal{A}, \text{outcome}(a_i) \in \text{outcomes}(a)\}$ is the set of effect edges that link outcome nodes with their corresponding add/delete effect predicates.
  - $E_f = \bigcup_{P \in f(a)} \{(a, P) \mid a \in \mathcal{A}\}$ is the set of precondition edges that link action schema nodes with their corresponding positive/negative precondition predicates.
  - $E_{ground} = \bigcup_{p=P(o_1,\ldots,o_{n_p}) \in s_0 \cup g} \{(p, P)\}$ is the set of grounding edges that link each grounded proposition to its predicate.
- $\omega : V \to \mathbb{R}^d$ where $d = 1 + \text{types}(O) + |\mathcal{P}| + |\mathcal{A}| + 1$ and types(O) returns the total number of types within the objects O. For feature vector $\omega(v) \in \mathbb{R}^d$, $v \in V$, we use $\omega(v)[i]$ to denote the i-th element of $\omega(v)$. The following holds:
  - $\omega(v)[1] = 1$ if v is a ground proposition node, i.e., $v \in s_0 \cup g$;
  - $\omega(v)[1 + j] = 1$ if v is an object node $v \in O$ and the type of v is the j-th element within the ordered set types(O).

  - $\omega(v)[1+\text{types}(O)+j] = 1$ if v is the j-th element of the ordered set $\mathcal{P}$, indicating it is a predicate node.
  - $\omega(v)[1 + \text{types}(O) + |\mathcal{P}| + j] = 1$ if v is the j-th element of the ordered set $\mathcal{A}$, indicating it is an action schema node.
  - $\omega(v)[-1] = prob_{a_j}$ if $v = a_j$ is any outcome node with probability $prob_{a_j}$.
- $\Sigma = \{1, 2, \ldots, (I + 8)\}$ where $I = \arg\max_{P \in \mathcal{P}} n_P$ denotes the maximum number of parameters within any single predicate $P \in \mathcal{P}$. The total number of colours is therefore $I + 8$.
- $\kappa : E \mapsto \Sigma$ is the colour function where:
  - $\kappa(p, o_i) = i$ for $(p, o_i) \in E_{param}$.
  - $\kappa(e) = I + 1$ for $e \in E_{out}$.
  - $\kappa(e) = I + 2$ for $e \in E_{eff_+}$.
  - $\kappa(e) = I + 3$ for $e \in E_{eff_-}$.
  - $\kappa(e) = I + 4$ for $e \in E_{pre_+}$.
  - $\kappa(e) = I + 5$ for $e \in E_{pre_-}$.
  - $\kappa(p, P) = I + 6$ for $p \in s_0 \cap g$, indicating p is ag.
  - $\kappa(p, P) = I + 7$ for $p \in s_0 \setminus g$, indicating p is ap.
  - $\kappa(p, P) = I + 8$ for $p \in g \setminus s_0$, indicating p is ug.

■

Figure 4.2 provides an illustrative example of the PLGS representation for $\mathbb{S}^L_{\textbf{box}}$. Intuitively, PLGS can be divided into a *problem subgraph* and a *domain subgraph*.

The problem subgraph can be viewed as an extension of ILG, focusing on problem-specific information s_0, g, O through three layers: the object layer, the (grounded) proposition layer, and the predicate layer. In PLGS, object nodes encode type information such that objects of different types have different features. Additionally, propositions are encoded not as a single node with $3 \times |\mathcal{P}|$ potential features, but as a combination of a proposition node (where all proposition nodes share the same feature) and a predicate node (with $|\mathcal{P}|$ potential features). Edges connecting these nodes can take on one of three colours, corresponding to ag, ap, and ug. This modification reduces the feature vector's dimensionality by $2 \times |\mathcal{P}| - 1$ without losing expressiveness, as the complexity is shifted onto the edges. The edges connecting proposition nodes to object nodes are the same as in ILG, representing the index of the object as a parameter in the proposition it connects to.

In contrast, the domain subgraph captures domain-specific information, especially how $\mathcal{P}$ and $\mathcal{A}$ interact. This subgraph includes three layers: predicate, outcome, and action

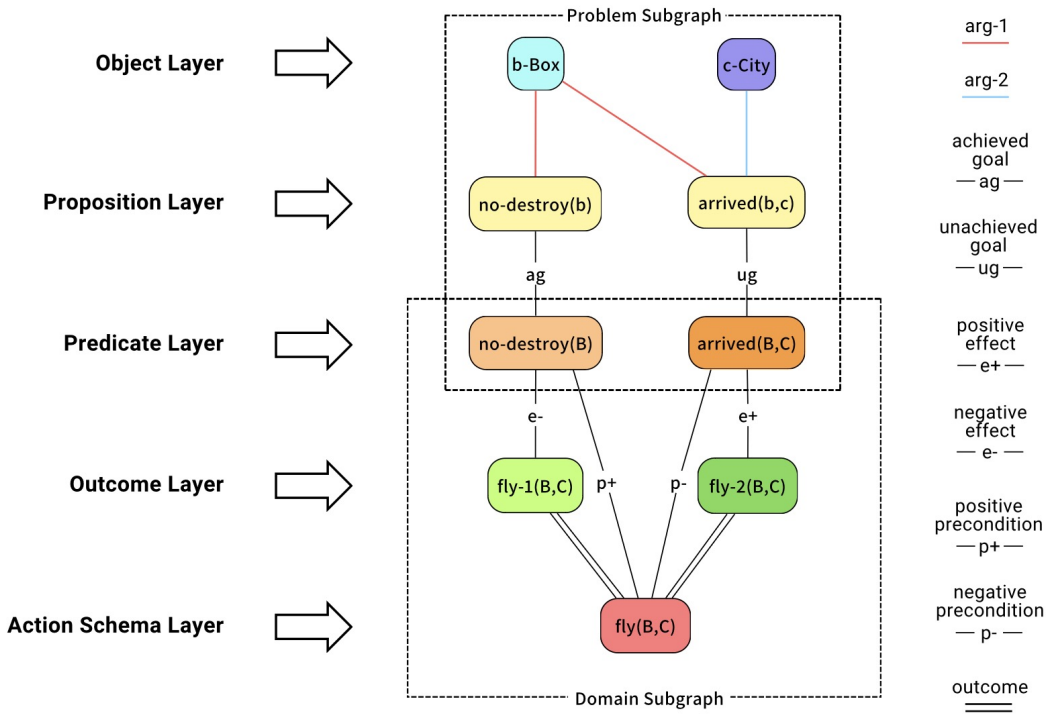

Figure 4.2: PLGS representation of $\mathbb{S}^L_{\mathbf{box}}$ (with only action fly). Different colours represent different node features. Edges are labelled as per Definition 19. PLGS can be interpreted as two subgraphs, referred to as "Problem subgraph" and "Domain subgraph".

schema. Each predicate and action schema, along with their associated outcomes, is encoded as a node in a lifted format (without instantiation by objects). Each predicate and action schema node has a unique feature, while outcome nodes are characterised by the probability of occurrence, represented as a value in $[0, 1]$ rather than an one-hot encoding. This encoding captures the probabilistic nature of action outcomes, distinguishing SSPs from classical planning problems. Precondition/effect relationships between predicates and action schema/outcome nodes are represented by distinct edges, with an additional outcome edge linking each action schema to its outcomes. For different problems within the same domain, this subgraph remains unchanged.

PLGS is designed to address ILG's limitations when applied to SSPs. It allows objects to have unique features based on their types. As we demonstrate in Section 4.3.4, this added information on object types increases the graph's expressiveness in certain domains. PLGS also changes the way propositions are encoded, reducing the feature dimensionality for propositions by $2 \times |\mathcal{P}| - 1$, making it more suitable as input for GNNs. Furthermore, PLGS incorporates domain-specific knowledge of SSPs within the

domain subgraph, embedding probabilistic information directly within the graph structure. Although the primary focus of this thesis is domain-dependent learning tasks, we expect PLGS to provide improved generalisability in learning-for-planning tasks, especially with limited training data, compared to ILG. This is because problem subgraphs or ILG contain only the objects and propositions present in initial and goal states; with sparse training data, models learned from them may struggle to make predictions on new graphs with previously unseen objects and propositions. The domain subgraph, however, enables the model to learn general interactions between predicates and action schemas across different problems within the same domain, as it remains complete and static for every problem in that domain. This learned information can benefit predictions on new problem subgraphs, as the connections between problem and domain subgraphs via predicate layers also remain static across all problems.

PLGS is larger than ILG due to the inclusion of the domain subgraph, which requires additional computational resources for training and evaluation. However, this can be mitigated by pre-computing the domain subgraph for the entire problem domain, as we discuss later in Section 2.4.1. By incorporating domain subgraphs, PLGS has the potential to be further adapted for domain-independent learning-for-planning tasks, similar to what Chen et al. [2023a] achieved with LLG. Although this is beyond the scope of this thesis, we explore it as a potential direction for future work in Section 6.2.

4.2.3 Probabilistic Learning Graph Large

In PLGS, the domain subgraph and problem subgraph can be considered as two distinct parts, meaning we do not expect the model to learn directly from the domain subgraph in ways that enhance learning on the problem subgraph. Instead, we expect the model to learn relationships between various edges and nodes within the domain subgraph across multiple problems within the same domain, which could indirectly support learning in the problem subgraph through the connecting edges. For instance, in MPNN, each node is influenced only by its neighbouring nodes within each training iteration, so it would take four iterations for information from an action schema node to reach an object node.

One motivation for improving this structure is that, although predicates and actions are lifted within the domain subgraph, we still expect the interactions between action parameters and lifted predicates for outcome effects to mirror those that occur if they were instantiated, i.e., they should interact the same way as actual objects do with instantiated grounded propositions. For instance, edges representing parameter indices connecting predicates and parameter variables should be the same as those connecting objects and propositions. If we can learn how these index edges interact among various predicate schemas and parameters, we can then apply this edge knowledge gained from domain subgraphs to new, unseen problem subgraphs which encode the same index edges. This approach allows the information learned on the domain subgraph across different problems to be directly beneficial for the problem subgraph, further enhancing generalisability in learning-for-planning under uncertainty. Following this motivation, we extend PLGS to create the *Probabilistic Learning Graph Large* (PLGL) as follows:

Definition 20 (Probabilistic Learning Graph Large)**.** A Probabilistic Learning Graph Large (PLGL) representation of a lifted SSP $\mathbb{S}^L = \langle\langle \mathcal{P}, \mathcal{A}\rangle, O, s_0, g\rangle$ is the graph $G^{PLGL} = \langle V, E, \omega, \kappa, \Sigma\rangle$ where:

- We use $G^{PLGS} = \langle V^{\text{PLGS}}, E^{\text{PLGS}}, \omega^{\text{PLGS}}, \kappa^{\text{PLGS}}, \Sigma^{\text{PLGS}}\rangle$ to denote the features from PLGS.
- $V = V^{\text{PLGS}} \cup N'(\mathcal{A}) \cup N(\mathcal{P})$ is an ordered set of nodes where:
  - $N'(\mathcal{A}) = \{\delta_{i,j} \mid a \in \mathcal{A}, \text{outcome}(a_i) \in \text{outcomes}(a), \delta_{i,j} \in \Delta a\}$ is the set of outcome parameter nodes, containing parameters for each action schema outcome.
  - $N(\mathcal{P}) = \bigcup_{f\in\{pre_+,pre_-\}}\{P_f \mid P_f \in \mathcal{P}, a \in \mathcal{A}, P_f \in f(a)\} \cup \bigcup_{e\in\{add,del\}}\{P_e \mid P_e \in \mathcal{P}, a \in \mathcal{A}, \text{outcome}(a_i) \in \text{outcomes}(a), P_e \in e(a_i)\}$ is the set of predicate schema nodes containing all the predicate schemas including preconditions of action schemas and effects from action outcomes.
- $E = E_{param} \cup E_{out} \cup \bigcup_{e\in\{add,del\}} E_e \cup \bigcup_{f\in\{pre_+,pre_-\}} E_f \cup E_{ground}$ is a set of edges where:
  - $E_{param} = E^{\text{PLGS}}_{param} \cup \{(\delta_{i,j}, a_i) \mid a \in \mathcal{A}, \text{outcome}(a_i) \in \text{outcomes}(a), \delta_{i,j} \in \Delta a\} \cup \bigcup_{e\in\{add,del\}} \bigcup_{p=P(\delta_{i,1},\ldots,\delta_{i,n_p})\in e(a_i), n_p\geq 1}\{(\delta_{i,j}, P_e) \mid a \in \mathcal{A}, \text{outcome}(a_i) \in \text{outcomes}(a)\}$ is the set of parameter edges linking object nodes with proposition nodes as in PLGS; linking predicate schema nodes with parameter nodes; and parameter nodes with action outcome nodes.
  - $E_{out} = E^{\text{PLGS}}_{out}$ is the set of outcome edges linking each action schema node with its outcome nodes as in PLGS.
  - $E_e = \bigcup_{P\in e(a)}\{(a_i, P_e), (P_e, P) \mid a \in \mathcal{A}, \text{outcome}(a_i) \in \text{outcomes}(a)\}$ is the set of effect edges linking outcome nodes with their corresponding add/delete effect predicate schema nodes and linking predicate schema nodes with corresponding predicate nodes.
  - $E_f = \bigcup_{P\in f(a)}\{(a, P_f), (P_f, P) \mid a \in \mathcal{A}\}$ is the set of precondition edges linking action schema nodes with their corresponding positive/negative precondition predicate schema nodes and linking those predicate schema nodes with corresponding predicate nodes.
  - $E_{ground} = E^{\text{PLGS}}_{ground}$ is the set of grounding edges linking each grounded proposition node to its predicate node as in PLGs.
- $\omega : V \to \mathbb{R}^d$ where $d = d^{\text{PLGS}} + 1$. For feature vector $\omega(v) \in \mathbb{R}^d$, $v \in V$, we use $\omega(v)[i]$ to denote the i-th element of $\omega(v)$. The following holds:
  - $\omega(v)[1] = 1$ if v is a ground proposition node, i.e., $v \in s_0 \cup g$.
  - $\omega(v)[2] = 1$ if v is a predicate schema node, i.e., $v \in \bigcup_{f=\{pre_+,pre_-,add,del\}} P_f$.

