Saarland University



Bachelor's Thesis



GNPGNorks - Generalized Neural Policies Using Graph Neural Networks

Submitted by:

Jan Kautenburger

Submitted on:

April 8, 2024

Reviewers:

Univ.-Prof. Dr. Verena Wolf

Univ.-Prof. Dr. Jörg Hoffmann

Advisors:

M.Sc. Timo P. Gros

M.Sc. Joschka Groß

Erklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Statement

I hereby confirm that I have written this thesis on my own and that I have not used any other media or materials than the ones referred to in this thesis.

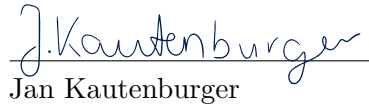
Einverständniserklärung

Ich bin damit einverstanden, dass meine (bestandene) Arbeit in beiden Versionen in die Bibliothek der Informatik aufgenommen und damit veröffentlicht wird.

Declaration of Consent

I agree to make both versions of my thesis (with a passing grade) accessible to the public by having them added to the library of the Computer Science Department.

Saarbrücken, April 8, 2024



Jan Kautenburger

Abstract

Planning is an essential aspect of artificial intelligence, especially in applications that require effective and adaptable sequential decision-making in uncertain and complex environments. However, finding problem-solving approaches that can transfer learned knowledge across instances of the same domain remains a challenge.

To address this, we introduce GNPGNorks, a framework that follows a recently published approach called *Symbolic Influence Network* (SymNet3.0) that uses *graph neural networks* (GNNs) to train generalized neural policies for solving planning problems defined in *relational dynamic influence diagram language* (RDDL). GNPGNorks represents a significant improvement in the infrastructure and enables domain-independent problem solving, setting a strong foundation for future research. However, despite intense efforts, replicating the exact results of SymNet3.0 was not possible.

Acknowledgments

I would like to express my gratitude to Prof. Dr. Verena Wolf and Prof. Dr. Jörg Hoffmann for providing me with an exciting topic for my thesis and for reviewing it. Furthermore, I thank my supervisor Timo P. Gros for his support and feedback throughout the process of conceiving and writing this thesis. My special thanks go to Joschka Groß, who supervised me during Timo P. Gros' parental leave.

Acronyms

SymNet2.0 *Symbolic Network 2.0* 11, 29, 30

SymNet3.0 *Symbolic Influence Network* iii, 1, 6, 11, 12, 18, 20, 25, 30, 32

SymNet *Symbolic Network* 11, 28–30

A3C *asynchronous advantage actor-critic* 28, 29

AI *artificial intelligence* 1

CI *continuous integration* 31

CPF *conditional probability function* 6, 8, 21, 25

DBN *dynamic Bayesian network* 1, 7, 8, 23, 24, 28

DRL *deep reinforcement learning* 1, 8

DRP *deep reactive policy* 27

GAT *graph attention network* 1, 5, 11, 18, 24, 28–30

GCN *graph convolutional network* 28

GNN *graph neural network* iii, 1, 5, 30

ICAPS *International Conference on Automated Planning and Scheduling* 6, 27

ID *influence diagram* 1, 7, 24

IDS *iterative deepening search* 9

IPC *International Planning Competition* 6, 7, 27

IPPC *International Probabilistic Planning Competition* 9, 11, 12, 14–16, 27, 31

LR *long-range* 12, 14–16

MCTS *Monte Carlo tree search* 8

MDP *Markov decision process* 1, 6, 9, 27

NLP *natural language processing* 1

PDDL *planning domain definition language* 7, 31

PE *policy evaluation* 8

POMDP *partially observable Markov decision process* 6

PPDDL *probabilistic planning domain definition language* 7

RDDL *relational dynamic influence diagram language* iii, 1, 6, 7, 9, 11, 12, 18, 25, 27, 28, 31

RL *reinforcement learning* 1, 28, 29

RMDP *relational Markov decision process* 1, 28

SDP *symbolic dynamic programming* 8

TCP/IP *transmission control protocol/internet protocol* 9

UCT *upper confidence bounds applied to trees* 9, 27

VI *value iteration* 8

XADD *extended algebraic decision diagram* 8, 21, 25, 31, 32

Contents

Acronyms vii

1. Introduction 1

2. Background 3
2.1. Sequential Decision-making Problems . . . 3
2.2. Markov Decision Processes . . . 4
2.3. Graph Neural Networks . . . 4
2.3.1. Graph Attention Networks . . . 5
2.4. Relational Dynamic Influence Diagram Language . . . 6
2.4.1. pyRDDLGym . . . 7
2.5. PROST - Probabilistic Planning Based on UCT . . . 9

3. Symbolic Influence Network 11
3.1. Overview . . . 11
3.2. Data . . . 11
3.2.1. RDDL . . . 12
3.2.2. Imitation Learning . . . 18
3.3. Architecture . . . 18

4. Results 25

5. Related Work 27

6. Conclusion and Future Work 31
6.1. Conclusion . . . 31
6.2. Future Work . . . 32

Appendices 41

A. Appendix 41
A.0.1. Hardware Specifications . . . 41

1. Introduction

Automated planning is one of the major subfields of *artificial intelligence* (AI) and describes the process of finding a goal-leading sequence of actions given a model of the underlying environment [17]. Planning is essential across a broad range of applications, including penetration testing, natural language generation, and logistics [8, 17, 30]. It is especially important for tasks that require efficient learning, often in adherence to safety constraints [3]. In addition to domain-independent planners that can solve increasingly complex tasks [10, 11], far-reaching successes have been achieved in combination with *deep reinforcement learning* (DRL) algorithms [13, 24, 27, 28].

However, it remains a challenge to find general problem-solving mechanisms in which previously learnt knowledge can be transferred [5, 9, 25, 26]. Real-world problems in particular provide for especially complex sequential decision-making problems with large factorized state spaces. In contrast to classical planning, highly stochastic exogenous and endogenous dynamics additionally complicate the planning process [9].

Markov decision processes (MDPs) have established themselves as the standard model for sequential decision-making problems [2]. For probabilistic planning and the frequent handling of large state spaces, mainly *relational Markov decision processs* (RMDPs) are used due to the need for relational abstraction [2, 14]. The planning language RDDL was influenced by the aforementioned with the general purpose to enable domain-independent planners. This is achieved by exploiting the compact representation of problems with similar properties [18].

Graph attention networks (GATs), a type of GNNs, have become increasingly popular for solving decision-theoretic planning problems. They leverage attention mechanisms to learn relationships and dependencies between nodes in a graph [5, 25, 26, 29]. Graph-structured data, as is the case in RDDL, which is semantically a *dynamic Bayesian network* (DBN) extended with an *influence diagram* (ID), is particularly suitable for this purpose [18, 26].

Nevertheless, GNNs have well-known limitations, including their limited suitability for capturing long-range dependencies [25, 26, 33]. To tackle this problem, promising hybrid approaches have emerged, such as the combination of GNNs with transformers [16]. Transformers are a neural architecture that has demonstrated remarkable success in *natural language processing* (NLP). However, this hybrid approach is largely unexplored in the field of *reinforcement learning* (RL) and automated planning in particular [12].

As we aim to pursue this promising approach in the future, this thesis establishes the necessary foundation by reconstructing and extending the work of Sharma et al. [26], who presented the domain-independent state-of-the-art planner SymNet3.0.

This work has several objectives. The first is to confirm the validity of the published results of the approach and to eliminate any major errors, such as the use of obsolete and unsupported packages. Secondly, the main goal of the implementation is to enable a modular integration of a transformer in the future and to automate as many tasks as possible. This includes in particular the automatic generation of training data with the simultaneous support of several planners, which is crucial for performance and better evaluation. In general, efforts are made to ensure that research findings can be replicated, which involves maintaining good documentation.

2. Background

This chapter aims to provide a basic understanding of the mathematical concepts involved in solving sequential decision-making problems.

2.1. Sequential Decision-making Problems

Sequential decision-making problems are encountered in various fields [30], where an agent needs to make a series of decisions over time. Each decision influences future outcomes, and the environment in which the agent operates may change over time due to its own dynamics or external influences. Similar to real-world problems, often, there is an element of uncertainty about the outcomes of actions and the evolution of the environment over time. This iterative agent-environment interaction is illustrated by Figure 2.1.

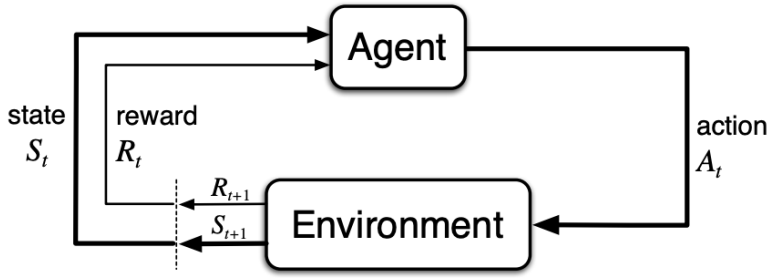


Figure 2.1.: Agent-environment interaction [30].

Understanding and solving sequential decision-making problems is crucial for designing intelligent systems that can make adaptive and informed decisions in complex and uncertain environments. The aim is to find an approach that works independently of the domain. However, since problems and associated challenges vary greatly, finding an approach that works domain-independently is generally a challenging task. The foundation for tackling this challenge is a planning formalism that includes a language, as defined in Section 2.4, for formulating all problems, and an algorithm, as covered in Chapter 3.

2.2. Markov Decision Processes

Markov decision processes (MDPs) have become the standard mathematical framework to model sequential decision-making problems [2].