  - $\omega(v)[2+j] = 1$ if v is an object node $v \in O$ or if v is a parameter variable node, i.e., $v = \delta_i$; meanwhile the type of v is the j-th element within the ordered set types(O).
  - $\omega(v)[2+\text{types}(O)+j] = 1$ if v is the j-th element of the ordered set $\mathcal{P}$, indicating it is a predicate node.
  - $\omega(v)[2 + \text{types}(O) + |\mathcal{P}| + j] = 1$ if v is the j-th element of the ordered set $\mathcal{A}$, indicating it is an action schema node.
  - $\omega(v)[-1] = prob_{a_j}$ if $v = a_j$ is any outcome node with probability $prob_{a_j}$.
- $\Sigma = \Sigma^{\text{PLGS}} = \{1, 2, \ldots, (I+8)\}$ where $I = \arg\max_{P \in \mathcal{P}} n_P$ denotes the maximum number of parameters within any single predicate $P \in \mathcal{P}$. The total number of colours is $I + 8$, each representing the same as in PLGS.
- $\kappa : E \mapsto \Sigma$ is the colour function where:
  - $\kappa(p, o_i) = \kappa(\delta_{j,i}, a_i) = i$ for $(p, o_i), (\delta_{j,i}, a_i) \in E_{param}$.
  - $\kappa(\delta, P) = i$ for $(\delta, P) \in E_{param}$ and δ is the i-th argument of P.
  - $\kappa(e) = I + 1$ for $e \in E_{out}$.
  - $\kappa(e) = I + 2$ for $e \in E_{eff_+}$.
  - $\kappa(e) = I + 3$ for $e \in E_{eff_-}$.
  - $\kappa(e) = I + 4$ for $e \in E_{pre_+}$.
  - $\kappa(e) = I + 5$ for $e \in E_{pre_-}$.
  - $\kappa(p, P) = I + 6$ for $p \in s_0 \cap g$, indicating p is ag.
  - $\kappa(p, P) = I + 7$ for $p \in s_0 \setminus g$, indicating p is ap.
  - $\kappa(p, P) = I + 8$ for $p \in g \setminus s_0$, indicating p is ug.

■

Figure 4.3 provides an illustrative example of the PLGL representation for $\mathbb{S}^L_{\textbf{box}}$. Intuitively, PLGL can also be divided into a *problem subgraph* and a *domain subgraph*, where the problem subgraph of PLGL uses the same encoding as in PLGS.

The domain subgraph of PLGL extends the domain subgraph of PLGS by adding two additional layers: an action parameter layer and a predicate schema layer. The predicate schema nodes represent either positive/negative preconditions or add/delete effects that are required or produced when applying an action schema. For each action outcome, we define corresponding action parameter nodes. A single feature represents all predicate schema nodes, as in the grounded proposition layer, while parameter nodes are defined based on object type features to match their corresponding types. All other nodes are encoded as in PLGS. Relationships between these nodes follow the encoding of

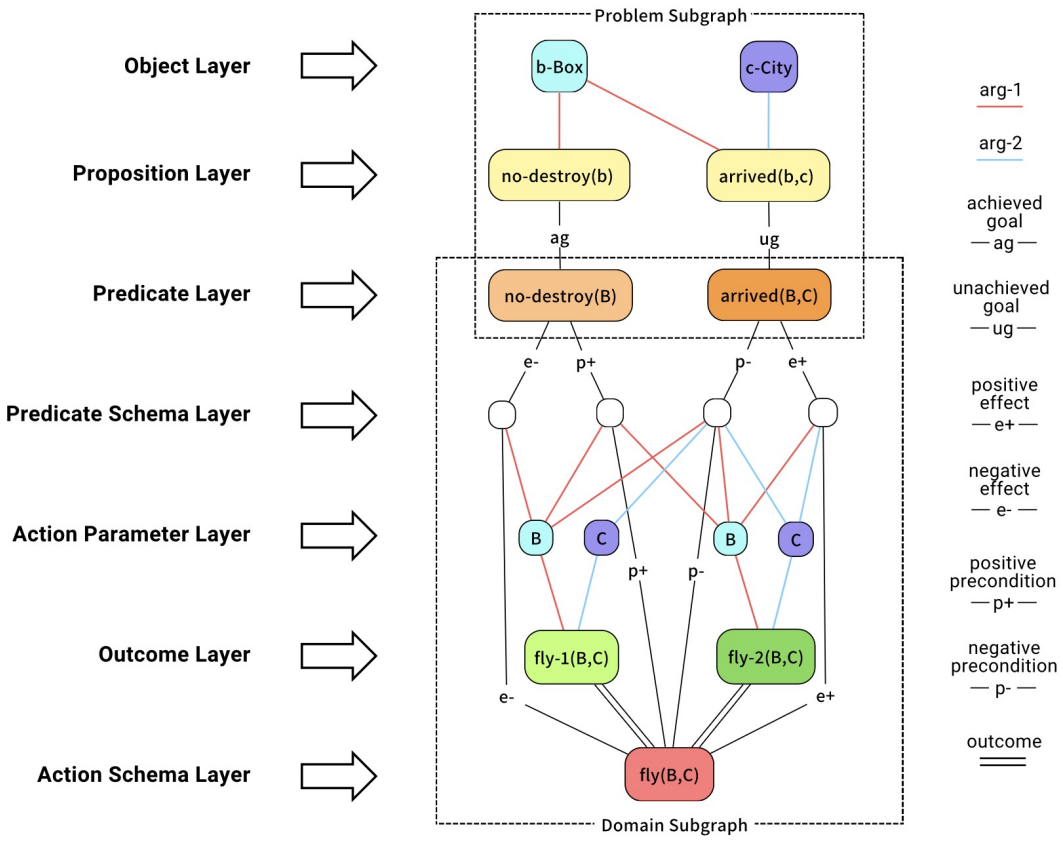

Figure 4.3: PLGL representation of $\mathbb{S}^L_{\mathbf{box}}$ (with only action fly). Different colours represent different node features. Edges are labelled as per Definition 20. PLGL can also be interpreted as two subgraphs, referred to as "Problem subgraph" and "Domain subgraph".

outcome edges and precondition/effect edges in PLGS, with the exception that predicate schema nodes act as intermediaries between precondition/effect predicate nodes and their corresponding action/outcome nodes. Additionally, predicate schema nodes connect to parameter nodes, and parameter nodes connect to outcome nodes, both using the same parameter index edges as in the problem subgraph. This structure aligns with the definition of action schema $\mathcal{A}$ within the lifted SSP $\mathbb{S}^L$ (Definition 17). As in PLGS, the domain subgraph remains unchanged for different problems within the same domain.

As discussed at the beginning of this subsection, PLGL enables knowledge learned from the domain subgraph to directly benefit the interpretation of the problem subgraph. This is achieved through learning how action parameters interact with predicate schemas, providing insights into both the index edges and the object nodes used to encode the problem subgraph. Since the domain subgraph is complete and encodes all possible ways

predicates can be affected by action schemas, we expect PLGL to achieve enhanced generalisability in learning-for-planning tasks, particularly when making predictions on new graphs with previously unseen propositions. Given that PLGL extends PLGS without omitting any encoding, we also expect it to be at least as expressive as PLGS for problems within the same domain; this is formally proved in Section 4.3.4. A potential drawback is that PLGL is larger than PLGS due to the inclusion of two additional layers within the domain subgraph. However, this increase in size can be mitigated by pre-computing the domain subgraph for the entire problem domain. Like PLGS, PLGL also has potential applications in domain-independent planning, which we will discuss in the future work section.

It is worth noting that Definitions 18, 19, and 20 can all be viewed as definitions for graph representation functions: $\mathcal{G}^{ILG} : \mathbb{S}^{cL} \mapsto G$, $\mathcal{G}^{PLGS} : \mathbb{S}^{L} \mapsto G$, and $\mathcal{G}^{PLGL} : \mathbb{S}^{L} \mapsto G$, respectively. Each of these functions translates a lifted planning task into a weighted multi-graph representation. In the next section, we will formally demonstrate how our framework, extended from GOOSE, enables regression tasks to be performed on these novel graph representations.

4.3 Regression Tasks with Learning Graphs

Recall from Section 3.3 that our second research objective is to extend GOOSE to handle SSPs and enable various models to learn heuristic functions for $p^{\max}$. With our graph representations, we can formally define the task that address this research objective as a *Max-Prob graph regression task*, extending the regression tasks from Definition 13 as follows:

Definition 21 (Max-Prob Graph Regression Task)**.** A Max-Prob graph regression task involves estimating a function $\mathcal{F}$ that maps a graph representation of a lifted SSP $\mathbb{S}^L$ to the maximum probability of reaching the goal in this problem, denoted as $p^{\max}$, formally:

Given a set of training data $\{(\mathbb{S}^L_n, p^{\max}_n)\}_{n=1}^N$ and a (stochastic) graph representation function $\mathcal{G} : \mathbb{S}^L \mapsto G$ where $p^{\max}_n$ is the maximum probability of reaching the goal in $\mathbb{S}^L_n$, and every problem $\mathbb{S}^L_n$ shares the same domain. The objective is to find a function $\mathcal{F} : G \mapsto \mathbb{R}$ such that, for any new input $\mathbb{S}^{L*}$ from the same problem domain with its corresponding target $p^{\max*}$, the function outputs $p^{\hat{\max}} = \mathcal{F}(\mathcal{G}(\mathbb{S}^{L*}))$, where $p^{\hat{\max}}$ is an accurate prediction of $p^{\max*}$. In the remainder of this thesis, we refer to such an $\mathcal{F}$ as a "Max-Prob regression model". ■

To solve the Max-Prob graph regression task, we extend the GOOSE framework from Chen et al. [2023a] and Chen et al. [2024] to build two frameworks, allowing either a GNN model or an SML model to be trained and serve as $\mathcal{F}$. We describe the GNN approach in Section 4.3.1 and the SML approach in Section 4.3.2. Additionally in Section 4.3.3, we provide a framework that uses all-outcome determinisation to enable classical planning graphs, including ILG and LLG, to be parsed as graph representations within the same Max-Prob graph regression task. Section 4.3.4 concludes our second research objective

with a theoretical analysis of ILG, PLGS, and PLGL in terms of their expressiveness on a Max-Prob graph regression task.

4.3.1 Learning Through GNN Models

In this section, we describe how an MPNN (Section 2.4.2) can be used to perform the Max-Prob graph regression task by creating a learning framework based on the approach in Chen et al. [2023a]. We use $\mathcal{FM}_\Theta : G \mapsto \mathbb{R}$ to denote an MPNN regression model with parameters Θ that takes input in the form of a weighted multi-edge graph G. Algorithm 2 outlines the learning framework to train a Max-Prob MPNN regression model for solving the Max-Prob graph regression task using the graph transition function $\mathcal{G}$, training data $\{(\mathbb{S}^L_n, p^{\max}_n)\}^N_{n=1}$, and any training algorithm $\texttt{train}(M, X, Y)$ where M is the model, and X and Y are lists of features and labels, respectively. For simplicity, we use $\mathcal{G}_d(\mathbb{S}^L)$ to represent the transition function that outputs only the domain subgraph G^{PLG}_d; $\mathcal{G}_p(\mathbb{S}^L)$ outputs only the problem subgraph G^{PLG}_p; and $\mathcal{G}(\mathbb{S}^L)$ outputs the entire PLG $G^{PLG} = G^{PLG}_d + G^{PLG}_p$.

Algorithm 2: Graph learning framework with MPNN

Input : Training dataset $D = \{(\mathbb{S}^L_n, p^{\max}_n)\}^N_{n=1}$; An MPNN model framework $\mathcal{FM}_\Theta$; A graph representation function $\mathcal{G} : \mathbb{S}^L \mapsto G^{PLG}$; Any training algorithm for MPNN $\texttt{train}(M, X, Y)$.

Output: A Max-Prob MPNN regression model

1 **Procedure** `LeaningMPNN`$(D, \mathcal{FM}_\Theta, \mathcal{G}, \textit{train()})$
2 $X \leftarrow [\,]$; *Phase 0*
3 $Y \leftarrow [\,]$;
4 $\mathbb{S}^L \leftarrow D[0][0]$; *Phase 1*
5 $G^{PLG}_d \leftarrow \mathcal{G}_d(\mathbb{S}^L)$;
6 **for** $(\mathbb{S}^L, p^{max}) \in D$ **do**
7 $G^{PLG}_p \leftarrow \mathcal{G}_p(\mathbb{S}^L)$; *Phase 2*
8 $G^{PLG} \leftarrow G^{PLG}_d + G^{PLG}_p$; *Phase 3*
9 $X \mathrel{+}= G^{PLG}$;
10 $Y \mathrel{+}= p^{\max}$;
11 $\texttt{train}(\mathcal{FM}_\Theta, X, Y)$; *Phase 4*
12 **return** $\mathcal{FM}_\Theta$

Algorithm 2 can be divided into four phases, as follows: in phase 1, we extract the first SSP from the training data and compute the domain subgraph (Lines 4–5). Since each problem shares the same domain, the domain subgraph only needs to be computed once. In phase 2, for each SSP in the training data, we compute the problem subgraph (Line 7). In phase 3, we combine the domain and problem subgraphs to form a PLG as an input feature with the corresponding label being the $p^{\max}$ value for that SSP; this feature-value pair is then appended to the MPNN dataset (Lines 8–10). In phase 4, we

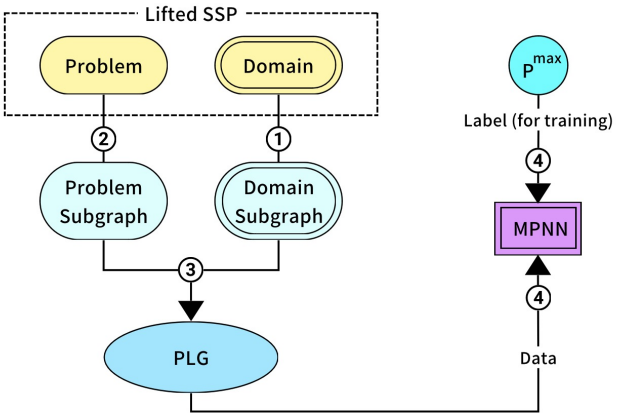

Figure 4.4: An illustration of the MPNN learning framework (Algorithm 2) that enables an MPNN to learn to predict $p^{\max}$ from a dataset of (SSP, $p^{\max}$) pairs. Each phase is visualised as an explicit edge to facilitate understanding.

use the constructed MPNN dataset to train the MPNN and obtain a Max-Prob MPNN regression model as the solution to the Max-Prob graph regression task (Lines 11–12). The framework enables the domain subgraph to be computed only once throughout the entire process, thus mitigating the efficiency overhead introduced by structural complexity. In the actual implementation, the domain subgraph is stored as part of the model structure once computed, and keep reused along the entire training process to further enhance efficiency. Figure 4.4 illustrates the four phases of the framework on a dataset D with a single (SSP, $p^{\max}$) pair.

4.3.2 Learning Through SML models

In this section, we describe how to use a GPR model (Section 2.4.1) to perform the Max-Prob graph regression task by constructing a learning framework following the approach of Chen et al. [2024]. We selected GPR because it has been demonstrated as one of the most efficient models among traditional SML regression models in Chen et al. [2024]. Although our framework can be applied to any SML regression model, unlike MPNN models, which allow graphs to be directly parsed as input, GPR models require input in the form of a feature vector $\mathbb{R}^d$. Therefore, we first explain how to transform a PLG into a feature vector.

We start by introducing a modified 1-WL algorithm in Algorithm 3. The modified 1-WL algorithm operates on a coloured multi-graph $G = \langle V, E, \omega, \kappa, \Sigma \rangle$, where $\omega : V \mapsto \mathbb{N}$ is the node colouring function. The only difference between the modified 1-WL and the original 1-WL in Algorithm 1 is that it incorporates coloured edges during colour refinement (Line 5). For simplicity, we will refer to Algorithm 3 as WL in the following sections.

Algorithm 3: Modified 1-WL Algorithm

Input : Coloured multi-graph $G = \langle V, E, \omega, \kappa, \Sigma \rangle$ where $\omega : V \mapsto \mathbb{N}$ is the node colouring function; A max iteration number J

Output: A multiset of node colours seen over all iterations.

Procedure `WLAlgorithm`(G, J)

  $\omega^0(v) \leftarrow \omega(v), \forall v \in V$ // *Initial coloring for all nodes*

  for $j = 1, \ldots, J$ **do**

    for $v \in V$ **do**

      $\omega^j(v) \leftarrow \text{hash}(\omega^{j-1}(v), \bigcup_{\iota \in \Sigma} \{\!\{(\omega^{j-1}(u), \iota) \mid u \in \mathcal{N}_\iota(v)\}\!\})$

  return $\bigcup_{j=0,\ldots,h} \{\!\{\omega^j(v) \mid v \in V\}\!\}$

Chen et al. [2024] and Shervashidze et al. [2011] present a method that translates the multiset of colours output by the WL algorithm into a feature vector by representing these multisets as histograms. The feature vector of a graph is a vector $\vec{v}$ whose size corresponds to the number of colours observed during training, where $v[k]$ counts how often the WL algorithm encounters each colour. Formally, let $G_1, G_2, \ldots, G_n$ be the set of training graphs where $G_i = \langle V_i, E_i, \omega_i, \kappa_i, \Sigma_i \rangle$. Then, the colours encountered by the WL algorithm in the training graphs are given by

$$\mathcal{C} = \{\omega_i^j(v) \mid i \in \{1, \ldots, n\}; j \in \{0, \ldots, J\}; v \in V_i\}$$

where $\omega_i^j(v)$ is the colour of node v in graph G_i during the j-th iteration of WL for $j > 0$ and $c_i^0(v) = c_i(v)$. Given a new graph G' and the set of colours $\mathcal{C}$ observed during training, we denote the feature vector function $\texttt{feat}_\mathcal{C} : G \mapsto \mathbb{R}^{|\mathcal{C}|}$ formally as in Equation 4.1.

$$\texttt{feat}_\mathcal{C}(G') = [\texttt{count}_\mathcal{C}(G', \mathcal{C}[0]), \texttt{count}_\mathcal{C}(G', \mathcal{C}[1]), \ldots, \texttt{count}_\mathcal{C}(G', \mathcal{C}[-1])] \tag{4.1}$$

where $\texttt{count}_\mathcal{C}(G', \mathcal{C}[i])$ is the number of times the colour $\mathcal{C}[i]$ appears in the output after running WL on G'. There is no guarantee that $\mathcal{C}$ contains all possible observable colours within a given planning domain; therefore, any colours not in $\mathcal{C}$ that are encountered after training are ignored.

Now that we have a transformation function allowing a coloured multi-graph to be represented as a feature vector, we illustrate how a PLG can be represented as a coloured multi-graph.

Definition 22 (PLG as a Coloured Multi-Graph)**.** A PLG $G^{PLG} = \langle V, E, \omega, \kappa, \Sigma \rangle$ can be represented as a coloured multi-graph $G^{PLG'} = \langle V, E, c, \kappa, \Sigma \rangle$ with the new colouring function $c : V \mapsto \mathbb{N}$ as defined in Equation 4.2. ■

$$c(v) = \begin{cases} C_p[\texttt{round}(B, prob_v)] & \text{if } v \in \mathcal{N}(\mathcal{A})_o \\ \texttt{idx}(\omega(v), 1) & \text{otherwise} \end{cases} \tag{4.2}$$

where:

- d is the dimension of feature vectors in G^{PLG}.
- $\mathcal{N}(\mathcal{A})_o$ denotes the set of all outcome nodes.
- $\texttt{idx}(\vec{v}, 1)$ is the index function that returns the position of the entry with value 1 in a one-hot encoding vector $\vec{v}$.
- $C_p = \{d-1, d, d+1, \ldots, d+B-2\}$ denotes the set of "probability bucket colours" of size B, indicating there are B different colours representing probabilities in $[0, 1]$. Each colour bucket $C_p[i]$ represents a probability interval $[i/B, (i+1)/B)$. For example, if $B = 20$, then $C_p[3]$ represents the probability interval $[0.15, 0.2)$.
- $\texttt{round}(B, prob)$ rounds the probability $prob$ to its corresponding colour bucket index. For example, $\texttt{round}(20, 0.17) = 3$.

Algorithm 4: Graph learning framework with GPR

Input : Training dataset $D = \{(\mathbb{S}_n^L, p_n^{\max})\}_{n=1}^N$; A GPR model framework $\mathcal{FG}_\Theta$; A graph representation function $\mathcal{G} : \mathbb{S}^L \mapsto G^{PLG'}$; Any training algorithm for GPR $\texttt{train}(M, X, Y)$; Max WL iteration J.

Output: A Max-Prob GPR model

Procedure $\texttt{LeaningGPR}(D, \mathcal{FG}_\Theta, \mathcal{G}, \texttt{train()}, J)$

- $X \leftarrow [\,]$; *Phase 0*
- $Y \leftarrow [\,]$;
- $\mathbb{S}^L \leftarrow D[0][0]$; *Phase 1*
- $G_d^{PLG'} \leftarrow \mathcal{G}_d(\mathbb{S}^L)$;
- **for** $(\mathbb{S}^L, p^{max}) \in D$ **do**
  - $G_p^{PLG'} \leftarrow \mathcal{G}_p(\mathbb{S}^L)$; *Phase 2*
  - $G^{PLG'} \leftarrow G_d^{PLG'} + G_p^{PLG'}$; *Phase 3*
  - $X \mathrel{+}= G^{PLG'}$;
  - $Y \mathrel{+}= p^{\max}$;
- $\mathcal{C} \leftarrow \bigcup_{x \in X} \texttt{WLAlgorithm}(x, J)$ *// Algorithm 3* *Phase 4*
- $X' \leftarrow \bigcup_{x \in X} \texttt{feat}_\mathcal{C}(x)$ *// Equation 4.1*
- $\texttt{train}(\mathcal{FG}_\Theta, X', Y)$; *Phase 5*
- **return** $\mathcal{FG}_\Theta \circ feat_\mathcal{C}$

Intuitively, since every non-outcome node uses a one-hot encoding to represent node features, we assign each colour to a unique node feature for nodes not in the outcome set, $v \notin \mathcal{N}(\mathcal{A})_o$. For the action outcome nodes $v \in \mathcal{N}(\mathcal{A})_o$, which have a non-zero feature $prob_v$ representing their probability of occurrence, we use a "bucket discretisation" method with B colour buckets to represent all possible probabilities and assigning each $prob_v$ to its corresponding bucket. In practice, this approach preserves probabilistic information if B is chosen to capture the smallest gap between any two probability values.

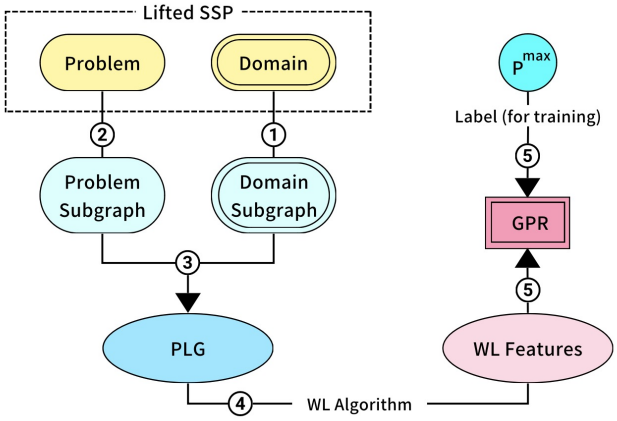

Figure 4.5: An illustration of the GPR learning framework (Algorithm 4) that enables a GPR to learn to predict $p^{\max}$ from a dataset of (SSP, $p^{\max}$) pairs. Each phase is visualised as an explicit edge to facilitate understanding

For example, in the IPC [Vallati et al., 2015], the smallest probabilistic effect unit is 0.05, so setting $B = 20$ ensures no loss of information during transformation.

Now, with the WL features and a coloured multi-graph representation of PLG, we can build a new graph learning framework with GPR. We denote a GPR model with parameters Θ as $\mathcal{FG}_\Theta : \mathbb{R}^c \mapsto \mathbb{R}$, which takes an input in the format of vector feature $x \in \mathbb{R}^c$ where c is undetermined yet. Algorithm 4 outlines the learning framework for training a Max-Prob GPR model to solve the Max-Prob graph regression task. The notation aligns with that used in Section 4.3.1, except that the graph transition function $\mathcal{G} : \mathbb{S}^L \mapsto G^{PLG'}$ now returns a coloured multi-graph representation of PLG (Definition 22).

Algorithm 4 can be divided into five phases: phases 1–3 (Lines 4–10) are identical to those in the MPNN learning framework in Algorithm 2. In phase 4, we first apply the WL algorithm to compute the feature generation function, then use this function to transform the training PLG set into a set of feature vectors. These feature vectors, along with the corresponding $p^{\max}$ values, form the GPR training dataset (Lines 11–12). In phase 5, we use the GPR training set to train the GPR model and obtain a Max-Prob GPR model. This model, as a composition of the trained GPR model with the feature generation function, serves as the solution to the Max-Prob graph regression task (Lines 13–14). Like the MPNN learning framework, the GPR framework also requires only one computation of the domain subgraph along the entire training process. Figure 4.5 provides an illustrative example of the five phases of the framework using a dataset D containing a single (SSP, $p^{\max}$) pair.

4.3.3 Learning Through Determinisation

In the previous two sections, we defined how to use our PLG representation to build two learning-for-planning frameworks that allow either GPR or MPNN models to predict $p^{\max}$ and thereby solve the Max-Prob Graph Regression Task (Definition 21). In this section, we propose a simple modification to both frameworks that enables a classical planning graph representation to be used within the same learning-for-planning frameworks through all-outcome determinisation on lifted SSPs.

All-outcome determinisation of a lifted SSP $\mathbb{S}^L$ follows the same process as introduced at the end of Section 2.1.2 (for all-outcome determinisation on a general SSP $\mathbb{S}$). The main idea is to transform each possible action outcome into a distinct action schema. Formally:

Definition 23 (All-Outcome Determinisation on Lifted SSPs)**.** All-outcome determinisation of a lifted SSP $\mathbb{S}^L = \langle\langle \mathcal{P}, \mathcal{A}\rangle, O, s_0, g\rangle$, denoted as $\texttt{det}(\mathbb{S}^L)$, returns a classical lifted planning problem $\mathbb{S}^{cL} = \langle\langle \mathcal{P}, \mathcal{A}'\rangle, O, s_0, g\rangle$ where:

- $\mathcal{P}, O, s_0, g$ remain unchanged.
- For each action outcome a_i where $a \in \mathcal{A}$ and $\text{outcome}(a_i) \in \text{outcomes}(a)$, we create a new action schema $\langle \Delta(a), \text{pre}(a), \text{add}(a_i), \text{del}(a_i), \text{cost}(a)\rangle$ in $\mathcal{A}'$.

■

We can now compose the all-outcome determinisation function $\texttt{det}(\mathbb{S}^L)$ with any classical graph representation function $\mathcal{G} : \mathbb{S}^{cL} \mapsto G$ (e.g., $\mathcal{G}^{ILG}$, $\mathcal{G}^{LLG}$) to create a stochastic graph representation function $\mathcal{G} \circ \texttt{det} : \mathbb{S}^L \mapsto G$. This resulting function can be directly used as input for either the MPNN learning framework (Algorithm 2) or the GPR learning framework (Algorithm 4), depending on whether G is a weighted multi-graph or a coloured multi-graph. For example, if we combine ILG's graph representation function with `det`, the resulting function $\mathcal{G}^{ILG} \circ \texttt{det}$ becomes a stochastic graph representation function that can directly be used as input for the MPNN learning framework.

Since ILG encodes only problem-specific information from $s_0, g, O, \mathcal{P}$, which remains unaffected by all-outcome determinisation, using ILG in a Max-Prob graph regression task with all-outcome determinisation does not impact ILG's expressiveness or its representation of the original SSP ($\mathbb{S}^L$) in learning-for-planning tasks. This enables us to directly compare the expressiveness of ILG and PLG in the next section. However, for some other classical planning graph representations such as LLG [Chen et al., 2023a] which rely on $\mathcal{A}$ in their graph encoding, all-outcome determinisation prevents them from representing the original SSP in the same way as PLG, as probabilistic information of various effects is lost in the determinisation process.