Definition 2.1 (Finite Markov Decision Process [2, 15, 30])**.** *A finite Markov decision process is defined by the tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \mu \rangle$, where $\mathcal{S}$ is the finite set of states s, $\mathcal{A}$ is the finite set of actions a, $\mathcal{T}$ is the partial transition probability function, $\mathcal{R}$ is the reward function, and μ is the initial state distribution.*

For each time step t, an agent selects an applicable action $a_t \in \mathcal{A}(s_t) \subset \mathcal{A}$ based on the current state s_t. Given a state-action pair (s_t, a_t), the partial transition probability function $\mathcal{T} : \mathcal{S} \times \mathcal{A} \rightharpoonup \mathcal{D}(\mathcal{S})$ defines the probability of transitioning to the successor state s_{t+1}. The numerical reward r_{t+1}, induced by the state transition $s_t \xrightarrow[a_t]{} s_{t+1}$ is defined by the reward function $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$.

A policy defines the agent's behaviour, determining which action to take in each state. The optimal policy follows the objective of maximizing the cumulative reward.

Definition 2.2 (Deterministic Policy [30])**.** *"A deterministic (history-independent) policy π is a function $\pi : \mathcal{S} \to \mathcal{A}$ that, given a state s_t, returns an applicable action $a_t \in \mathcal{A}(s_t)$." [15]*

2.3. Graph Neural Networks

Graph neural networks (GNNs) are neural network architectures that are specifically designed to tackle problems that involve graph-structured data. They are able to capture the interdependencies between nodes and their connections by implementing the principle of *message passing* [6], which allows nodes to aggregate feature information from their neighbouring nodes and update their own feature representation accordingly.

The output node feature is given by

$$\vec{h}'_i = \sigma\left(\sum_{j \in \mathcal{N}_i} \alpha_{ij} \mathbf{W} \vec{h}_j\right), \tag{1}$$

where $\mathcal{N}_i$ is the neighbourhood of node i, α_{ij} is the weighting factor, σ is an activation function, $\mathbf{W}$ is a weight matrix, and $\vec{h}_j$ is the node feature of node j.

Figure 2.2 represents a graph convolution operation, which is the process of aggregating features from a node's neighbourhood. Although this is only shown for node b, the

aggregation takes place for each existing node. However, GNNs often require the explicit definition of a weighting factor, determining the significance of certain node features for another node, which is disadvantageous [32].

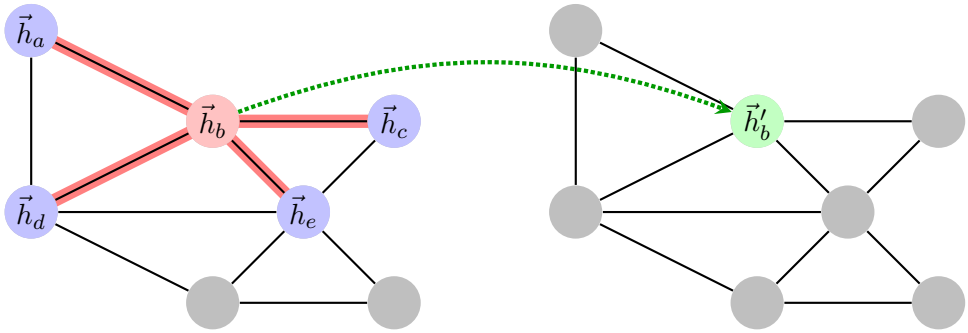


Figure 2.2.: Graph convolution operation to derive the updated node feature. [31]

2.3.1. Graph Attention Networks

Graph attention networks (GATs) are a type of GNNs that use attention mechanisms to focus on more important node features during the aggregation phase. Instead of explicitly defining a weighting factor α_{ij}, which determines the importance of node $j's$ features to node i, GATs utilize self-attention [32].

The attention coefficients are defined as

$$e_{ij} = a(\mathbf{W}\vec{h}_i, \mathbf{W}\vec{h}_j), \tag{2}$$

where a is a shared attentional mechanism in form of a single-layer feedforward neural network, $\mathbf{W}$ is a weight matrix, and $\vec{h}$ are the respective node features [32].

The weighting factor α_{ij} from Equation 1 is replaced by the normalized attention coefficients from Equation 2 as

$$\alpha_{ij} = \text{softmax}_j(e_{ij}) = \frac{\exp(e_{ij})}{\sum_{k \in \mathcal{N}_i} \exp(e_{ik})}. \tag{3}$$

To stabilize the learning process of self-attention, multiple attention heads are used. Figure 2.3 depicts the process for three attention heads. The graph convolution operation is independently performed three times and the outputs are aggregated. Each coloured line in Figure 2.3 represents an independent operation.

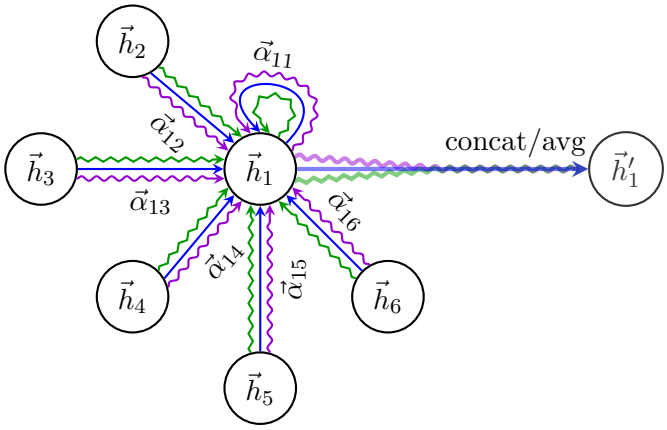


Figure 2.3.: Graph attentional layer, illustrating the multi-head self-attention mechanism of a node over its direct neighbours [32]

2.4. Relational Dynamic Influence Diagram Language

RDDL is a domain modelling language specifically designed to represent real-world stochastic planning problems, particularly MDPs, in efficient manner. It has been utilized in the *International Conference on Automated Planning and Scheduling* (ICAPS) in multiple years, including 2011, 2014, 2018, and the latest in 2023 for the Probabilistic Planning and Reinforcement Learning track of the *International Planning Competitions* (IPCs) [21].

Complex stochastic environments are described by a set of parameterized variables (pVariables), *conditional probability functions* (CPFs), reward functions, action preconditions, state invariants and a termination definition. Parameterized variables include constant values, states and action values parameterizable by one or more objects or, contrary to what the name suggests, can also be non-parameterized. Additionally, intermediate fluents can be included to express derived predicates and correlated effects, while observation fluents are relevant for *partially observable Markov decision processes* (POMDPs). However, since SymNet3.0 focuses on fully observable domains that do not require intermediate fluents, the latter two can be neglected [22].

CPFs define state transition functions that represent state-fluents based on their dependencies at the next time step with some probability distribution function [22]. The reward block defines an objective function, while action preconditions and state invariants are used as constraints to ensure that specific properties are not violated. The termination block specifies terminal states in the MDP, and when reached, the simulation will end [22]. Figure 2.4 provides a detailed overview of the dependency structure of the fluent variables in RDDL.

RDDL represents a relational factored MDP or POMDP which can be understood as a

relational DBN with an ID [18]. The purpose of RDDL is to simplify the modelling of a certain class of complex problems that are challenging to model with other planning domain languages like *probabilistic planning domain definition language* (PPDDL) and *planning domain definition language* (PDDL) [18]. These problems involve independent concurrent stochastic actions and events.

Planning problems defined in RDDL require a *domain.rddl* and an *instance.rddl* file. The domain file provides the general environment structure, as described above, while the instance file provides concrete variable assignments and initial state descriptions, resulting in domain grounding.

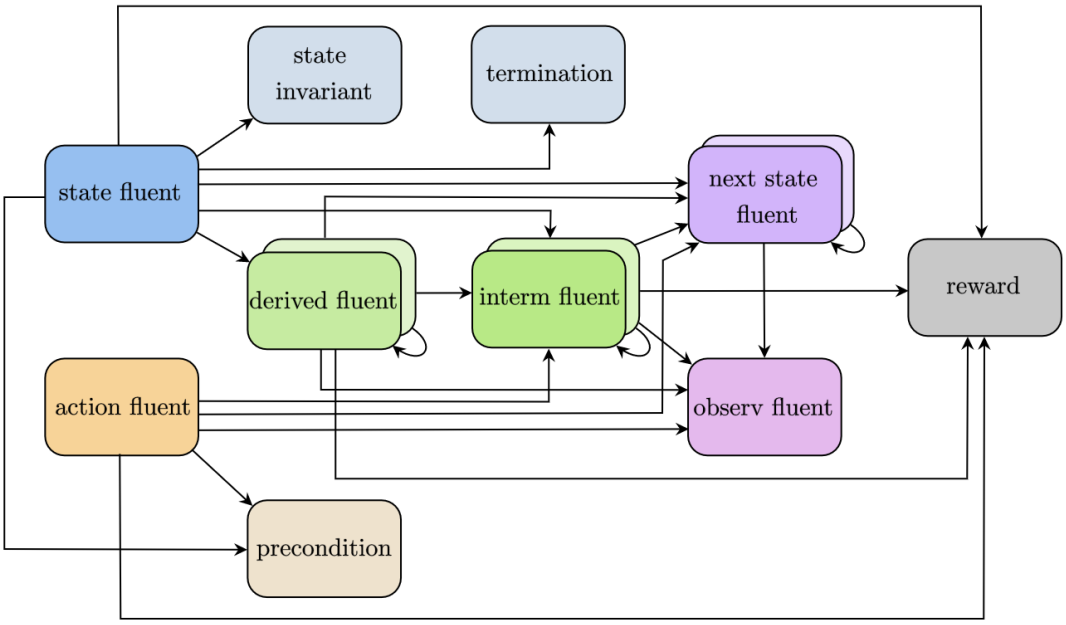


Figure 2.4.: Dependency structure of fluent variables in RDDL.

2.4.1. pyRDDLGym

The Python package *pyRDDLGym* provides a "toolkit for the auto-generation of OpenAI Gym environments from Relational Dynamic Influence Diagram Language (RDDL) description files" [23]. It is the official parser, simulator and evaluation system for RDDL since July 2022 and replaces *RDDLSim*, a Java-based predecessor. The package is under active development and integrates all new features and enhancements of the RDDL language.

In 2023, pyRDDLGym has been used the first time for the Probabilistic and Reinforcement Learning Track at the IPC. Figure 2.5 shows an overview of the most important

features of pyRDDLGym. In addition to the features already mentioned, there is a collection of domains and a symbolic toolkit. The latter enables the symbolic compilation of CPFs in *extended algebraic decision diagram* (XADD), visualizing DBNs with XADD as well as solvers for *value iteration* (VI) and *policy evaluation* (PE) using *symbolic dynamic programming* (SDP).

They also support several reference planners suitable for different problem types. The PROST planner, based on *Monte Carlo tree search* (MCTS), is suited for discrete planning problems not containing intermediate fluents as they are not supported. The JAX planner is a gradient-based planning algorithm suited for continuous problems. Even though it supports discrete state/action problems through model relaxations, the performance on discrete problems is often poorly and requires additional hyperparameter tuning as well as potential adjusting of the CPF definition in conjunction with additional constraints via the action-preconditions or state-invariants [7] [41]. The Gurobi optimizer is suited for mixed-integer linear and quadratic problems, but is "less suitable for highly stochastic problems or problems with (stochastic) dead ends" [20]. Additionally, using the Gurobi planner requires a product licence to solve problems of unlimited size. Alternatively, there is a wrapper to use DRL algorithms from the *Stable-Baselines3* or *RLib* packages.

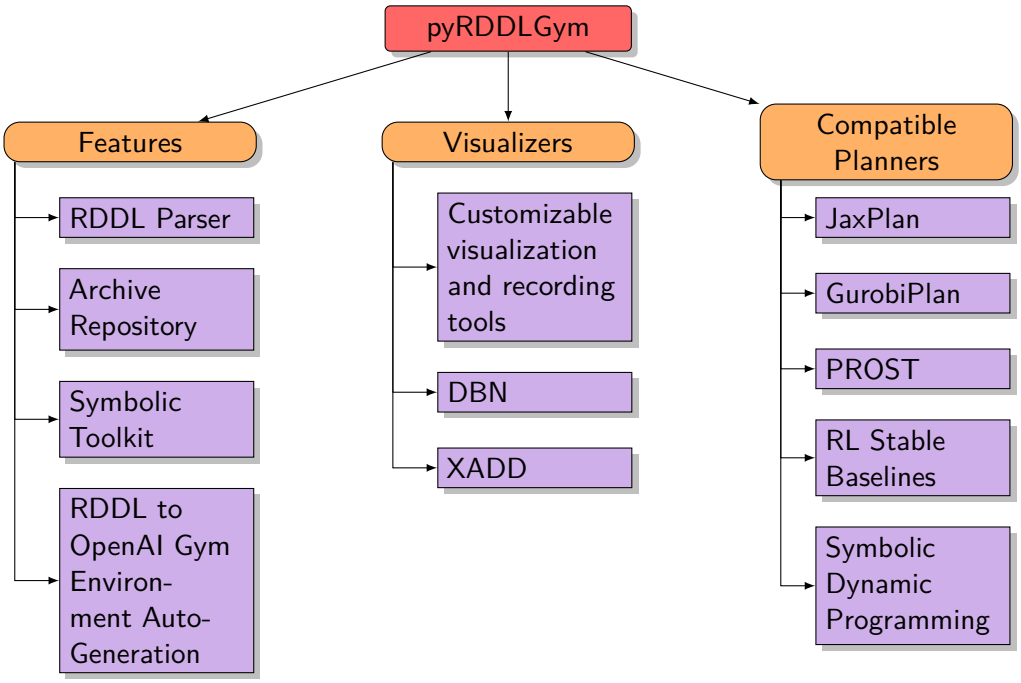


Figure 2.5.: Overview of the most important features of pyRDDLGym.

2.5. PROST - Probabilistic Planning Based on UCT

PROST is a domain-independent probabilistic planning system based on *upper confidence bounds applied to trees* (UCT). It is designed to handle probabilistically complex domains that are formulated in RDDL [10]. However, it is only capable of handling discrete domains and does not support planning problems requiring intermediate fluents.

Despite this limitation, PROST and its enhanced versions have been victorious in the discrete MDP track of the *International Probabilistic Planning Competition* (IPPC) in 2011, 2014 and 2018. The competitiveness of the UCT algorithm is based on several improvements. These include reducing the branching factor by identifying and removing superfluous actions. Additionally, the search depth is limited to aim for higher rewards in the short-term, accepting the risk of missing strategic long-term decisions [10]. When the rewards are consistent independently of the action taken, the reward is locked to avoid unnecessary computations. Also, instead of starting with random exploration, PROST uses a Q-Value initialization based on *iterative deepening search* (IDS). This allows for more promising areas of the search space to be visited, leading to faster convergence in general.

PROST is implemented in *C++* and requires the Linux operating system due to multiple library requirements. Intended to work with RDDLSim, which provides the Gym environment, PROST computes which action to take in the current state. The communication between RDDLSim and PROST, to exchange information such as the current state, the action to take, and the immediate as well as the round reward, is handled via a *transmission control protocol/internet protocol* (TCP/IP) connection.

3. Symbolic Influence Network

The latest approach for learning generalized neural policies for planning problems expressed in RDDL was presented by Sharma et al. [26]. Their architecture, called *Symbolic Influence Network* (SymNet3.0), is based on GATs which are trained by imitation learning. In this chapter, we describe their approach for solving sequential decision-making problems in detail.

3.1. Overview

SymNet3.0 is a planning system that builds on the earlier versions of *Symbolic Network* (SymNet) by Garg et al. [5] and *Symbolic Network 2.0* (SymNet2.0) by Sharma et al. [25]. It is designed for offline planning and represents instances of relational domains as *instance graph*. The architecture is based on GATs of fixed depth with parameters shared across instances to learn trajectories generated by the online planner PROST via imitation learning. The learned state representations are then decoded to generate a policy [26].

While SymNet3.0 has shown promising results on various domains, finding a suitable graph structure for the state space is still a challenge [26]. Given the source code of SymNet3.0 published on GitHub the initial goal was to build on the existing framework and integrate potential improvements. However, even though its recent publication in August 2023 the code is based on deprecated packages [34] and despite several debugging attempts, leading among other things to the discovery of missing source code [42], not executable [43]. The code lacks proper documentation and best practices are disregarded in general. Furthermore, domain specific information is hard coded limiting the application of the approach on unknown domains in general. PROST is not integrated, which requires a manual effort to generate the imitation learning dataset. Moreover, Linux as operating system is required.

Therefore, we decided to implement the general approach of SymNet3.0 and develop a framework that provides a solid basis for further research in this area.

3.2. Data

The data consists of RDDL specifications along with the corresponding PROST log files that lead to the imitation learning dataset. Two types of domains are considered: IPPC

and *long-range* (LR) domains. The IPPC domains are already known from the IPPC 2011 and 2014. The LR domains were introduced by Sharma et al. [26] and require the recognition of LR influences to compute good solutions.

Due to naming inconsistencies, the unambiguous assignment of domains was sometimes possible only by comparing the domain description and RDDL specifications. For LR domains, we expected 1000 instances for training, 100 for validation, and 200 for testing. For IPPC domains, we expected 200 for training, 10 for validation, and 40 for testing. However, for the TriangleTireworld domain, we only found 100 training instances. Additionally, during training for the Navigation and TriangleTireworld domains, SymNet3.0 accessed only 20 training instances.

Out of the data folder, 3670 files were not imported because they were either unclassifiable or irrelevant for training. We also found some additional unused domains. Except for the Corridor and Wall domains, all other domains were ignored due to missing training instances.

To ensure that there was no data leakage leading to overly optimistic performance metrics, we pairwise checked all instances of a domain for similarity, detecting a total of 71 instance duplicates. However, duplicates only occurred within the training dataset, which led to a negligible bias during training.

Figure 3.1 and 3.2 show the number of state fluents for each dataset type for each domain. This is important to ensure that the neural model can be trained on smaller instances and generalize to unseen instances of bigger size of the same domain, proving transfer learning. We observed this to be true for all domains except for SkillTeaching and Wildfire. Additionally, it is important to note that due to the often same number of states in the training and validation sets, considering the validation loss as a metric to evaluate generalizability is not robust.

3.2.1. RDDL

In the following, all used planning problems are described briefly. This step is crucial in understanding and evaluating any result. Furthermore, it allows comparing domain results under the aspects of domain similarity and evaluating results based on their reward function definition.