4.3.4 Expressiveness of Graphs

In Section 4.2, we stated that ILG is less expressive than PLGS in learning-for-planning tasks under uncertainty, while PLGL is at least as expressive as PLGS. Here, we formally

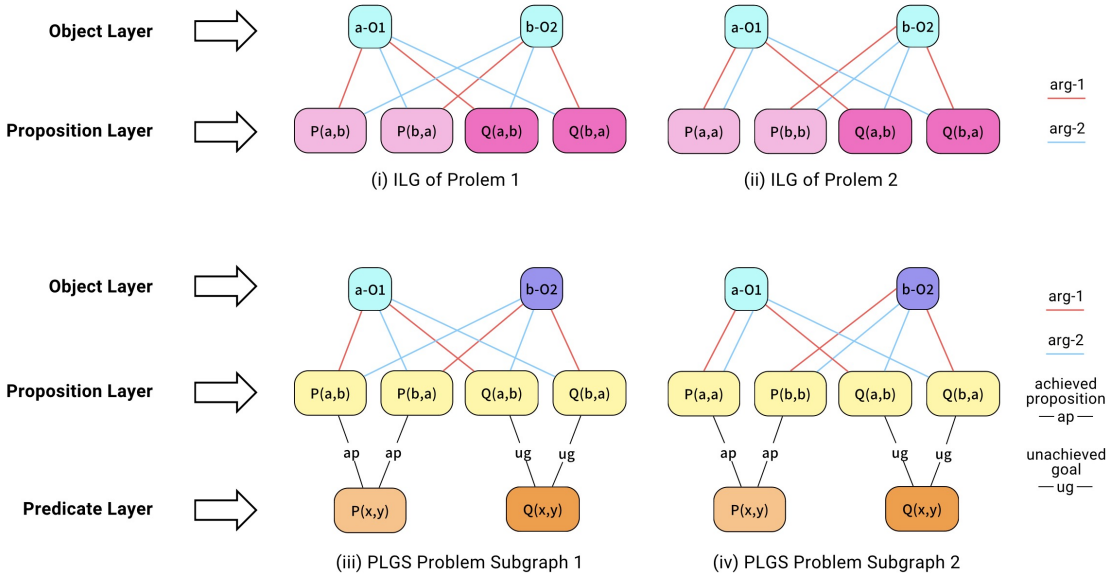

Figure 4.6: ILG and PLGS representations of two SSPs as presented in proof (second part of Theorem 2). Models learning from the ILG representation fail to distinguish between two problems, as both (i) and (ii) generate the same WL features (Algorithm 3). In contrast, PLGS allows for the generation of different WL features for (iii) and (iv) due to the presence of an additional object type feature distinguishing a and b as neighbours of proposition nodes with different features.

prove these statements using the frameworks discussed earlier. Since Chen et al. [2024]; Xu et al. [2018] have already shown that both $\mathcal{FM}$ and $\mathcal{FG}$ have equal power in distinguishing planning tasks when given the same graph structure as input, we will use $\mathcal{F}$ as either the MPNN or GPR learning-for-planning framework within our proof, focusing on comparing the expressiveness of different graph structures. We use $\mathcal{F}_{\Phi}^{\{PLGS,PLGL\}}$ to denote the framework with parameters Φ which use the graph representation functions $\mathcal{G}^{\{PLGS,PLGL\}}$.

As discussed in the previous section, all-outcome determinisation does not affect the expressiveness of ILG on SSPs since it does not modify $s_0, g, O, \mathcal{P}$. This enables direct comparison of ILG's expressiveness with PLG's on $\mathcal{F}_{\Phi}^{ILG}$, which denotes the learning framework with parameters Φ using the graph representation function $\mathcal{G}^{ILG} \circ \texttt{det}$. Whenever we refer to ILG on an SSP, we mean the ILG on the all-outcome determinisation of the SSP, omitting the term for brevity. We use set notation $\subseteq, \subsetneq$ to denote expressiveness hierarchy.

Theorem 2. *$\mathcal{F}^{PLGS}$ is strictly more expressive than $\mathcal{F}^{ILG}$ in distinguishing SSPs within the same domain, i.e., (ILG $\subsetneq$ PLG):*

- *(ILG $\subseteq$ PLGS) Let $\mathbb{S}_1^L$ and $\mathbb{S}_2^L$ be any two SSPs from a given domain. For all Φ, if $\mathcal{F}_\Phi^{ILG}(\mathbb{S}_1^L) \neq \mathcal{F}_\Phi^{ILG}(\mathbb{S}_2^L)$, then there exists a corresponding set of parameters Ψ such that $\mathcal{F}_\Psi^{PLGS}(\mathbb{S}_1^L) \neq \mathcal{F}_\Psi^{PLGS}(\mathbb{S}_2^L)$.*
- *(ILG $\not\supseteq$ PLGS) There exists a pair of SSPs $\mathbb{S}_1^L$ and $\mathbb{S}_2^L$ such that there exists a Ψ where $\mathcal{F}_\Psi^{PLGS}(\mathbb{S}_1^L) \neq \mathcal{F}_\Psi^{PLGS}(\mathbb{S}_2^L)$; however, for all Φ, $\mathcal{F}_\Phi^{ILG}(\mathbb{S}_1^L) = \mathcal{F}_\Phi^{ILG}(\mathbb{S}_2^L)$.*

Proof. (ILG $\subseteq$ PLGS). To show that PLGS is at least as expressive as ILG, we demonstrate that the ILG representation of $\mathbb{S}^L$ can always be transformed into a problem subgraph of PLGS. This transformation proceeds as follows: for each object node o in ILG, create an object node o' of default object type for PLGS. For each proposition node $p = P(o_1, o_2, \dots)$ in ILG with goal status $gs \in \{ap, ug, ag\}$, create a predicate node P' and a proposition node $p' = P'(o_1, o_2, \dots)$, and connect the predicate node to the proposition node using an edge labelled with the goal status gs. For each edge (o_i, p) in ILG, create the corresponding edge (o_i', p') in PLGS.

Since each unique proposition feature in ILG maps to a unique combination of proposition, predicate, and grounded edges in PLGS, and since each object node and argument edge maps to a corresponding object node and argument edge meanwhile both ILG and the transformed PLGS subgraph describe the same $\mathbb{S}^L$. Therefore, there exists a parameter pair Φ and Ψ such that $\mathcal{F}_\Phi^{ILG}(\mathbb{S}^L)$ equals $\mathcal{F}_\Psi^{PLGS}(\mathbb{S}^L)$ (e.g., a fully connected layer in MPNN that assigns the same weight to corresponding nodes and edges). Hence, under this parameter pair Φ and Ψ, if $\mathcal{F}_\Phi^{ILG}(\mathbb{S}_1^L) \neq \mathcal{F}_\Phi^{ILG}(\mathbb{S}_2^L)$, then $\mathcal{F}_\Psi^{PLGS}(\mathbb{S}_1^L) \neq \mathcal{F}_\Psi^{PLGS}(\mathbb{S}_2^L)$, proving the first part.

(ILG $\not\supseteq$ PLGS). To show that PLGS is strictly more expressive than ILG, consider two SSPs, $\mathbb{S}_1^L = \langle \mathcal{P}, \mathcal{A}, O, s_{0,1}, g \rangle$ and $\mathbb{S}_2^L = \langle \mathcal{P}, \mathcal{A}, O, s_{0,2}, g \rangle$, with $O = \{a - O1, b - O2\}$, $\mathcal{P} = \{P(x{-}O, y{-}O), Q(x{-}O, y{-}O)\}$, $s_{0,1} = \{P(a, b), P(b, a)\}$, $s_{0,2} = \{P(a, a), P(b, b)\}$, $g = \{Q(a, b), Q(b, a)\}$, and a single action $a \in \mathcal{A}$ with precondition $P(x, y)$ and an outcome that adds effect $Q(x, y)$ with a probability of 1 (i.e., it always succeeds). Consequently, the corresponding $p^{\max}$ will be 1.0 for $\mathbb{S}_1^L$ and 0 for $\mathbb{S}_2^L$, as $Q(a, b)$ cannot be reached from $P(a, a), P(b, b)$. We illustrate this example in Figure 4.6 for both ILG and the PLGS problem subgraph representations. As shown in Chen et al. [2024], the ILG representation $\mathcal{FG}^{ILG}$ fails to distinguish the two problems since Algorithm 3 always return the same WL features for both SSPs. Therefore, for all Φ, $\mathcal{FG}_\Phi^{ILG}(\mathbb{S}_1^L) = \mathcal{FG}_\Phi^{ILG}(\mathbb{S}_2^L)$. This inability to distinguish the two arises because ILG cannot differentiate the object nodes despite their distinct types. In PLGS, however, the WL features for the problem subgraphs differ due to the object type features, allowing GPR to distinguish between the two SSPs, i.e., there exists a Ψ such that $\mathcal{FG}_\Psi^{PLGS}(\mathbb{S}_1^L) \neq \mathcal{FG}_\Psi^{PLGS}(\mathbb{S}_2^L)$. □

Theorem 3. *$\mathcal{F}^{PLGL}$ is at least as expressive as $\mathcal{F}^{PLGS}$ in distinguishing SSPs within the same domain, i.e., (PLGS $\subset$ PLGL): let $\mathbb{S}_1^L$ and $\mathbb{S}_2^L$ be any two SSPs from a given domain. For all Φ, if $\mathcal{F}_\Phi^{PLGS}(\mathbb{S}_1^L) \neq \mathcal{F}_\Phi^{PLGS}(\mathbb{S}_2^L)$, then there exists a corresponding set of parameters Ψ such that $\mathcal{F}_\Psi^{PLGL}(\mathbb{S}_1^L) \neq \mathcal{F}_\Psi^{PLGL}(\mathbb{S}_2^L)$.*

Proof. The proof is straightforward. According to Definitions 19 and 20, the domain subgraph for both PLGS and PLGL does not change across problems in the same domain, meaning that domain-dependent encoding in both PLGS and PLGL is based on the problem subgraphs. Since PLGS and PLGL share the same problem subgraph encoding, for any SSPs $\mathbb{S}_1^L$ and $\mathbb{S}_2^L$ within the same domain, their PLGS and PLGL representations will share identical problem subgraphs. Consequently, if $\mathcal{F}_\Phi^{PLGS}(\mathbb{S}_1^L) \neq \mathcal{F}_\Phi^{PLGS}(\mathbb{S}_2^L)$, there will always be a corresponding parameter set Ψ such that $\mathcal{F}_\Psi^{PLGL}(\mathbb{S}_1^L) \neq \mathcal{F}_\Psi^{PLGL}(\mathbb{S}_2^L)$. □

Combining Theorems 2 and 3, we have theoretically proven that our novel graph representations, PLGS and PLGL, are strictly more expressive in domain-dependent graph regression tasks than the state-of-the-art ILG representation. As a result, our novel graph representations enable more effective learning of SSP features, including $p^{\max}$. The second part of the proof for Theorem 2 serves as a good example where ILG fails to distinguish two SSPs with $p^{\max}$ values of 0 and 1 respectively but PLGS provides distinct encodings for both.

In the next section, we will switch focus back to our first research objective and discuss how to use our designed frameworks to develop a heuristic function for $p^{\max}$ that can be directly applied to any heuristic search algorithm on Max-Prob SSPs.

4.4 Learning to Solve Max-Prob

Recall in Section 3.3, we presented our first research objective as "to develop an effective heuristic for $p^{\max}$ using machine learning techniques that account for probabilistic information, applicable in the Max-Prob stage of i-dual to enhance performance over current state-of-the-art heuristics for $p^{\max}$". In this section, we formally present how can we develop such heuristic using our existing framework designed earlier in this chapter.

In Section 2.3.1, we defined a heuristic $h : S \mapsto \mathbb{R}$ as a function estimating the cost to reach the goal from a given state (Definition 12). To address the first research objective, we can formally define a heuristic that addresses our first research objective as follows:

Definition 24. [Max-Prob heuristic] A Max-Prob heuristic $h_{mp} : \mathbb{S}^L \mapsto [0, 1]$ is a domain-dependent heuristic function for an SSP domain $\mathbb{D}$, where $\mathbb{S}^L = \langle \mathbb{D}, O, s_0, g \rangle$. This function provides an estimate of $p^{\max}$ for $\mathbb{S}_0^L$. ■

Since h_{mp} estimates $p^{\max}$, which always falls within the range $[0, 1]$ as defined in Definition 9, we restrict its co-domain to real numbers within this interval. Furthermore, since $p^{\max}$ serves as the heuristic for SSPs under the Max-Prob criterion, which is a maximisation problem, an admissible h_{mp} will always give an upperbound of $p^{\max}$.

To generate such a heuristic, we can directly use a Max-Prob regression model $\mathcal{F}$ trained with either an MPNN framework (Algorithm 2) or a GPR framework (Algorithm 4) under a graph transition function $\mathcal{G}$. By applying a modification function $\texttt{mod}: \mathbb{R} \mapsto [0, 1]$ to map the predicted result into a heuristic value, we obtain the Max-Prob heuristic.

For example, the simplest approach is $\texttt{clip}(x, 0, 1)$, which clips values of $x > 1$ to 1 and values of $x < 0$ to 0. We formally define the process for creating a Max-Prob heuristic in Algorithm 5: to predict the heuristic value of a SSP, we first translate it into a graph representation, then put this graph as input to a Max-Prob regression model trained on the same graph representation, we than call the a modification function on the model output to obtain the final heuristic value as an estimation of $p^{\max}$ for the input problem.

Algorithm 5: Graph learning heuristic function

Input : A trained Max-Prob regression model $\mathcal{F} : G \mapsto \mathbb{R}$; a graph transition function $\mathcal{G} : \mathbb{S}^L \mapsto G$; a modification function $\texttt{mod}$: $\mathbb{R} \mapsto [0, 1]$.

Output: A Max-Prob heuristic $\mathbb{S}^L \mapsto [0, 1]$

1 **Procedure** $\texttt{GLHeuristic}(\mathcal{F}, \mathcal{G}, \mathit{mod})$

2 **return** $\mathit{mod} \circ \mathcal{F} \circ \mathcal{G}$

With Algorithm 5, we obtain a Max-Prob heuristic $\texttt{GLHeuristic}(\mathcal{F}, \mathcal{G}, \texttt{mod})$ that can be applied to any heuristic search algorithm which requires a Max-Prob heuristic for solving Max-Prob SSPs, such as FERT, or i-dual in solving LP 2. However, since this heuristic is derived by modifying a predicted feature from a machine learning model, we cannot guarantee its admissibility. We will explore and evaluate the potential impact of this on heuristic performance in the second experiment set in Section 5.3.

Chapter 5

Empirical Evaluation

In this chapter, we put everything together to conduct two sets of experiments aligned with the research objectives outlined in Section 3.3. In Section 5.1, We begin by introducing the two testbed domains and the two models that are being used along the entire chapter. Section 5.2 focuses on a comprehensive series of experiments that compare the performance of different models trained under various graph representations in learning $p^{\max}$ across different datasets for both domains. This experimental set covers a wide range of scenarios and, through evaluations across these different contexts, we address Research Objective 2, demonstrating that our PLG representations facilitate more effective feature learning compared to ILG. In Section 5.3, we introduce the second set of experiments, where we focus on a single domain and deploy models trained on different problem instances within that domain as Max-Prob heuristics to guide i-dual in the Max-Prob stage. We compare and evaluate the performance of our model-based heuristic against two benchmarks: a baseline heuristic and the current state-of-the-art heuristic for this domain. By assessing the impact of our heuristics on performance, we address Research Objective 1 and provide insights that may inspire future work.

5.1 Domains and Models

We begin this chapter by discussing the benchmark domains selected for our experiments. As previously noted, this thesis focuses on learning $p^{\max}$ from SSPs particularly in challenging cyclic domains. Unlike classical planning, this area has seen limited research, which also restricts the range of available cyclic, non-resource-constrained SSP domains suitable for our study. Consequently, we select two representative domains as our experimental testbeds, both previously examined by and obtained from Trevizan et al. [2017b]: the *Exploding Blocksworld* and *Triangle Tireworld* domains.

Another key reason for selecting these domains is that as discussed in Section 3.2.2,

the state-of-the-art Max-Prob PDB heuristic fails to outperform the 0-1 $h^{\max}$ heuristic within these environments. Therefore, these domains are also considered some of the most challenging in probabilistic planning, demonstrating a pronounced need for robust heuristic solutions. Our choice enables us to directly compare the performance of our heuristics with the existing state-of-the-art 0-1 $h^{\max}$.

5.1.1 Exploding The Blocks World

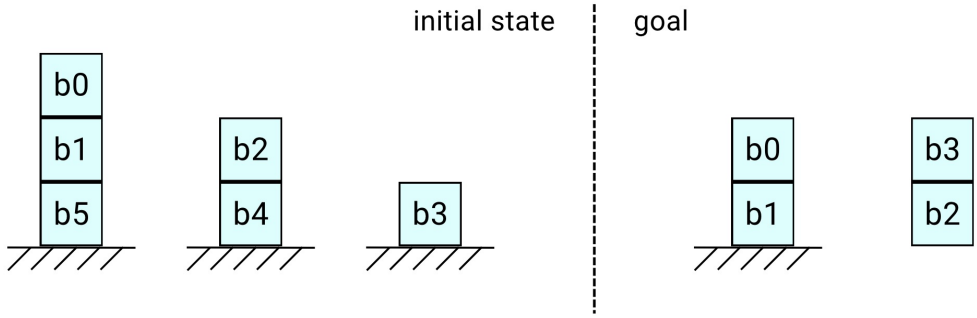

Figure 5.1: An example of an EXBW instance with an initial state (left) consisting of: $b0$ on $b1$ on $b5$ on the table; $b2$ on $b4$ on the table; and $b3$ on the table. The partial goal state (right) specifies the goal of placing $b3$ on $b2$ and $b0$ on $b1$ on table.

Our first testbed domain, Exploding Blocksworld (EXBW) is initially introduced in the 2008 International Planning Competition (IPPC) [Bryce and Buffet, 2008]. EXBW represents a challenging probabilistic extension of the classic deterministic Blocksworld. For a comprehensive overview of the deterministic Blocksworld domain, we refer readers to Slaney and Thiébaux [2001]; this thesis focuses on EXBW as an SSP, assuming the audience is familiarity with the Blocksworld concept. Figure 5.1 provides an illustrative example of an EXBW task.

EXBW extends the traditional deterministic blocks world by introducing the possibility of block explosions that can destroy other blocks or even the table itself. While all actions retain the same primary effects as their deterministic counterparts, the *put-down* and *put-on-block* actions in EXBW include probabilistic side effects. Specifically, there is a 0.4 probability that the block currently held will detonate, potentially destroying the table, and a 0.1 probability of damaging the block directly beneath.

Once a block or the table is destroyed, they become non-functional: objects cannot be placed on them, and destroyed blocks are immovable. As a result, most configurations in EXBW can lead to unavoidable dead ends (since the unnecessary put down action is not safe). Furthermore, since some actions allow state revisitation, EXBW is a cyclic domain (e.g., in Figure 5.1, we can repeatedly picking up and putting down $b3$). These characteristics make EXBW uniquely challenging in probabilistic planning. Importantly, each block can only detonate once. After detonation, a block can be safely moved, which

necessitates a strategic shift post-detonation. Additionally, we use a corrected version of this domain, where blocks are explicitly forbidden from being placed on themselves, refining the problem space by removing trivial and unrealistic configurations.

5.1.2 Triangle Tiresworld

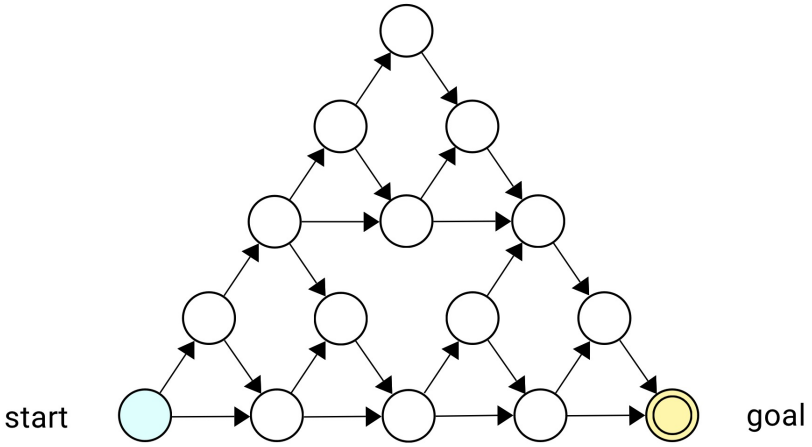

Figure 5.2: An example of a Triangle Tireworld instance with the initial state (blue node) and the goal state (yellow node). Every node along the longest path (from the blue node to the top corner to the yellow node) is equipped with a spare tire.

Our second testbed domain Triangle Tireworld (referred to hereafter as "tireworld") is introduced by Little et al. [2007] and also frequently used in IPPCs. Tireworld is a probabilistic planning domain where a car must navigate a triangular map of locations to reach a designated goal from its starting position. Each move carries a 0.5 probability of resulting in a flat tire, which immobilises the car unless it has a spare tire or one is available at its current location. Only the longest path in this domain is equipped with sufficient spare tires to guarantee a successful route to the goal with probability 1. As a result, problems in this domain include avoidable dead ends, meaning unlike in EXBW, $p^{\max} = 1$ for all tireworld instances. However, shorter paths lack adequate spare tires and pose a probabilistic risk of stranding the car.

Triangle Tireworld is also cyclic, as the car may need to revisit locations to access spare tires or avoid risky paths. Moving between locations often involves returning to previous states to maintain progress. The triangular map structure allows multiple routes between points, leading to loops. Figure 5.2 provides an illustrative example of a tireworld task where the longest path runs from the left corner to the top corner to the bottom corner.

5.1.3 Model Configuration

In this chapter, we examine two models: an MPNN model for learning within the MPNN framework (Algorithm 2) and a GPR model for learning within the GPR framework (Algorithm 4). Specifically, we select MPNN and GPR model configurations that have demonstrated optimal performance in domain-dependent learning-for-planning tasks [Chen et al., 2024]. The model details are listed below:

Model for MPNN Framework

Within the MPNN framework, we utilise a modified Relational Graph Convolutional Network (RGCN) [Schlichtkrull et al., 2018] with a mean aggregator function and 10 message-passing layers, which is selected as the best performing layer that ensuring each node in PLGL receives at least one message from the most distant node in the graph meanwhile not adding too much overhead to computation time. The update function for our RGCN extends Equation 2.14 as follows:

$$\mathbf{h}_u^{(t+1)} = \varphi\left(\mathbf{W}_0^{(t)}\mathbf{h}_u^{(t)} + \sum_{l\in\Sigma}\sum_{(u,v)\in\mathcal{N}_l(u)}\mathbf{W}_l^{(t)}\mathbf{h}_v^{(t)}\right). \tag{5.1}$$

All other hyperparameters for the framework directly follow the optimal configuration provided in GOOSE Chen et al. [2023a].

Model for GPR Framework

In the GPR framework, we employ a GPR model with a dot-product kernel, setting the overall iteration count of the WL algorithm to 10, aligning with the 10 layers used in RGCN. The remaining hyperparameters for this framework are directly inherited from the optimal settings specified in WL-GOOSE Chen et al. [2024].

Environment Setting

All experiments in this chapter were conducted on a system equipped with an AMD Ryzen 7 7745HX 3.60GHz CPU and a single NVIDIA GeForce RTX 4070 GPU with CUDA version 12.5. The models were implemented in PyTorch [Paszke et al., 2019] and interfaced with the `C++` implementation of i-dual for MCMP [Trevizan et al., 2017b] via `pybind` [Jakob, 2023].

5.2 Learning with Different Graph Representations

In this section, we conduct the first set of experiments to comprehensively compare and evaluate the performance of PLGS/PLGL against ILG in learning $p^{\max}$. We assign all graph representations to both frameworks across four tasks covering both EXBW and Tireworld domains.

5.2.1 Experimental Setup

To enable both MPNN and GPR frameworks to learn $p^{\max}$, we first use i-dual with the 0-1 $h^{\max}$ heuristic to solve problems within these domains, obtaining $p^{\max}$ values and an optimal policy $\pi \in \Pi^{MP}$. Within a 30-minute time limit for each problem instance, i-dual with 0-1 $h^{\max}$ in the Max-Prob stage successfully solved 9 EXBW and 13 Tireworld instances. For each solved instance, we use the visited states returned by i-dual with their corresponding $p^{\max}$ values as individual training data points. We denote the solved EXBW instances as `exbw_p1`–`exbw_p9` and the Tireworld instances as `tire_3`–`tire_15`. Based on the features of these datasets, we design four tasks. For each task, we conduct six sets of experiments representing the combination of ILG, PLGS, and PLGL on both the RGCN and GPR frameworks. We run RGCN for five times and take the mean for final time, accuracy.

- EXBW Task 1 investigates the generalisability of different graphs/models to unseen problems within the same domain. Here, we train models only with `exbw_p1` and test on `exbw_p2`–`exbw_p9`.
- EXBW Task 2 examines how performance is influenced by an increase in training data. Models are trained with `exbw_p1`–`exbw_p3` and tested on `exbw_p4`–`exbw_p9`.
- EXBW Task 3 explores the impact of large training datasets without adjusting any hyperparameters that could affect overfitting analysis. Models are trained with `exbw_p1`–`exbw_p5` and tested on `exbw_p6`–`exbw_p9`.
- Tire Task: due to the extensive size of training plans for each Tireworld instance, the memory limitations of our device restrict the GPR framework to a maximum of one input problem. Therefore, we train models with `tire_3` and investigate performance across `tire_3`–`tire_15`, aiming to provide insights through comparisons with other tasks.

5.2.2 Results and Analysis

Table 5.1 summarises the experimental results for all configurations, including accuracy, node count, computation time, and hit-colour ratio, while Figures 5.3, 5.5, and 5.4 provide visualisations of accuracy and node counts to enhance intuition and highlight trends across the tasks. In Table 5.1, the RGCN node counts represent the median number of graph nodes generated per problem instance, while the GPR node counts indicate the total number of nodes across all training instances, reflecting the GPR framework's iterative node aggregation approach. Time refer to the time it takes for model to make predictions. Meanwhile the hit-colour ratio is the amount of seen colour compared to the total colour during testing stage only exist for GPR frameworks.

In this subsection, we evaluate these results to assess the performance of ILG, PLGS, and PLGL across different configurations, focusing on accuracy, computational requirements, and generalisation.

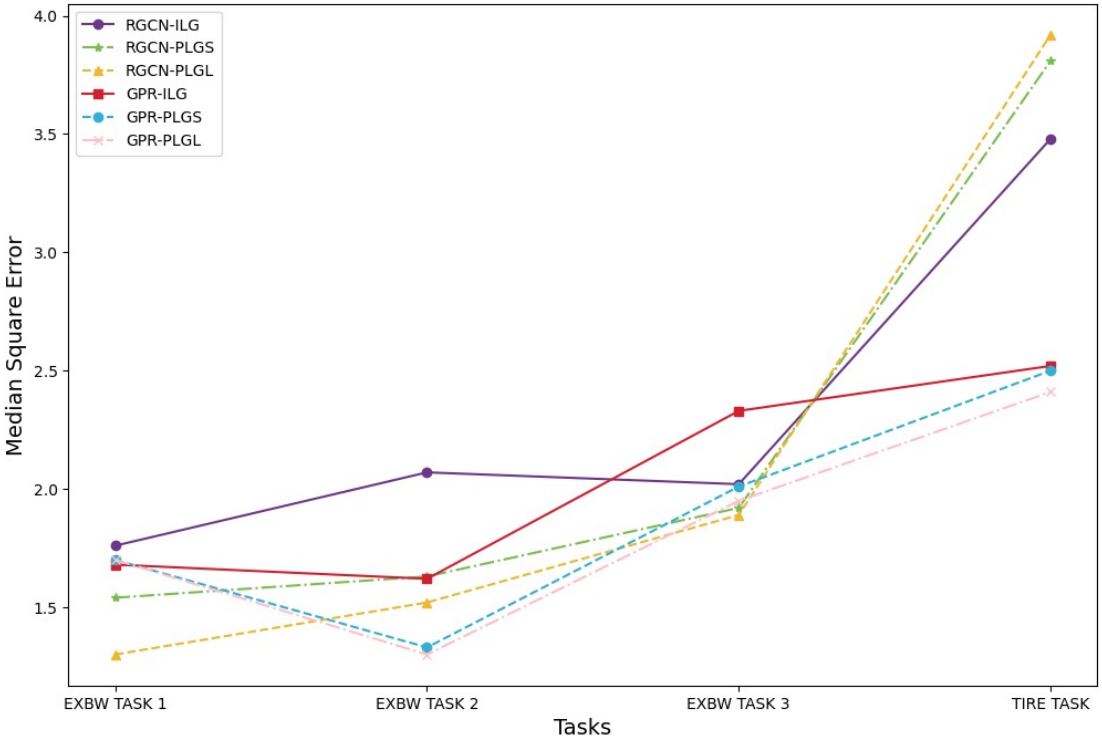

Figure 5.3: Overall test accuracy in terms of MSE (Equation 2.11). The x-axis indicates the four tasks; the y-axis represents the corresponding test accuracy. Baseline ILG graphs are shown with solid lines.