1. **Academic Advising (IPPC):** In this domain, a student tries to graduate by taking courses, each with a certain probability of being passed depending on the completion of its prerequisites. Taking courses incurs a cost, and additional penalties are applied for retaking courses and for each time step the student's program remains incomplete. The strategic challenge lies in selecting which courses to take each semester to efficiently progress toward graduation, considering the

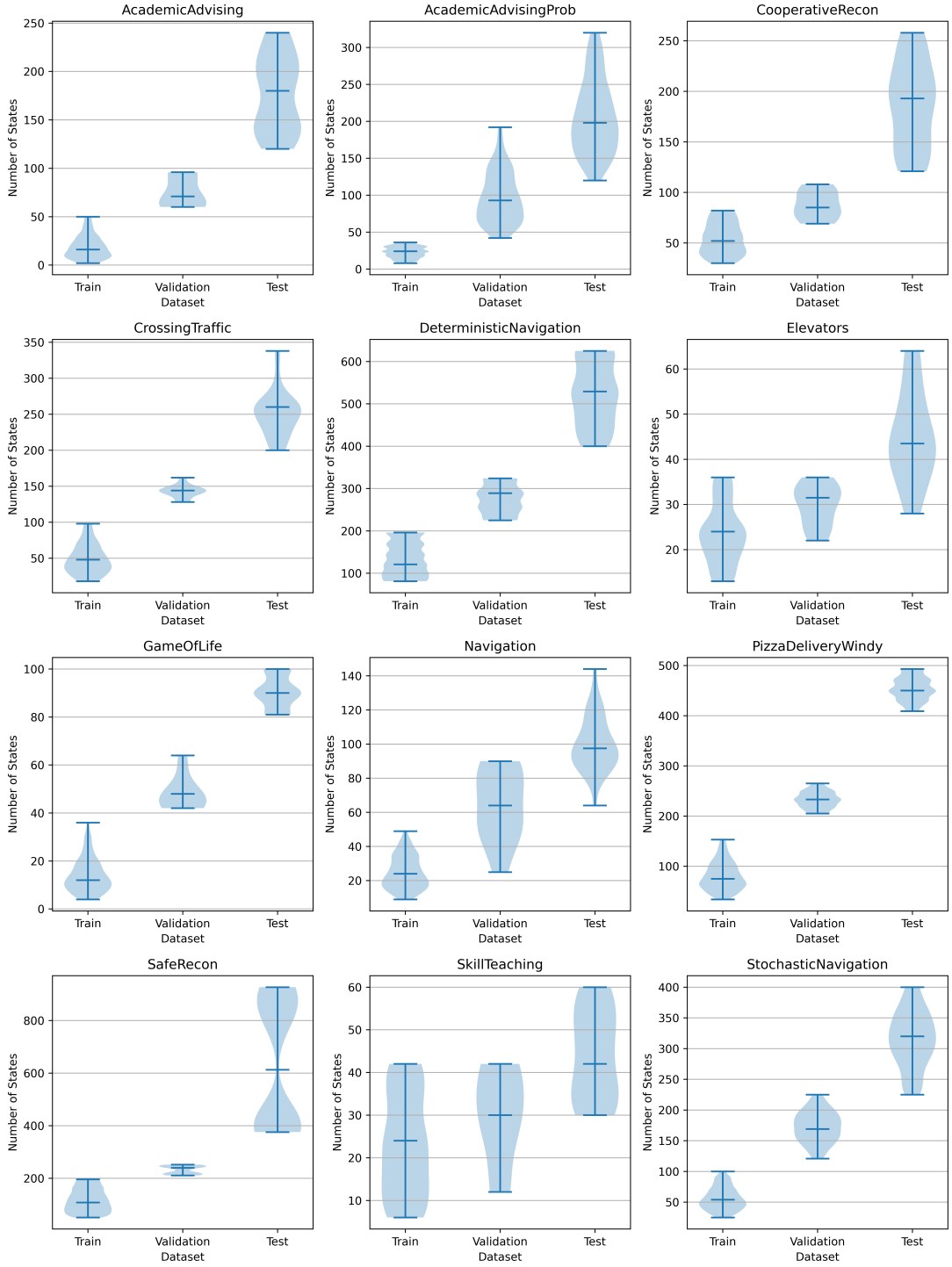


Figure 3.1.: Number of state fluents for each domain.

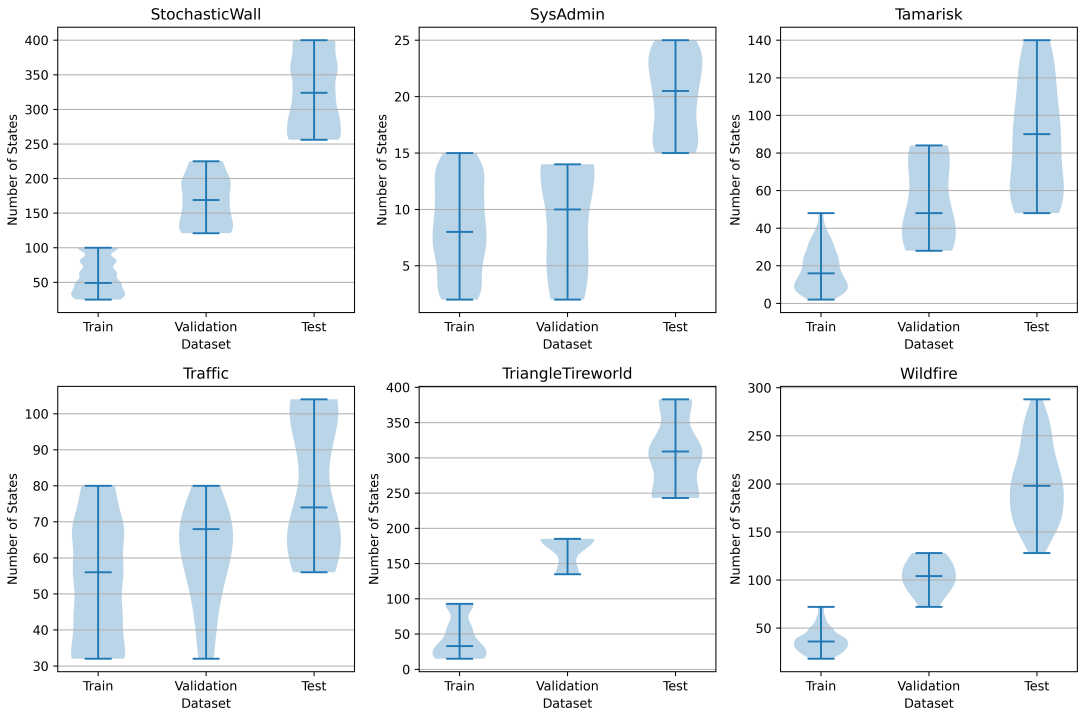


Figure 3.2.: Number of state fluents for each domain.

prerequisites, the cost of courses, and the ongoing penalty for not yet completing the program.

2. **Extreme Academic Advising (LR):** Is a variant of Academic Advising. It "has a set of courses arranged in a directed acyclic graph with some courses as program requirements that the agent has to complete in order to complete the degree. The probability of completing a course without completing all its prerequisites is very low. Therefore, in the optimal policy, a course should be taken only if it is an ancestor of some far-away program requirement." [26]

3. **Cooperative Recon (IPPC):** An agent navigates a 2D grid to use tools for detecting water, life, and taking pictures of objects while avoiding or dealing with hazardous squares that can damage the tools, impacting their effectiveness. The agent's decision-making involves selecting objects for tool use, deciding when to repair tools at the base, and strategically moving through or around hazards to minimize tool damage. Positive rewards are gained for successfully taking pictures where life is detected, and negative rewards for pictures where life is not detected, with the major planning challenges revolving around tool management, object selection, and hazard navigation.

4. **Crossing Traffic (IPPC):** A robot must navigate through a grid to reach a goal while avoiding obstacles that move left across the grid. Obstacles appear

randomly based on a Bernoulli distribution, and if an obstacle overlaps with the robot, the robot disappears, effectively ending the game.

5. **Deterministic Navigation (LR):** Similar to the Navigation domain, however, fully deterministic, i.e., the probability of the robot disappearing is zero.

6. **Elevators (IPPC):** In this domain, elevators operate within a building to transport passengers between only the top and bottom floors, with passengers arriving at floors based on a Bernoulli distribution for each floor. The domain encourages optimizing elevator operations by "hovering" near floors with a high likelihood of passenger arrivals and coordinating the movements of multiple elevators to cater to the directional preferences of the passengers efficiently.

7. **Game of Life (IPPC):** Conway's Game of Life on a grid, where the state of each cell (alive or dead) evolves based on its neighbours, with rules for under-population, overcrowding, and reproduction. The goal is to generate patterns keeping the most cells alive.

8. **Navigation (IPPC):** A robot navigates through a grid to reach a goal, contending with cells that offer varying probabilities of the robot disappearing. The challenge is to select a path that maximizes the robot's likelihood of reaching the goal within a finite time horizon, highlighting the trade-off between shorter paths with higher risk of disappearance and longer, safer paths.

9. **Pizza Delivery Windy (LR):** "A robot in a 2D-grid has to pick pizza from one of the outlets and deliver it to a customer in the shortest time in a windy (stochastic) environment. The robot should choose an outlet that minimizes the total distance, rather than going to the closest one." [26]

10. **Safe Cooperative Recon (LR):** Identical to Cooperative Recon, but with a higher number of states, as shown in Figure 3.1.

11. **Skill Teaching (IPPC):** An agent (teacher) aims to improve a student's proficiency in a group of related skills by providing hints and multiple-choice questions. In this learning environment, mastering a skill affects the student's ability to learn other skills that are prerequisites. The student's proficiency levels will increase with correct answers, decrease with incorrect ones, and randomly decline at times. The teacher must make decisions about when to introduce new skills based on the student's passed prerequisites, prioritizing skills based on their importance, and balance the use of hints and questions to maximize learning.

12. **Stochastic Navigation (LR):** Identical to Navigation domain with the only difference that the safe corridor (zero per cent probability of disappearance) does not have to be in the first column.

13. **Stochastic Wall (LR):** "A robot has to reach a goal location in a 2D grid, where the grid contains either a horizontal or a vertical wall. Each cell in the wall has a high death probability except for one randomly selected safe passage in between. An agent has to locate the safe passage in the wall and reach the goal." [26]

14. **SysAdmin (IPPC):** This domain models a network of interconnected computers that need to be managed. Each computer can be either running or not, and administrators can choose to reboot computers to ensure they are running. The goal is to maximize the overall network performance by strategically rebooting computers, taking into account the immediate penalty for rebooting and the increased likelihood of computers running due to the reboot action.