Accuracy in Terms of Median Square Error (MSE)

Accuracy, measured by Mean Square Error (MSE), serves as a fundamental metric for evaluating how well each graph representation captures $p^{\max}$ across the various tasks. As summarised in Table 5.1, PLGL consistently achieves the lowest MSE in all EXBW tasks, affirming its superior predictive accuracy over both ILG and PLGS. This advantage is particularly evident in EXBW Task 1, where PLGL achieves an MSE of 1.30 with RGCN, outperforming ILG and PLGS, which reach MSEs of 1.76 and 1.54, respectively. The design of PLGL allows it to encode complex probabilistic relationships effectively, a quality that proves beneficial as it maintains this advantage across subsequent EXBW tasks. This trend highlights PLGL's robustness and adaptability, especially in SSP domains where probabilistic dependencies are challenging to model. The persistent gap between PLGL and PLGS further underscores PLGL's enhanced capability to capture structural information essential for generalisation in probabilistic planning.

PLGS generally outperforms ILG in EXBW Tasks 1 and 2, where its added structural expressiveness enables lower MSEs. For instance, in EXBW Task 2 under the GPR framework, PLGS achieves an MSE of 1.33, surpassing ILG's 1.62. This improvement

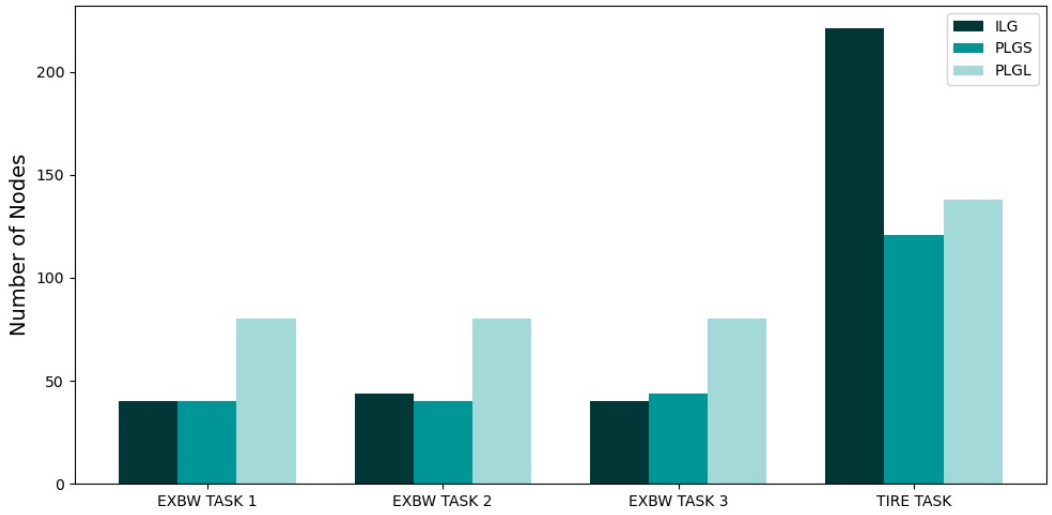

Figure 5.4: Median number of graph nodes created by RGCN across different graph representations for all training input problems. The x-axis indicates the four tasks; the y-axis indicates the median number of graph nodes created across the training set for each graph representation. Labels for graph representations are provided in the top-right corner.

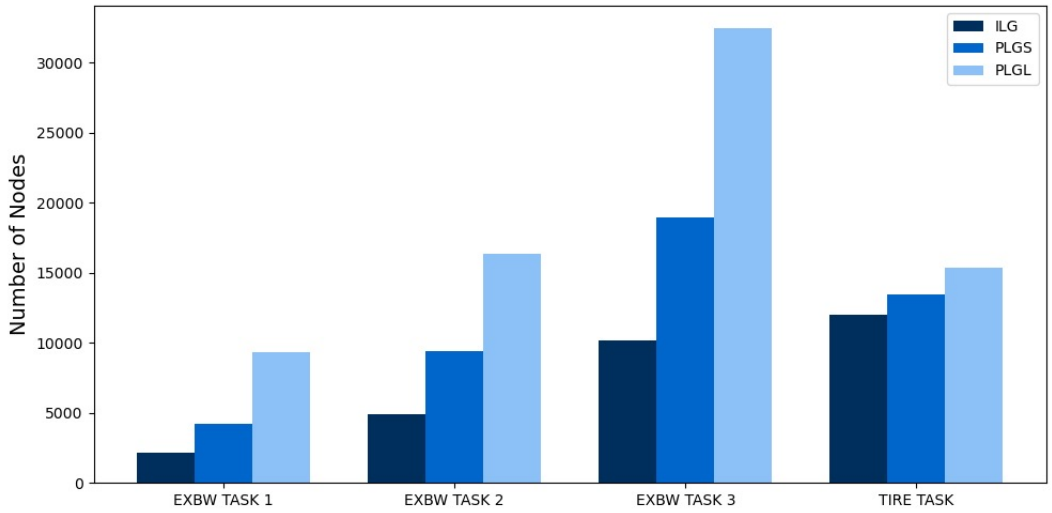

Figure 5.5: Total number of graph nodes created by the GPR framework across different graph representations for all training input problems. The x-axis indicates the four tasks; the y-axis indicates the total number of graph nodes created across the training set for each graph representation. Labels for graph representations are provided in the top-right corner.

is attributable to PLGS's ability to incorporate more domain-specific features into its graph structure, which enhances its generalisability within the EXBW domain. However, the performance gap between PLGS and PLGL suggests that while PLGS provides a better representation than ILG, it lacks the same depth of structural adaptability as PLGL, particularly in more complex scenarios.

An interesting pattern emerges in the Tireworld task, where the performance hierarchy among graph representations shifts depending on the model framework. Under the RGCN framework, ILG actually surpasses both PLGS and PLGL, achieving an MSE of 3.48 compared to PLGS's 3.81 and PLGL's 3.92. This outcome is directly influenced by the high number of objects in Tireworld relative to the schema variables, which leads ILG to generate more nodes than PLGS and PLGL, compensating for its otherwise limited structure by effectively encoding object-specific transitions. The excessive objects reduce the advantage of PLGS and PLGL in RGCN, allowing ILG to slightly outperform them in this specific setup. However, when using the GPR model in Tireworld, PLGL regains its superiority, achieving an MSE of 2.41 compared to ILG's 2.52 and PLGS's 2.50. This demonstrates that GPR's iterative learning approach mitigates the limitations observed with RGCN in PLGS and PLGL, allowing these graph representations to fully leverage their structural advantages, even in object-heavy domains like Tireworld.

Graph Node Count Analysis

The node count for each graph representation reveals key insights into structural complexity and computational overhead. In Table 5.1, the RGCN's node counts represent the median nodes generated per problem instance, while the GPR's represent the total nodes across all training instances, reflecting GPR's iterative aggregation approach. Under the RGCN framework, PLGL consistently generates a higher median node count across EXBW tasks. This increased node count reflects PLGL's richer expressiveness, which enhances its ability to model complex relationships between states and actions within probabilistic environments, thereby contributing to its improved accuracy. However, in domains like Tireworld, characterised by a higher object-to-schema ratio, ILG produces a greater node count than PLGS and PLGL under RGCN. This increase allows ILG to approximate the probabilistic structure more effectively in such object-heavy problems, accounting for its slightly better performance in Tireworld with RGCN.

In contrast, under the GPR framework, the total node count is generally higher across all representations due to the iterative node aggregation inherent in GPR. While PLGL again yields the highest total node count, the disparity among representations is less pronounced, as GPR's design tends to balance node creation. In Tireworld, for instance, PLGL generates 15,367 nodes, compared to 13,465 for PLGS and 12,030 for ILG. This convergence indicates that GPR's iterative approach minimises the node count differences seen in RGCN, allowing PLGL's structural advantages to shine. The additional nodes in PLGL under GPR directly contribute to its superior accuracy, especially in domains like Tireworld, where rich structural information can enhance probabilistic modelling.

Table 5.1: Summary of experiment results of Section 5.2 across all configurations, with best performing model highlighted for each task

	RGCN			**GPR**		
Exbw Task 1 (train 1 test 2–9)	ILG	PLGS	PLGL	ILG	PLGS	PLGL
accuracy	1.76	1.54	**1.30***	1.68	1.70	1.70
no. nodes	40	40	80	2160	4184	9312
time (s)	1.62	1.61	2.72	0.03	0.08	0.12
hit colour ratio	–	–	–	0.39	0.18	0.22
	RGCN			**GPR**		
Exbw Task 2 (train 1–3 test 4–9)	ILG	PLGS	PLGL	ILG	PLGS	PLGL
accuracy	2.07	1.63	1.52	1.62	1.33	**1.30***
no. nodes	44	40	80	4889	9399	16369
time (s)	1.23	1.42	2.82	0.04	0.22	0.33
hit colours ratio	–	–	–	0.45	0.19	0.22
	RGCN			**GPR**		
Exbw Task 3 (train 1–5 test 6–9)	ILG	PLGS	PLGL	ILG	PLGS	PLGL
accuracy	2.02	1.92	**1.89***	2.33	2.01	1.95
no. nodes	40	44	80	10203	18937	32435
time (s)	1.32	1.28	2.42	2.09	0.60	0.90
hit colour ratio	–	–	–	0.42	0.20	0.23
	RGCN			**GPR**		
Tire Task	ILG	PLGS	PLGL	ILG	PLGS	PLGL
accuracy	3.48	3.81	3.92	2.52	2.50	**2.41***
no. nodes	221	121	138	12030	13465	15367
time (s)	1.32	1.42	1.24	0.82	2.99	4.28
hit colour ratio	–	–	–	0.34	0.13	0.20

Computation Time Evaluation

The computation time results, as shown in Table 5.1, highlight the trade-offs between accuracy and computational efficiency. In the RGCN framework, ILG and PLGS generally execute faster than PLGL across all tasks. This difference is especially pronounced in EXBW Task 3 and Tireworld, where PLGL's increased complexity results in significantly longer runtimes. For example, in Tireworld, PLGL requires 4.28 seconds under GPR, whereas ILG completes the task in 0.82 seconds. The added computational cost for PLGL is attributable to its larger node count and complex structure, which demand more processing time during both training and inference.

While PLGL provides superior accuracy, its increased time requirements suggest that it may be less suitable for time-sensitive applications. ILG, with its simpler structure, performs efficiently, particularly in the RGCN framework, where it even surpasses PLGS and PLGL in accuracy for the Tireworld task. This outcome underlines the importance of selecting graph representations based on both accuracy and runtime requirements, as the most expressive model (PLGL) is not always the most efficient, especially in domains where excessive detail can hinder rather than help.

It is also noteworthy that an increase in training size can lead to additional overhead in the GPR framework, while the prediction time for RGCN remains largely unaffected by the size of the training input.

Hit-Colour Ratio Analysis

The hit-colour ratio, applicable in the GPR framework, provides additional insights into each representation's generalisation ability. This metric reflects the proportion of WL (Weisfeiler-Lehman) colours in test data that are also present in training data, indicating how well the model generalises across similar structures. PLGL achieves a relatively high hit-colour ratio across the EXBW tasks compared to PLGS, suggesting it effectively captures recurring structural patterns necessary for accurate prediction in EXBW. For example, in EXBW Task 2, PLGL achieves a hit-colour ratio of 0.22, outperforming PLGS (0.19). This outcome suggests that PLGL's richer structure facilitates generalisation within EXBW tasks, allowing it to recognise similar probabilistic configurations more consistently than PLGS.

In the Tireworld task, however, hit-colour ratios are generally lower for all representations, reflecting the domain's structural complexity and the limited overlap in node features between training and test instances. PLGL's hit-colour ratio remains competitive, albeit slightly lower than in EXBW, indicating its resilience in handling structural diversity in cyclic domains. While the absolute values decrease, PLGL's consistently higher hit-colour ratio in comparison to PLGS suggests that it maintains better generalisation, even in structurally challenging tasks like Tireworld.

5.2.3 Summary of First Experiment Set

In conclusion, PLGL stands out as the most accurate and structurally expressive representation, particularly in probabilistically complex tasks, albeit with increased computational costs. PLGS offers a balanced improvement over ILG in EXBW tasks but is more constrained in generalisability than PLGL. ILG, while structurally simpler, exhibits unexpected strengths in object-heavy tasks like Tireworld, particularly under the RGCN framework, however it achieves that accuracy at a cost of losing its efficiency. This analysis underscores the importance of selecting graph representations and frameworks aligned with domain characteristics and task-specific requirements.

5.3 Learning Max-Prob Heuristic to Guide Search

In this section, we conduct a second set of experiments by combining all developed components into a Max-Prob heuristic, as outlined in Algorithm 5. This heuristic is then used to guide i-dual to focus on more promising areas when solving the Max-Prob stage of MCMP. We evaluate the performance of our learned heuristic function against benchmark heuristics and assess its effectiveness as well as its potential for future application.

5.3.1 Experimental Setup

Given that the Tireworld domain features avoidable dead ends, resulting in $p^{\max}$ always equalling one, this section focuses exclusively on EXBW, which contains both cycles and unavoidable dead ends. Since ILG has demonstrated limited effectiveness in terms of expressiveness and generalisability, we exclude it here, assessing only the performance of heuristics learned from both RGCN and GPR models. The models were trained on three distinct training sets and applied to a single randomly selected test problem. Specifically, we defined the training sets as follows:

- **Train Set 1:** Consists of four problems not similar to the test problem.
- **Train Set 2:** Containing three problems similar to the test problem.
- **Train Set 3:** Contain just one most similar problem to the test problem.

We introduce a new clipping function, $h < X$, which returns a value of 0 for any state where the heuristic value $h(s)$ is strictly less than X. This function effectively designates such states as dead ends within the solver. We evaluate the influence of this clipping function on the performance of our heuristic h in comparison with two benchmarks: the `always-1` heuristic, which returns the upper bound of 1, and the 0-1 $h^{\max}$ heuristic, which achieves dead-end detection and demonstrates state-of-the-art performance.

All experiments use i-dual [Trevizan et al., 2017b] to solve the Max-Prob stage of SSP under MCMP, as detailed in Theorem 1. We use l/s to simplify PLGS/PLGL.