15. **Tamarisk (IPPC):** This domain covers the spread of invasive tamarisk plants in a river system, competing against native species. The goal is to minimize the overall cost by effectively managing tamarisk invasion and promoting native growth, balancing eradication and restoration efforts against the financial and ecological impacts of invasive species spread.

16. **Traffic (IPPC):** Models the dynamics of traffic flow through a network of cells, based on the Cell Transition Model (CTM), with a focus on efficiently managing traffic congestion without considering turns or turn probabilities. It features a simplified binary state model where cells are either occupied or not, and incorporates concurrent actions to handle traffic signals at intersections, allowing for the simulation of traffic light control strategies. The objective is to minimize traffic congestion by penalizing consecutive occupied cells, aiming for smooth traffic flow.

17. **Triangle Tireworld (IPPC):** Planning problem for navigating a vehicle through a road network to a goal location while managing the risk of flat tires. The objective is to reach the goal location efficiently, with rewards for success and penalties for each move made without achieving the goal, finding a balance between shorter, riskier routes and longer, safer routes.

18. **Wildfire (IPPC):** Describes a wildfire fighting scenario, where the goal is to manage and mitigate the spread of fires across a grid-based map. Fire spreading is based on the state of neighbouring cells. The goal is to maximize the reward by minimizing penalties, i.e. protecting valuable cells, and managing costs effectively.

Figure 3.3 shows the number of action fluents of each dataset type for each domain. It is important to note that a higher number of action fluents makes it more challenging for imitation learning as it decreases the probability of accidentally selecting the correct action, and vice versa. Domains that only contain non-parameterized action fluents are not included in Figure 3.3 due to their dataset type independent constant number of action fluents.

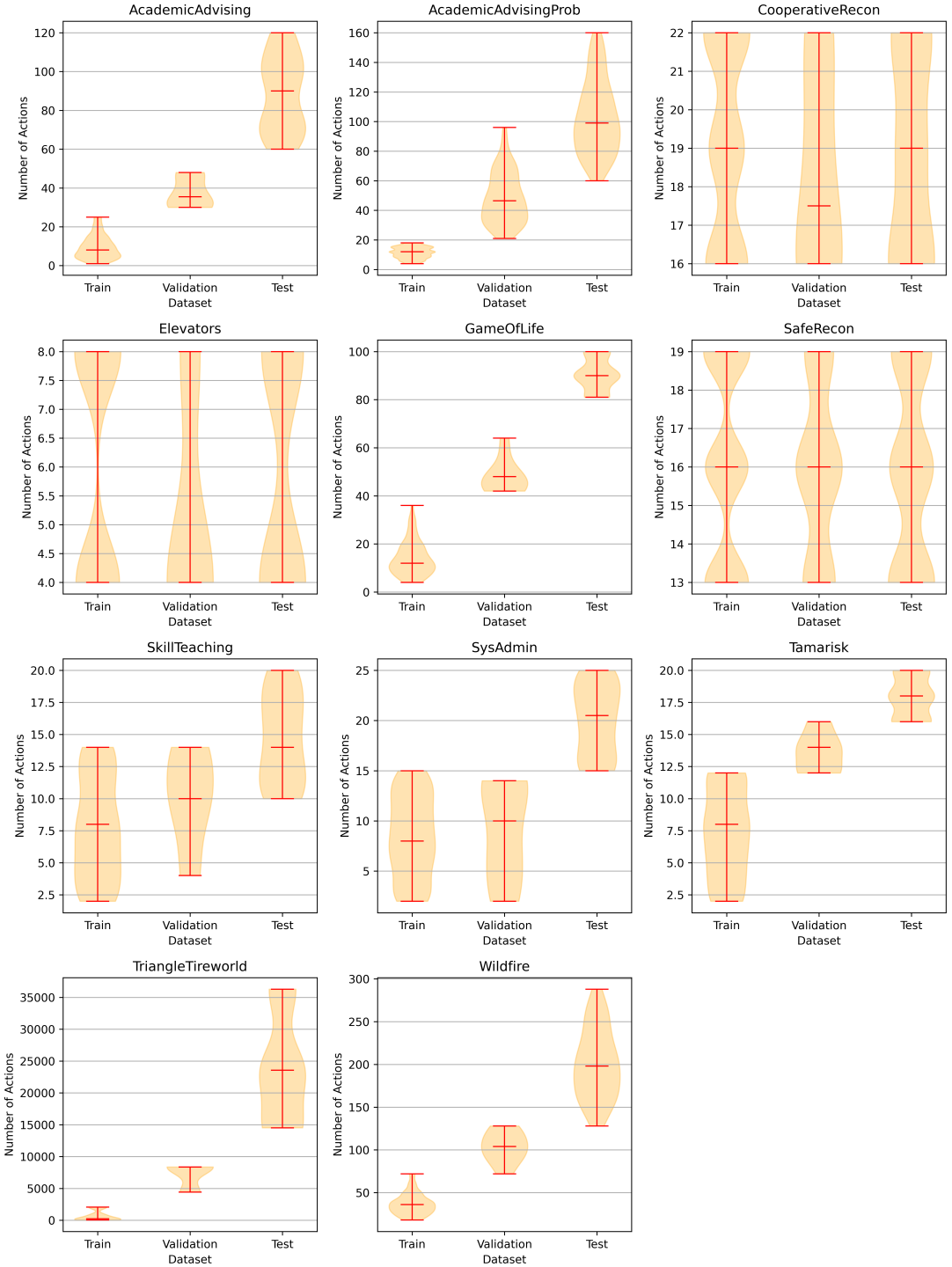


Figure 3.3.: Number of action fluents for each domain.

3.2.2. Imitation Learning

The trajectories used for imitation learning were generated by the online planner PROST. However, PROST is not integrated in SymNet3.0, which requires the manual addition of PROST results. At the start of this thesis, there was also no official PROST support using pyRDDLGym [39]. However, PROST is dependent on pyRDDLGym for providing the gym environment and exchanging information like the current state, the action to perform, and the received reward.

To address these issues, a multi-platform supporting dockerized integration of PROST and pyRDDLGym was developed. Unlike the official solution, mounted volumes were used to avoid the need for unnecessarily copying or moving files between folders, allowing the Docker container to access the correct files. PROST can be executed on single instances or several instances via a Python integration.

However, switching from the Java-based RDDLSim package to the Python-based pyRDDLGym package, some domain specifications had to be rewritten due to arithmetic errors resulting from pyRDDLGym not supporting short-circuiting for vectorized environments [35]. Additionally, there was a bug in the communication between PROST and pyRDDLGym that always returned zero as intermediate reward [36], leading to incorrect results. For instances with a high number of states, several log files were truncated [37], possibly due to peak memory consumption requirements. To address this issue, a more memory-efficient JSON parser was used [37].

Due to the number of bugs [35–37] and the need of updating the Docker image to support version two of the pyRDDLGym package, the number of trajectories had to be reduced from 30 to 10 due to the overall very high runtime of PROST, as shown by Figure 3.4 and 3.5. Identical to PROST, the most frequent action is selected for each state in order to avoid ambiguity [26]. As PROST improves its performance over time, the imitation learning dataset is not limited to the first 300 steps for each instance.

3.3. Architecture

The SymNet3.0 architecture consists of a three-step process to create a policy, as shown in Figure 3.6. Initially, it generates an *instance graph* from an RDDL instance using a GAT, where nodes aggregate information from their immediate neighbourhood through message passing. Additionally, an *influence graph* is constructed to represent state variables connected via edges based on the transition probability function. This graph calculates pairwise distances between nodes to capture long-range dependencies.

Next, an action decoder, which is a specialized neural network, translates the resulting node embeddings into action scores, which are then normalized to derive a policy.