5.3.2 Results and Analysis

Table 5.2 summarise the experimental results for all models, with features interpreted as follows:

- **P*:** The solution returned by i-dual in the Max-Prob stage of MCMP. If $P* = p^{\max}$, we achieve the correct optimal solution. Otherwise, if $P* = p^{\tau} < p^{\max}$, indicating a suboptimal probability, the result is highlighted in blue.
- **No. states:** The total number of states expanded during the search process.
- **No. iterations:** The number of solver iterations, directly reflecting the computational effort.

- $h < X$**:** The clipping threshold applied to the Max-Prob heuristic.
- **h time:** The computation time for generating the heuristic value.
- **Solver time:** The total time taken by i-dual to solve the problem using the specified heuristic.

Due to Python-related limitations, the computation time for heuristic generation (*h time*) is not directly comparable across different heuristics. Thus, we rely primarily on the number of expanded states (*No. states*) and the number of solver iterations (*No. iterations*) as our main indicators of performance, as these metrics are unaffected by Python-related delays and offer a more reliable assessment of search efficiency.

Our learned Max-Prob heuristics, derived from both WL and RGCN models, demonstrate substantial efficiency improvements over the `always-1` heuristic across all configurations. For example, This significant difference proves the basic-level ability for our heuristic to be able to focus the search on promising paths, minimising unnecessary expansions.

In certain configurations, our heuristics achieve the optimal $p^{\max}$ with comparable state expansions and iterations than $h^{\max}$. For instance, in Train Set 3, GPR-l achieves $P^* = 0.90$ (the optimal $p^{\max}$) with 901 states and 3390 iterations, approaching the efficiency of $h^{\max}$, which reaches $P^* = 0.90$ with only 875 states and 1257 iterations. This indicates that, under favourable configurations, our learned heuristics can attain optimality with efficient state expansions, approaching the performance of $h^{\max}$.

Moreover, even when our heuristics do not yield the optimal $p^{\max}$, they often provide a strong lower bound on $p^{\max}$, which is advantageous in guiding the search process effectively. For example, in Train Set 3, using GPR-l with a $h < 0.25$ clipping threshold results in $P^* = 0.73$ while expanding only 157 states and requiring 270 iterations. Although suboptimal, this solution offers a reliable lower bound on $p^{\max}$, drastically reducing computational effort. This trade-off showcases the flexibility of our heuristics to balance efficiency and solution quality, allowing users to prioritise computational savings when optimality is not strictly required. This balance between efficiency and optimality in MCMP underscores both the strengths and potential limitations of our approach, which we discuss further in the next chapter.

The choice of training set also plays a significant role in heuristic performance. Heuristics trained on more extensive and diverse sets, such as Train Set 1, tend to generalise better, achieving near-optimal $p^{\max}$ values with fewer states and iterations. Conversely, heuristics trained on smaller sets, such as Train Set 3, while computationally efficient, may return a lower P^* due to limited exposure to diverse state configurations. This observation highlights the importance of training set diversity in enhancing the robustness and overall efficacy of heuristics in structured probabilistic domains.

Table 5.2: Summary of experimental results for heuristic performance in Section 5.3

	Data				Time (s)	
Domain-independent	P*	No. states	No. iterations	$h < X$	h time	Solver time
$h^{\max}$	0.90	875	1257	none	0.01	0.05
always-1	0.90	25461	71884	none	-	1.65
	Data				**Time (s)**	
Train Set 1	P*	No. states	No. iterations	$h < X$	h time	Solver time
GPR-s	0.90	3893	17953	none	11.41	0.51
GPR-l	0.90	1615	5872	none	6.21	0.21
GPR-l	0.44	1253	3644	0.3	5.69	0.11
GPR-l	0.73	2089	7469	0.2	9.65	0.27
GPR-l	0.73	1869	7320	0.1	8.52	0.27
RGCN-s	0.81	7581	48449	none	23.14	1.54
RGCN-l	0.60	3864	20441	none	11.64	0.53
	Data				**Time (s)**	
Train Set 2	P*	No. states	No. iterations	$h < X$	h time	Solver time
GPR-s	0.90	2065	8712	none	8.19	0.28
GPR-l	0.90	1523	7065	none	5.12	0.26
GPR-l*	0.44	652	675	0.3	3.91	0.02
GPR-l	0.73	1003	4319	0.2	4.50	0.13
GPR-l*	0.73	874	3542	0.1	3.65	0.11
RGCN-s	0.81	2567	18231	none	9.15	0.52
RGCN-l	0.81	2130	15714	none	8.21	0.49
	Data				**Time (s)**	
Train Set 3	P*	No. states	No. iterations	$h < X$	h time	Solver time
GPR-s	0.90	1031	3709	none	1.25	0.12
GPR-l	0.90	901	3390	none	2.25	0.11
GPR-l*	0.00	257	478	0.3	0.47	0.02
GPR-l*	0.73	157	270	0.25	0.29	0.01
GPR-l*	0.73	230	478	0.2	0.42	0.01
GPR-l*	0.83	550	1511	0.1	1.00	0.04
RGCN-s	0.81	1711	10397	none	4.61	0.30
RGCN-l	0.81	2552	18447	none	7.46	0.53

5.3.3 Summary of Second Experiment Set

Our analysis shows that Max-Prob heuristics from WL and RGCN models achieve considerable efficiency gains over the `always-1` heuristic by minimising state expansions and iterations, effectively guiding the search away from less promising paths. In some cases, these heuristics perform comparably to or even exceed the efficiency of $h^{\max}$, particularly when incorporating with heuristic clipping. While the inadmissibility of these heuristics can occasionally lead to suboptimal solutions, they often provide a robust lower bound on $p^{\max}$, balancing efficiency and solution quality. This trade-off, advantageous in many scenarios, suggests avenues for further optimisation to enhance efficiency without sacrificing solution quality. Additionally, the computational overhead from Python implementation reinforces the importance of state and iteration counts as primary performance metrics, ensuring that our heuristics deliver robust guidance in complex probabilistic environments. We will explore potential enhancements to address these limitations as part of our future work in the next chapter.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

This research presents a novel approach to addressing the Max-Prob criterion in Stochastic Shortest Path Problems (SSPs) with unavoidable dead ends by introducing probabilistically enriched graph representations and machine learning-driven heuristics that offer enhanced search guidance. Motivated by the limitations of existing Max-Prob heuristics, which often lack probabilistic nuance and rely on simplifications like determinisation; meanwhile inspired by the effectiveness of learning-based approaches in deterministic domains, we developed two novel graph representations: Probabilistic Learning Graph Small (PLGS) and Probabilistic Learning Graph Large (PLGL). Both representations encode critical probabilistic dependencies directly within their structure, eliminating the need for determinisation and preserving essential uncertainty-related information. Through these enriched representations, our extended framework leverages Graph Neural Networks (GNNs) and Statistical Machine Learning (SML) models to accurately predict $p^{\max}$, allowing us to construct heuristics that strengthen the guidance provided during the Max-Prob stage of i-dual—the foremost algorithm for tackling SSPs under the robust Minimising Cost given Maximum Probability (MCMP) criterion.

Our empirical evaluation provides strong evidence that PLGS and PLGL yield substantial gains over traditional deterministic graph representations, consistently achieving lower prediction error in estimating $p^{\max}$, which ultimately enhances i-dual's search efficiency across challenging, probabilistic planning domains. By accurately encoding the probabilistic dependencies in SSPs, our learned heuristics from both GNN and SML models deliver high-quality estimates that not only match but can occasionally exceed the performance of existing best-performing heuristics such as $h^{\max}$. This competitive performance is especially noteworthy given that our heuristics are generated directly from graph-based learning, allowing for seamless integration into heuristic-guided search frameworks. Moreover, our learned heuristics provide a strong lower bound on $p^{\max}$ in

cases where exact optimality is not reached, enabling an effective balance between computational efficiency and solution quality. This trade-off underscores the practical utility of our approach, offering improved efficiency and predictive accuracy while maintaining the flexibility to adapt across a range of SSP domains.

6.2 Future Work

In the last section, we highlight potential extensions and unresolved challenges that remain associated with our research.

Optimising Engineering Performance

As discussed in Section 5.3, components of our framework that contribute to heuristic generation are implemented in Python and subsequently transferred to `C++` within i-dual using the `pybind` package. However, this setup prevents a direct comparison of heuristic generation time as part of the overall performance metric, as native `C++` implementations, such as $h^{\max}$, run considerably faster than Python code interfaced through `pybind`. To address this disparity, a promising upgrade would involve implementing our entire framework in `C++`, enabling us to factor heuristic generation times into performance comparisons with $h^{\max}$.

Adapting i-dual for Inadmissible Max-Prob Heuristics

The heuristic produced by our model for $p^{\max}$ is inadmissible, leading to i-dual's Max-Prob stage occasionally yielding p^{τ}, a lower bound for $p^{\max}$, as observed in Section 5.3. Consequently, the Min-Cost stage may not produce the optimal solution for SSPs under the MCMP criterion. This limitation arises because, in the Min-Cost stage, i-dual solves the linear program LP 1 using the probability requirement p^{τ} in constraint (C6). If p^{τ} is indeed a lower bound rather than equal to $p^{\max}$, the search halts once enough flow has reached the goal state at $p^{\tau} < p^{\max}$, potentially leaving remaining flow in non-goal state sinks. Since transferring this remaining flow incurs additional cost, i-dual will return the solution at goal probability p^{τ}, rather than pushing remaining flow to the goal state sinks. Improving i-dual's Min-Cost stage to continue exploring when non-goal state sinks retain residual flow could enable us to find optimal MCMP solutions even if the Max-Prob stage produces a lower bound. When p^{τ} is a reliable lower bound, solving the Min-Cost stage would require minimal additional computational effort while preserving the efficiency advantage of the Max-Prob stage with our learned heuristic.

Learning Metrics Beyond $p^{\max}$

Our framework is currently tailored for learning $p^{\max}$ in SSPs, employing trained model predictions as a Max-Prob heuristic. However, the framework's design can be adapted to train models that predict any SSP feature, providing a range of heuristic options for search guidance. This adaptation involves substituting the $p^{\max}$ label in the training

data with another SSP feature. For example, we could learn a heuristic for V^* and apply it to the Min-Cost stage of i-dual or other heuristic search algorithms under the finite-penalty criterion. Furthermore, the framework could be extended to capture metrics like search effort, which assesses the difficulty of reaching the goal from a given state and helps direct search algorithms towards more computationally efficient paths [Ferber et al., 2022].

Extending Heuristics to Constrained and MOPLTL SSPs

A Constrained Stochastic Shortest Path problem (C-SSP) is an SSP with $k + 1$ cost functions, where the objective is to optimise the primary cost function while respecting upper bounds on the remaining k cost functions [Altman, 1999]. Heuristics for C-SSPs estimate the likelihood of reaching the goal while adhering to these constraints, though designing effective heuristics is challenging due to probabilistic transitions and high-dimensional state spaces, which limit the applicability of standard deterministic heuristic methods [Geißer et al., 2020]. Extending our framework to support C-SSPs could involve integrating model predictions with a search algorithm that dynamically compensates for potential model inaccuracies, allowing the algorithm to adjust predictions in real-time and improve heuristic reliability in constraint-driven environments.

A language capable of encoding these constraints is Probabilistic Linear Temporal Logic (PLTL), which applies temporal constraints to SSPs, known as multi-objective PLTL SSPs (MOPLTL-SSPs) Baumgartner et al. [2018]. MOPLTL-SSPs require precise tracking of probabilities across state-action sequences, significantly increasing computational demand. Describing PLTL constraints often leads to exponential state-space expansion, with a complexity of $O(2^{2^n})$, where n denotes the size of the largest LTL formula. MOPLTL-SSP heuristics estimate the feasibility of meeting PLTL constraints while reaching the goal. Specifically, they approximate the likelihood that an action or state sequence will satisfy PLTL-defined temporal and probabilistic conditions. Such heuristics guide search algorithms towards paths that are both cost-effective and more likely to satisfy specified constraints, including safety, goal reachability, and sequential dependencies. MOPLTL-SSP's extreme complexity limits the efficacy of existing heuristics, which often struggle with computational constraints [Mallett et al., 2021]. Extending our framework to encode MOPLTL-SSPs as graphs containing both SSPs and PLTL constraints could allow our model to learn MOPLTL-SSP heuristics from these encodings, balancing MOPLTL-SSP's high complexity with the enhanced evaluation speed offered by ML-based frameworks.

Extending Heuristics to Multi-Objective Planning

Multi-Objective Planning (MOP) involves optimising multiple objectives—such as cost, time, and risk—under probabilistic conditions, where improvements in one objective may necessitate trade-offs with others [Geißer et al., 2022]. SSPs with unavoidable outcomes can be framed as Bi-Objective Planning problems with conflicting goals of maximising safety and minimising cost. In MCMP, we address these conflicts by prioritising safety

maximisation before cost minimisation, using separate heuristic functions for Max-Prob and Min-Cost that guide the search independently at each stage, thereby avoiding the need to consider inter-objective interactions. However, developing vector-valued heuristics directly representing all objectives in MOP introduces additional challenges, as these heuristics require accounting for complex interdependencies between objectives and remain computationally intensive—an area where existing heuristics often fall short [Chen et al., 2023b]. Extending our framework to support learning heuristics for MOP may involve exploring new graph encodings and model architectures that improve accuracy and efficiency, yielding heuristics better suited for complex MOP tasks. The primary challenge in MOP heuristic learning lies in that the heuristics must function as mappings from a state to a dynamic set of vectors, where the number of vectors varies across states and is unknown a priori. This variability presents a unique challenge to all current learning-for-planning methods, as they traditionally predict fixed-size outputs, such as a single heuristic value or an individual action for policies.

Bibliography

Adler, R. J., 2010. *The geometry of random fields*. SIAM. [Cited on page 29.]

Altman, E., 1999. *Constrained Markov decision processes*. Routledge. [Cited on pages 21, 24, and 82.]

Arfaee, S. J.; Zilles, S.; and Holte, R. C., 2011. Learning heuristic functions for large state spaces. *Artificial Intelligence*, 175, 16-17 (2011), 2075–2098. [Cited on page 39.]

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 39.]

Baumgartner, P.; Thiébaux, S.; and Trevizan, F., 2018. Heuristic search planning with multi-objective probabilistic ltl constraints. In *Sixteenth international conference on principles of knowledge representation and reasoning*. [Cited on page 82.]

Bertsekas, D. P. and Tsitsiklis, J. N., 1991. An analysis of stochastic shortest path problems. *Mathematics of Operations Research*, 16, 3 (1991), 580–595. [Cited on pages 1 and 7.]

Bertsimas, D. and Tsitsiklis, J. N., 1997. *Introduction to linear optimization*, vol. 6. Athena Scientific Belmont, MA. [Cited on page 21.]

Bishop, C. M. and Nasrabadi, N. M., 2006. *Pattern recognition and machine learning*, vol. 4. Springer. [Cited on pages 3, 29, and 30.]

Bonet, B., 2001. Planning as heuristic search. *Artificial Intelligence*, (2001). [Cited on pages 2, 3, 25, 26, and 39.]

Bonet, B. and Geffner, H., 2003. Labeled rtdp: Improving the convergence of real-time dynamic programming. In *ICAPS*, vol. 3, 12–21. [Cited on pages 2, 12, 25, and 36.]

Bonet, B. and Geffner, H., 2012. Action selection for mdps: Anytime ao* versus uct. In *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 26, 1749–1755. [Cited on page 2.]

Bryce, D. and Buffet, O., 2008. 6th international planning competition: Uncertainty part. *Proceedings of the 6th International Planning Competition (IPC'08)*, (2008). [Cited on page 67.]

Cai, J.-Y.; Fürer, M.; and Immerman, N., 1992. An optimal lower bound on the number of variables for graph identification. *Combinatorica*, 12, 4 (1992), 389–410. [Cited on page 33.]

Chen, D. Z.; Thiébaux, S.; and Trevizan, F., 2023a. Goose: Learning domain-independent heuristics. In *NeurIPS 2023 Workshop on Generalization in Planning*. [Cited on pages 3, 4, 32, 40, 51, 55, 56, 61, and 69.]

Chen, D. Z.; Trevizan, F.; and Thiébaux, S., 2023b. Heuristic search for multi-objective probabilistic planning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 37, 11945–11954. [Cited on page 83.]

Chen, D. Z.; Trevizan, F.; and Thiébaux, S., 2024. Return to tradition: Learning reliable heuristics with classical machine learning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 34, 68–76. [Cited on pages 3, 4, 33, 40, 46, 47, 55, 57, 62, 63, and 69.]

Cortes, C., 1995. Support-vector networks. *Machine Learning*, (1995). [Cited on pages 29 and 40.]

d'Epenoux, F., 1963. A probabilistic production and inventory problem. *Management Science*, 10, 1 (1963), 98–108. [Cited on pages 21 and 37.]

E-Martín, Y.; R-Moreno, M. D.; and Smith, D. E., 2014. Progressive heuristic search for probabilistic planning based on interaction estimates. *Expert Systems*, 31, 5 (2014), 421–436. [Cited on page 37.]

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 *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 32, 583–587. [Cited on page 82.]

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 39.]

Garg, S.; Bajpai, A.; et al., 2020. Symbolic network: generalized neural policies for relational mdps. In *International Conference on Machine Learning*, 3397–3407. PMLR. [Cited on page 39.]

Geffner, H., 2018. Model-free, model-based, and general intelligence. *arXiv preprint arXiv:1806.02308*, (2018). [Cited on pages 1, 3, and 39.]

Geffner, P. H. and Haslum, P., 2000. Admissible heuristics for optimal planning. In *Proceedings of the 5th Internat. Conf. of AI Planning Systems (AIPS 2000)*, 140–149. [Cited on page 26.]

Geisser, F.; Haslum, P.; Thiébaux, S.; and Trevizan, F., 2022. Admissible heuristics for multi-objective planning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 32, 100–109. [Cited on page 82.]

Geisser, F.; Povéda, G.; Trevizan, F.; Bondouy, M.; Teichteil-Königsbuch, F.; and Thiébaux, S., 2020. Optimal and heuristic approaches for constrained flight planning under weather uncertainty. In *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 30, 384–393. [Cited on page 82.]

Ghallab, M.; Nau, D.; and Traverso, P., 2004. *Automated Planning: theory and practice*. Elsevier. [Cited on pages 1, 6, and 13.]

Gilmer, J.; Schoenholz, S. S.; Riley, P. F.; Vinyals, O.; and Dahl, G. E., 2017. Neural message passing for quantum chemistry. In *International conference on machine learning*, 1263–1272. PMLR. [Cited on page 31.]

Goodfellow, I., 2016. Deep learning. [Cited on page 3.]

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 39.]

Gross, J. L.; Yellen, J.; and Anderson, M., 2018. *Graph theory and its applications*. Chapman and Hall/CRC. [Cited on pages 25, 30, 31, and 32.]

Hansen, E. A. and Zilberstein, S., 2001. Lao: A heuristic search algorithm that finds solutions with loops. *Artificial Intelligence*, 129, 1-2 (2001), 35–62. [Cited on pages 2, 12, 25, and 36.]

Hart, P. E.; Nilsson, N. J.; and Raphael, B., 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4, 2 (1968), 100–107. [Cited on page 26.]

Haslum, P.; Lipovetzky, N.; Magazzeni, D.; Muise, C.; Brachman, R.; Rossi, F.; and Stone, P., 2019. *An introduction to the planning domain definition language*, vol. 13. Springer. [Cited on pages 14 and 15.]

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 page 26.]

Howard, R. A., 1960. Dynamic programming and markov processes. *MIT Press google schola*, 2 (1960), 39–47. [Cited on pages 12, 36, and 37.]

Jakob, W., 2023. Rhinelander j., and moldovan d. 2017. pybind11—seamless operability between c++ 11 and python. [Cited on page 69.]

Keyder, E. and Geffner, H., 2008. The hmdpp planner for planning with probabilities. *Sixth International Planning Competition at ICAPS*, 8 (2008). [Cited on page 37.]

Kipf, T. N. and Welling, M., 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, (2016). [Cited on page 40.]

Klauck, M.; Steinmetz, M.; Hoffmann, J.; and Hermanns, H., 2020. Bridging the gap between probabilistic model checking and probabilistic planning: Survey, compilations, and empirical comparison. *Journal of Artificial Intelligence Research*, 68 (2020), 247–310. [Cited on page 38.]

Klössner, T.; Hoffmann, J.; Steinmetz, M.; and Torralba, A., 2021. Pattern databases for goal-probability maximization in probabilistic planning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 31, 201–209. [Cited on pages 3, 37, and 38.]

Kolobov, A.; Mausam, M.; Weld, D.; and Geffner, H., 2011. Heuristic search for generalized stochastic shortest path mdps. In *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 21, 130–137. [Cited on pages 2, 12, 17, and 37.]

Kolobov, A.; Weld, D.; et al., 2012a. A theory of goal-oriented mdps with dead ends. *arXiv preprint arXiv:1210.4875*, (2012). [Cited on pages 2 and 19.]

Kolobov, A. et al., 2012b. *Planning with Markov decision processes: An AI perspective*, vol. 17. Morgan & Claypool Publishers. [Cited on pages 2 and 16.]

Krizhevsky, A.; Sutskever, I.; and Hinton, G. E., 2012. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25 (2012). [Cited on page 3.]

Little, I.; Thiebaux, S.; et al., 2007. Probabilistic planning vs. replanning. In *ICAPS Workshop on IPC: Past, Present and Future*, 1–10. [Cited on page 68.]

Mallett, I.; Thiébaux, S.; and Trevizan, F., 2021. Progression heuristics for planning with probabilistic ltl constraints. In *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, 11870–11879. [Cited on page 82.]

OpenAI, 2023. Gpt models by openai. https://openai.com. Accessed: 2023-10-16. [Cited on page 3.]

Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; et al., 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32 (2019). [Cited on page 69.]

Patel, A. J., 2014. Introduction to the a* algorithm. (2014). Https://www.redblobgames.com/pathfinding/a-star/introduction.html. [Cited on page 27.]

Pearl, J., 1984. *Heuristics: intelligent search strategies for computer problem solving*. Addison-Wesley Longman Publishing Co., Inc. [Cited on pages 21 and 26.]

Puterman, M. L., 2014. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons. [Cited on pages 1, 11, and 36.]

Rintanen, J., 2003. Expressive equivalence of formalisms for planning with sensing. In *ICAPS*, 185–194. [Cited on page 43.]

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 page 32.]

Russell, S. J. and Norvig, P., 2016. *Artificial intelligence: a modern approach*. Pearson. [Cited on pages 1, 6, 7, 12, 13, 25, 26, and 27.]

Salhi, S. and Thompson, J., 2022. The new era of hybridisation and learning in heuristic search design. In *The Palgrave Handbook of Operations Research*, 501–538. Springer. [Cited on page 39.]

Samadi, M.; Felner, A.; and Schaeffer, J., 2008. Learning from multiple heuristics. In *AAAI*, 357–362. [Cited on page 39.]

Sanner, S. et al., 2010. Relational dynamic influence diagram language (rddl): Language description. *Unpublished ms. Australian National University*, 32 (2010), 27. [Cited on page 39.]

Schlichtkrull, M.; Kipf, T. N.; Bloem, P.; Van Den Berg, R.; Titov, I.; and Welling, M., 2018. Modeling relational data with graph convolutional networks. In *The semantic web: 15th international conference, ESWC 2018, Heraklion, Crete, Greece, June 3–7, 2018, proceedings 15*, 593–607. Springer. [Cited on pages 40 and 69.]

Sharma, V.; Arora, D.; Geisser, F.; Singla, P.; et al., 2022. Symnet 2.0: Effectively handling non-fluents and actions in generalized neural policies for rddl relational mdps. In *Uncertainty in Artificial Intelligence*, 1771–1781. PMLR. [Cited on page 39.]

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 3, 39, and 40.]

Shen, W.; Trevizan, F.; Toyer, S.; Thiébaux, S.; and Xie, L., 2019. Guiding search with generalized policies for probabilistic planning. In *Proceedings of the International Symposium on Combinatorial Search*, vol. 10, 97–105. [Cited on pages 10 and 39.]

Shervashidze, N.; Schweitzer, P.; Van Leeuwen, E. J.; Mehlhorn, K.; and Borgwardt, K. M., 2011. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12, 9 (2011). [Cited on pages 33 and 58.]

Slaney, J. and Thiébaux, S., 2001. Blocks world revisited. *Artificial Intelligence*, 125, 1-2 (2001), 119–153. [Cited on page 67.]

Smola, A. J. and Schölkopf, B., 2004. A tutorial on support vector regression. *Statistics and computing*, 14 (2004), 199–222. [Cited on page 29.]

Steinmetz, M.; Hoffmann, J.; and Buffet, O., 2016. Revisiting goal probability analysis in probabilistic planning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 26, 299–307. [Cited on page 37.]

Teichteil-Königsbuch, F., 2012. Stochastic safest and shortest path problems. In *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 26, 1825–1831. [Cited on pages 2 and 18.]

Teichteil-Königsbuch, F.; Vidal, V.; and Infantes, G., 2011. Extending classical planning heuristics to probabilistic planning with dead-ends. In *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 25, 1017–1022. [Cited on page 13.]

Theodoridis, S., 2015. *Machine learning: a Bayesian and optimization perspective*. Academic press. [Cited on page 28.]

Toyer, S.; Thiébaux, S.; Trevizan, F.; and Xie, L., 2020. Asnets: Deep learning for generalised planning. *Journal of Artificial Intelligence Research*, 68 (2020), 1–68. [Cited on pages 3 and 38.]

Toyer, S.; Trevizan, F.; Thiébaux, S.; and Xie, L., 2018. Action schema networks: Generalised policies with deep learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32. [Cited on page 3.]

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 pages 2 and 37.]

Trevizan, F.; Thiébaux, S.; Santana, P.; and Williams, B., 2016. Heuristic search in dual space for constrained stochastic shortest path problems. In *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 26, 326–334. [Cited on pages 2, 10, 11, 21, 24, 26, 27, and 37.]

Trevizan, F. W.; Teichteil-Königsbuch, F.; and Thiébaux, S., 2017b. Efficient solutions for stochastic shortest path problems with dead ends. In *UAI*. [Cited on pages 2, 11, 16, 17, 18, 19, 20, 21, 22, 24, 27, 28, 37, 38, 66, 69, and 76.]

Trevizan, F. W. and Veloso, M. M., 2014. Depth-based short-sighted stochastic shortest path problems. *Artificial Intelligence*, 216 (2014), 179–205. [Cited on pages 7, 9, and 10.]

Trudeau, R. J., 2013. *Introduction to graph theory*. Courier Corporation. [Cited on page 25.]

Vallati, M.; Chrpa, L.; Grześ, M.; McCluskey, T. L.; Roberts, M.; Sanner, S.; et al., 2015. The 2014 international planning competition: Progress and trends. *Ai Magazine*, 36, 3 (2015), 90–98. [Cited on pages 15 and 60.]

Vanderbei, R. J., 1998. Linear programming: foundations and extensions. *Journal of the Operational Research Society*, 49, 1 (1998), 94–94. [Cited on page 21.]

Vapnik, V., 2013. *The nature of statistical learning theory*. Springer science & business media. [Cited on page 3.]

Vaswani, A., 2017. Attention is all you need. *Advances in Neural Information Processing Systems*, (2017). [Cited on page 3.]

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 39.]

Weisfeiler, B. and Leman, A., 1968. The reduction of a graph to canonical form and the algebra which appears therein. *nti, Series*, 2, 9 (1968), 12–16. [Cited on pages 33 and 40.]

Williams, C. K. and Rasmussen, C. E., 2006. *Gaussian processes for machine learning*, vol. 2. MIT press Cambridge, MA. [Cited on pages 3, 29, and 40.]

Xu, K.; Hu, W.; Leskovec, J.; and Jegelka, S., 2018. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, (2018). [Cited on pages 33 and 62.]

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 39.]

Younes, H. L. and Littman, M. L., 2004. Ppddl1. 0: An extension to pddl for expressing planning domains with probabilistic effects. *Techn. Rep. CMU-CS-04-162*, 2 (2004), 99. [Cited on pages 4, 15, and 43.]

Zhou, J.; Cui, G.; Hu, S.; Zhang, Z.; Yang, C.; Liu, Z.; Wang, L.; Li, C.; and Sun, M., 2020. Graph neural networks: A review of methods and applications. *AI open*, 1 (2020), 57–81. [Cited on pages 3 and 31.]