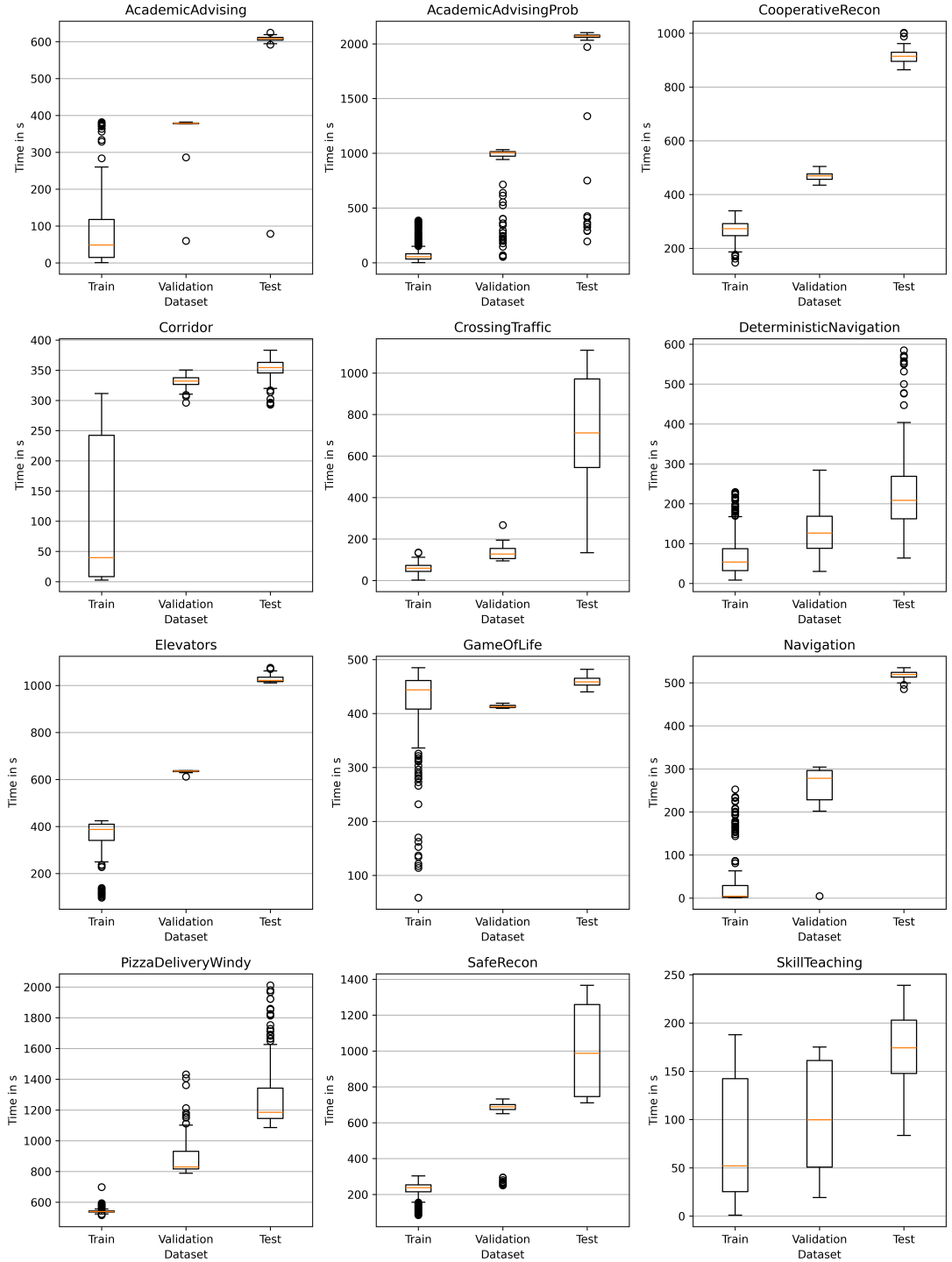


Figure 3.4.: PROST execution time for each domain.

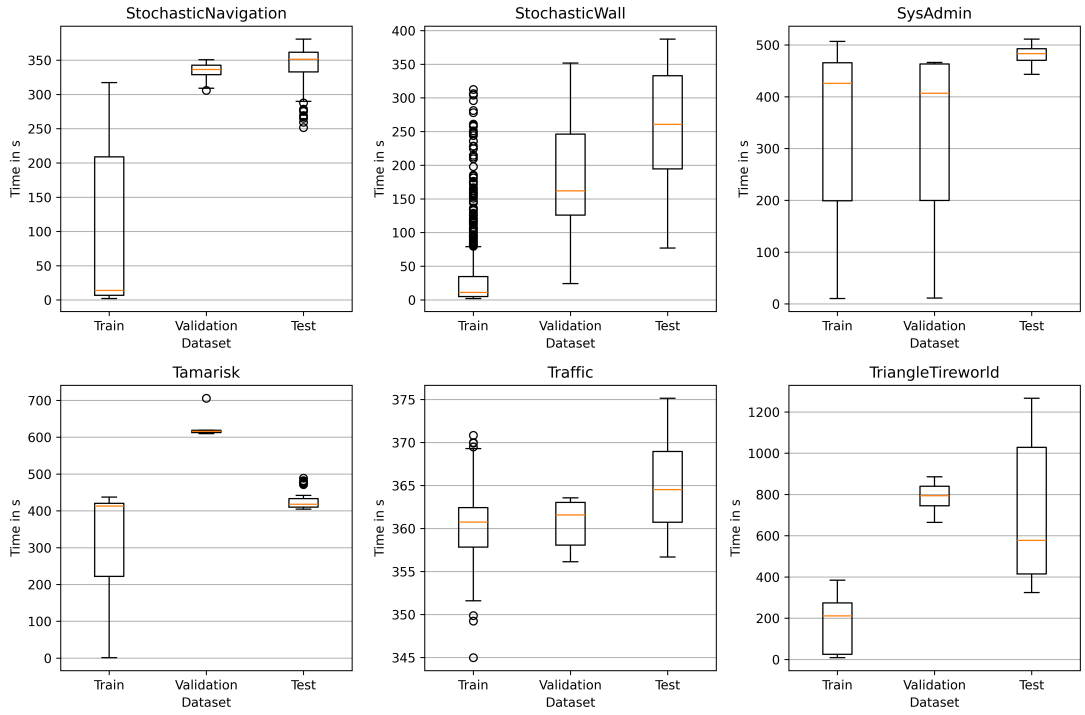


Figure 3.5.: PROST execution time for each domain.

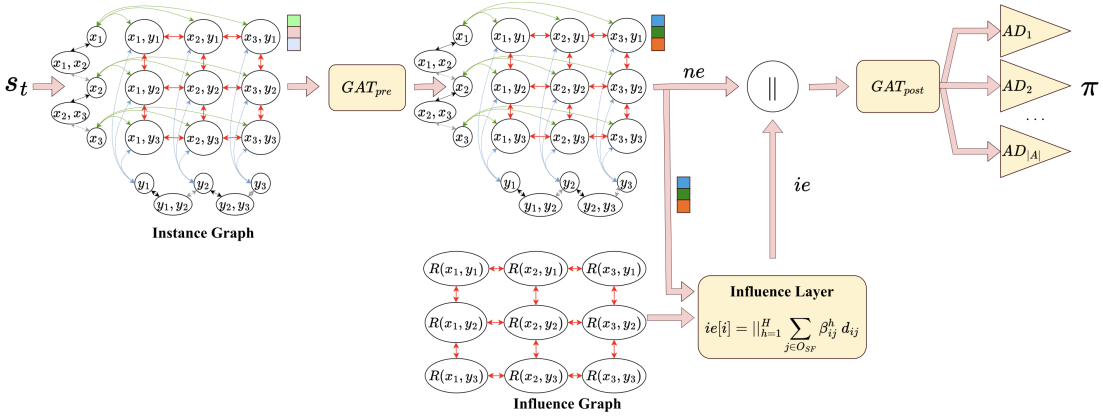


Figure 3.6.: Overview of the SymNet3.0 architecture, representing the Navigation domain, for policy prediction [26].

The following section provides a detailed description of the architecture, based on the Wildfire domain, as shown in Listing 3.1. The primary focus is on the construction of the instance graph. However, the influence graph is not included, as the XADD compilation of CPFs does not yet support state invariants [38, 40], leading to an incorrect graph representation. The latter is discussed in Chapter 4.

```
domain wildfire_mdp {

 types {
  x_pos : object;
  y_pos : object;
 };

 pvariables {
  // Action costs and penalties
  // Cost to cut-out fuel from a cell
  COST_CUTOUT : {non-fluent , real, default = -5 };
  // Cost to put-out a fire from a cell
  COST_PUTOUT : {non-fluent, real, default = -10 };
  // Penalty for each target cell that is burning
  PENALTY_TARGET_BURN : {non-fluent, real, default = -100 };
  // Penalty for each non-target cell that is burning
  PENALTY_NONTARGET_BURN : {non-fluent, real, default = -5 };

  // Topology of the cells (can be any neighborhood topology, not
      necessarily rectangular)
  NEIGHBOR(x_pos, y_pos, x_pos, y_pos) : { non-fluent, bool, default =
      false };

  // High value cells that should be protected from fire
  TARGET(x_pos, y_pos) : {non-fluent, bool, default = false };

  // State fluents
  burning(x_pos, y_pos) : { state-fluent, bool, default = false };
  out-of-fuel(x_pos, y_pos) : { state-fluent, bool, default = false };

  // Action fluents
  put-out(x_pos, y_pos) : {  action-fluent , bool, default = false };
  cut-out(x_pos, y_pos) : {  action-fluent , bool, default = false };

 };

 cpfs {

  burning'(?x, ?y) =
   if ( put-out(?x, ?y) ) // Intervention to put out fire?
      then false
      // Modification: targets can only start to burn if at least one
          neighbor is on fire
       else if (~out-of-fuel(?x, ?y) ^ ~burning(?x, ?y))
       // Ignition of a new fire? Depends on neighbors.
         then [if (TARGET(?x, ?y) ^ ~exists_{?x2: x_pos, ?y2: y_pos} (
             NEIGHBOR(?x, ?y, ?x2, ?y2) ^ burning(?x2, ?y2)))
                 then false
                    else Bernoulli( 1.0 / (1.0 + exp[4.5 - (sum_{?x2:
```

```
                        x_pos, ?y2: y_pos} (NEIGHBOR(?x, ?y, ?x2, ?y2) ^
                        burning(?x2, ?y2))]) ) ]
46    else
47     burning(?x, ?y); // State persists
48
49   // Modification: only allow non-target cells to be cut-out (cannot
        remove fuel from targets, e.g., housing)
50   out-of-fuel'(?x, ?y) = out-of-fuel(?x, ?y) | burning(?x,?y) | (~TARGET
        (?x, ?y) ^ cut-out(?x, ?y));
51
52  };
53
54  reward =
55       [sum_{?x: x_pos, ?y: y_pos} [ COST_CUTOUT*cut-out(?x, ?y) ]]
56     + [sum_{?x: x_pos, ?y: y_pos} [ COST_PUTOUT*put-out(?x, ?y) ]]
57       // Modification: if a target is out-of-fuel, it was burnt so still
            penalize (since it could not have been cut-out)
58     + [sum_{?x: x_pos, ?y: y_pos} [ PENALTY_TARGET_BURN*[ (burning(?x, ?
          y) | out-of-fuel(?x, ?y)) ^ TARGET(?x, ?y) ]]]
59     + [sum_{?x: x_pos, ?y: y_pos} [ PENALTY_NONTARGET_BURN*[ burning(?x,
          ?y) ^ ~TARGET(?x, ?y) ]]];
60 }
```

Listing 3.1: Wildfire domain description [19].

Each domain description defines object types (C), parameterized variables, such as state-fluents (SF), non-fluents (NF), and action-fluents (A), a transition probability function (T), and a reward function (R). For instance, given the Wildfire domain in Listing 3.1, we obtain

$$
\begin{aligned}
C &= \{x_pos, y_pos\} \\
SF &= \{burning(x_pos, y_pos), out\text{-}of\text{-}fuel(x_pos, y_pos)\} \\
NF &= \{\text{NEIGHBOR}(x_pos, y_pos, x_pos, y_pos), \text{TARGET}(x_pos, y_pos), \\
&\qquad \text{COST_CUTOUT}, \text{COST_PUTOUT}, \text{PENALTY_TARGET_BURN} \\
&\qquad \text{PENALTY_NONTARGET_BURN}\} \\
A &= \{put\text{-}out(x_pos, y_pos), cut\text{-}out(x_pos, y_pos)\}.
\end{aligned}
$$

The instance description provides the grounding of a domain and defines the ground objects ($\mathcal{O}$), the initial state (s_0), the discount factor (γ) and the horizon (H). For example, given the Wildfire instance in Listing 3.2, we get the following grounding

$$
\begin{aligned}
C_{\mathcal{O}} &= \{x1,\, x2,\, y1\} \\
SF_{\mathcal{O}} &= \{\mathit{burning}(x1,\, y1),\, \mathit{burning}(x2,\, y1),\, \mathit{out\text{-}of\text{-}fuel}(x1,\, y1),\, \mathit{out\text{-}of\text{-}fuel}(x2,\, y1)\} \\
NF_{\mathcal{O}} &= \{\textsc{neighbor}(x1,\, y1,\, x2,\, y1),\, \textsc{neighbor}(x2,\, y1,\, x1,\, y1), \\
&\qquad \textsc{target}(x1,\, y1),\, \textsc{target}(x2,\, y1),\, \textsc{cost_cutout},\, \textsc{cost_putout}, \\
&\qquad \textsc{penalty_target_burn},\, \textsc{penalty_nontarget_burn}\} \\
A_{\mathcal{O}} &= \{\mathit{put\text{-}out}(x1,\, y1),\, \mathit{put\text{-}out}(x2,\, y1),\, \mathit{cut\text{-}out}(x1,\, y1),\, \mathit{cut\text{-}out}(x2,\, y1)\}.
\end{aligned}
$$

Note that even though $\textsc{target}(x1,\, y1)$ is undefined in Listing 3.2, it represents a valid grounding.

```
non-fluents nf_wildfire_inst_mdp__1 {
 domain = wildfire_mdp;
 objects {
  x_pos : {x1,x2};
  y_pos : {y1};
 };
 non-fluents {
  NEIGHBOR(x1,y1,x2,y1);
  NEIGHBOR(x2,y1,x1,y1);
  TARGET(x2,y1);
 };
}

instance wildfire_inst_mdp__1 {
 domain = wildfire_mdp;
 non-fluents = nf_wildfire_inst_mdp__1;
 init-state {
  burning(x1,y1);
 };

 max-nondef-actions = 1;
 horizon  = 40;
 discount = 1.0;
}
```

Listing 3.2: Wildfire instance [19].

The instance graph contains the singleton object nodes $C_{\mathcal{O}}$ and each unique object-tuple of $SF_{\mathcal{O}} \cup NF_{\mathcal{O}}$. The edges are based on the DBN. Grounded action-fluents are represented by blue ellipses, grounded state-fluents by green rectangles, next-state state-fluents by yellow ellipses, and the reward by a red diamond, as shown in Figure 3.7.

Singleton objects are connected to object-tuples through an undirected edge if they are part of the tuple. Edges between object-tuples indicate the dependencies between state-fluents or if a state-fluent affects another state-fluent based on an action-fluent [26]. The nodes are assigned features that correspond to the state-fluents and non-fluents

relevant to them, as well as a one-hot encoded vector denoting their type.

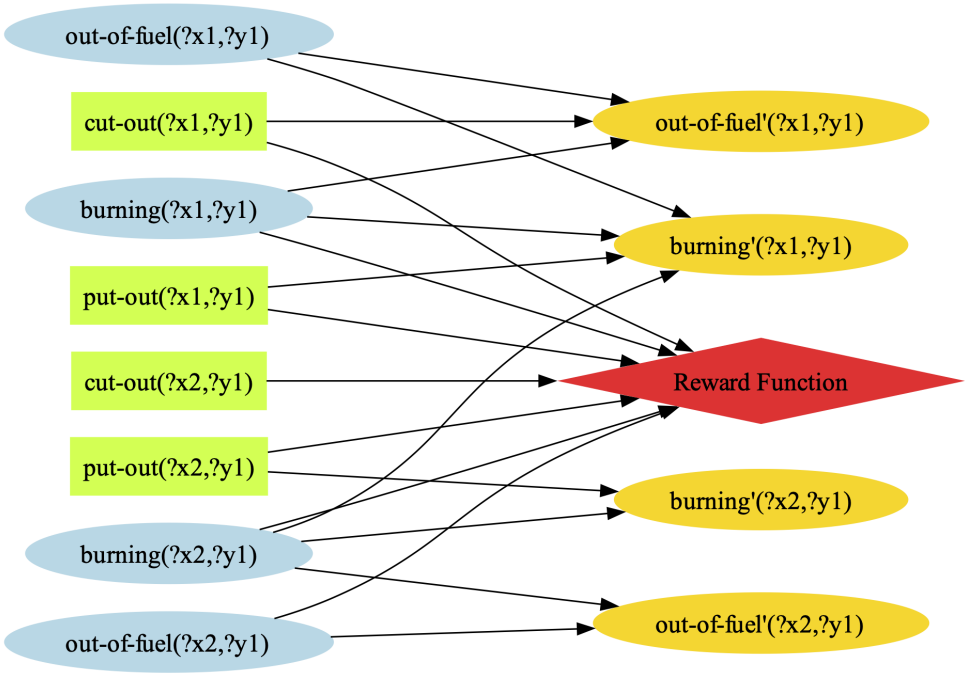


Figure 3.7.: DBN and ID of the Wildfire domain.

The instance graph is a representation of a GAT, which is used to compute node embeddings aggregating information from the respective neighbourhood. For each action type, a corresponding action decoder is created. These action decoders are used to calculate a score for every action based on the concatenation of the node embeddings the action affects. The results are then normalized to return a policy.

4. Results

This chapter discusses the accomplishments of the thesis, as well as the challenges encountered in attempting to replicate the findings of SymNet3.0. The thesis aimed to create a robust framework for the learning of generalized neural policies through graph neural networks for planning problems expressed in RDDL to establish a solid foundation for future research.

During the progress of the thesis, several unexpected challenges were encountered that revealed a more complex scope than initially anticipated. Although SymNet3.0 is a novel approach for generating generalized neural policies across selected domains, it depends heavily on outdated tools and requires domain-dependent hard-coding, to generate graphs. This dependence highlights the inherent limitations of the current framework.

Addressing these unforeseen challenges consumed a considerable amount of time, but it was crucial for the progress of this thesis. Among other things, this has led to the development of a multi-platform supporting dockerised integration of PROST and pyRDDLGym, significantly simplifying the evaluation process. The transition to pyRDDLGym not only facilitated the seamless integration of additional reference planners [7, 20] and supported the use of the latest language features since it is the official tool for parsing, simulating and evaluating planning problems defined in RDDL.

The thesis also achieved domain-independent parsing and graph generation without the need for hardcoding, marking a significant improvement over the work by Sharma et al. [26]. The integration of continuous integration practices into the development workflow ensures the consistent reliability and integrity of the framework through automated testing of updates and expansions. Strictly defined dependencies further enhance the framework's reproducibility.

Despite these achievements, the thesis encountered difficulties in replicating the results presented by Sharma et al. [26]. The reported results could not be independently verified, and testing was limited to the Wildfire domain due to time constraints. Despite exploring a range of hyperparameters, the model failed to learn, partly due to the fact that the XADD compilation of CPF does not support state-invariants. This resulted for some domains in partly wrong adjacency matrices and, consequently, the generation of divergent graphs. Additionally, most of the hyperparameters are extracted from the original source code, and these parameters may not match the ones that produced the results reported by Sharma et al. [26].

In summary, the thesis has made notable improvements in the infrastructure for learning generalized neural policies using graph neural networks, but it has also highlighted some

remaining challenges in reproducing prior results. The advancements made lay a solid foundation for future research, offering enhanced tools. However, the mentioned issues must still be addressed to verify the results claimed by Sharma et al. [26], and depending on the outcome, potential future steps can be taken as mentioned in Chapter 6.

5. Related Work

The IPC is organized in the context of the ICAPS and has been a platform for comparison of planning and scheduling approaches since 1998. The IPPC is a part of the IPC that focuses on evaluating the latest probabilistic planning systems on various benchmark problems through empirical means. Since 2011, RDDL has been the standard planning language for expressing probabilistic problems. A basic distinction is made between discrete and continuous problems as well as online and offline planners. Online planners are decision-making systems that interact with the environment in real-time, which enables them to adjust to changes in the environment. In contrast, offline planners rely on a pre-calculated plan and are thus dependent on a relatively static environment that does not change frequently.

In 2011, Keller and Eyerich [10] introduced their probabilistic online planner PROST that is based on UCT. The planner won the probabilistic discrete MDP track of the IPPC in 2011 and 2014, and was used as a reference planner at the IPPC in 2018 to facilitate better comparison of results. At the IPPC in 2018, PROST and its modifications took the first four places, owing to their dominance and overall good performance.

Although PROST has demonstrated superiority in previous competitions, the online planner requires a significant amount of computational power to make quick decisions, while also being slow due to the need to perform search and reasoning at every step. Additionally, it is not possible to transfer knowledge between instances of the same domain, meaning generalization is not achievable. Despite these limitations, PROST is often used as a benchmark or to generate training data, as it generally performs well and can be used for imitation learning.

Issakkimuthu et al. [9] proposed a method to learn *deep reactive policies* (DRPs) using supervised learning for probabilistic planning problems expressed in RDDL. Their approach imitates the action choices of non-reactive planners to learn from them. The DRP is represented as a deep neural network, enabling the evaluation through a single feed-forward pass. However, their approach does not focus on generalizing the learned policy across different instances of the same domain [9].

A major challenge of their approach is the design of the deep neural network architecture, which requires a trade-off between overfitting and generalization. Depending on the complexity of the domain, different architectures are necessary to achieve optimal results. Consequently, a DRP needs to be trained for each RDDL domain, involving the generation of training data using imitation learning and parameter tuning of the selected DRP architecture [9]. The quality of the expert planner is critical in determining the performance of the DRP. Despite the challenges, Issakkimuthu et al. [9] report that there is only a minor drop in performance but with a dramatic improvement in

decision-making time.

In 2018, Bajpai et al. [1] presented a domain-independent transfer algorithm for planning domains expressed in RDDL. Their approach exploits the by RDDL given DBN structure to generate a state embedding using a *graph convolutional network* (GCN) [1]. They then used this generated state embedding to train an *asynchronous advantage actor-critic* (A3C) RL agent, which could then be transferred across instances. Essentially, by learning a latent space representation of the given state and action spaces, in which the neural policy was learned, Bajpai et al. [1] were able to transfer knowledge to new instances of the same domain by relearning a mapping from the latent action space to actual actions.

Neural planners are capable of acquiring latent representations that allow them to transfer learned policies from one instance to another instance within the same domain. This approach becomes significantly advantageous when there is a need to transfer a policy faster instead of going through the time-consuming process of training it from scratch. However, for every instance, the action decoder needs to be retrained. Additionally, knowledge transfer is limited to equi-sized problems due to its fixed-size action decoder [1].

Garg et al. [4] have addressed the shortcomings of the method proposed by Bajpai et al. [1] by introducing a new technique that enables size-independent neural transfer for planning problems expressed in RDDL. The approach architecture is similar to the one used by Bajpai et al. [1], using A3C to train problem-independent policy and value networks. However, a GAT instead of a GCN is used to obtain size-invariant embeddings enabling size-independent transfer. The authors also aimed to achieve a good zero-shot setting that leads to better generalization and superior learning curves compared to a policy trained from scratch. However, the approach presented by Garg et al. [4] has its limitations. For instance, it does not support the handling of fluents parameterizable with more than one object, except for one binary non-fluent.

In previous research, the main challenge was to find an appropriate problem representation that could support problem-solving independent of both the domain and the instance. To address this challenge, SymNet was proposed by Garg et al. [5], a neural approach for solving RMDPs expressed in RDDL. For problem representation, they utilized the ground DBN available in RDDL, which provides the connectivity structure that determines the interdependence between states and actions. Each instance's ground DBN is transformed into a directed instance graph that serves as input for a GAT. The GAT returns node embeddings and a state embedding. Via a value and policy decoder, the latent representations of the tuple embeddings and the state embedding are mapped to a state value function and policy, respectively.

To address the challenges introduced by a generalized policy decoder, Garg et al. [5] construct a policy and value decoder for the number of distinct actions. The concrete architecture, therefore, depends on the specific RDDL domain and instance. For training

the state encoder, as well as the policy and value decoder, the general approach of the model-free A3C RL algorithm is applied. In other words, the parameters for the encoder and decoders are learned using updates, and the loss function for the policy and value network is identical to the one from A3C. Subsequently, using the trained networks, a policy can be computed with a single forward pass [5].

However, the design choice has several disadvantages. Since the DBN only contains part of the relevant information and lacks information about objects and their relation to actions, generalizability is poor and different actions may not be identifiable [5].

The SymNet2.0 paper, authored by Sharma et al. [25], is an improved version of the SymNet architecture. It achieves this by enhancing the graph with additional information, which leads to an improvement of the decoders. The SymNet architecture had non-fluents included in the DBN, which represented the graph structure used in SymNet but were compiled away. SymNet2.0 inserts singleton nodes and connects them to object-tuple nodes via edges. This makes it easier to handle action identification during action decoding, especially for action arguments that were not identifiable [25].

Additionally, SymNet2.0 introduces a new set of graphs, namely position-based graphs, action-induced graphs, and a graph that captures action-independent effects. Position-based graphs capture the argument position of a state- or non-fluent an object takes. The other two graphs, already known from SymNet, contain knowledge about which fluents and objects are action-independent and which state-fluents are influenced by actions. Edges in the latter graph show which state-fluents can be reached by actions from other state fluents.

Like SymNet, SymNet2.0 uses imitation learning for training, using data generated by the online planner PROST. This approach helps to circumvent the training and exploration issues faced by RL algorithms [25]. Although PROST outperforms SymNet2.0 on most instances, SymNet2.0 occasionally performs better on larger instances. However, this is largely due to the given time limits for PROST [25].

The SymNet2.0 framework addresses several of the shortcomings that were present in the original SymNet architecture. One of the most significant improvements is the inclusion of previously missing information into the updated general representation. This is accomplished by introducing new graph structures that are specifically tailored to the particular domain and instance being analysed. However, there are still some challenges that need to be overcome. For instance, in large instances, the memory requirements can become too high [25], making the approach unfeasible. Additionally, detecting long-range dependencies [25] has been identified as a major problem. Due to the fixed depth of the GAT, each node can only access information from its immediate neighbours. As a result, each node only obtains knowledge about the local environment. This can sometimes lead to situations where relevant information is not available, causing certain actions to receive an identical score for different states [25]. To mitigate this problem, increasing the depth of the GAT is one possible solution, but it often leads to a blow-up

in parameters and therefore is not a viable solution.

In their latest effort, SymNet3.0, Sharma et al. [26] have tackled the challenges mentioned earlier, aimed at learning generalized neural policies using GNNs. This builds on their previous attempts, SymNet and SymNet2.0, with the fundamental framework involving offline planning and training through supervised learning using data generated by PROST.

The architecture in SymNet3.0 consists mainly of an instance graph and an influence graph. The latter is utilized to identify long-range dependencies by capturing the influence distance over state variables. The weight is determined by the distance between different nodes. To learn node embeddings and aggregate neighbourhood information, GATs are utilized. The attention mechanism ensures that the neighbours are weighed according to their significance [26].

In the proposed SymNet3.0 framework, the concatenated node embeddings serve as the input for the action decoder. To enable the generation of diverse actions, there is one decoder per action type. The parameterized actions are calculated, and Softmax is taken over all scores to obtain the policy.

However, it's worth noting that Symnet3 depends on PROST, making it unsuitable for domains without interm-fluents or continuous domains. Additionally, the framework's reliance on pairwise distances between nodes may not capture policies that depend on distances between a group of nodes simultaneously [26].

6. Conclusion and Future Work

6.1. Conclusion

RDDL can be considered the default planning language for the probabilistic planning and reinforcement learning track of the IPPC. It is designed to model complex stochastic real-world planning scenarios which are difficult or impossible to express using other available planning languages. Therefore, it is desirable to solve domain-independent planning problems expressed in RDDL.

However, pyRDDLGym, which is the official toolkit for working with RDDL problems, is still in its early stages. Several bugs had to be fixed, and features were integrated one after the other. Due to the absence of a *continuous integration* (CI) process, some fixes introduced bugs in other parts of the code. This led to a continuous loop of updating package dependencies, identifying and solving new bugs locally, writing pull requests or issues, and waiting until they were merged or solved by other contributors.

Unfortunately, this loop was inevitable from the thesis perspective as there was a need to integrate the fixed bugs to get closer to the overall goal. Nonetheless, this took considerably more time than expected and affected the overall progress of the thesis. Since the main objective of the thesis was also to provide a solid foundation for learning generalized neural policies with graph neural networks, it was not an option to rely on outdated and only temporarily working solutions.

Re-implementing the code from scratch proved to be a challenging task due to the lack of documentation and the non-executable nature of the original code. In addition to pure re-implementation, changes were made to the framework removing the need for domain-specific hard coding, making it truly domain-independent. Furthermore, PROST, a reference planner was integrated into the framework.

Unfortunately, due to limited time, the official results presented by Sharma et al. [26] could not be reproduced. This is partially caused by the current XADD compilation, which ignores state-invariants leading to partially “incorrect” adjacency matrices and subsequently to different graphs. Furthermore, most of the hyperparameters are extracted from the original code, and these parameters may not match the ones that produced the results reported by Sharma et al. [26]. Despite this, we assume that the approach is working in general, based on the positive results of a similar approach used for planning problems defined in PDDL by Ståhlberg et al. [29].

6.2. Future Work

The success of SymNet3.0 relies heavily on the quality of the reference planner used to generate the imitation learning dataset. Currently, SymNet3.0 uses PROST for this purpose. However, PROST is limited to discrete domains that do not require the use of intermediate fluents. As many complex domains require intermediate fluents, it is important to find new reference planners that can guarantee the imitation learning process for those domains. This would also impact the overall approach of building graphs and how to incorporate intermediate fluents.

Similar to PROST, it is important to identify superfluous actions and integrate the gained knowledge, as the respective action probability can immediately be reduced to zero. However, since a node within a graph can potentially represent multiple parameterized variables, in most cases it will not be possible to exclude them from the graph.

Additionally, it would be useful to explore better ways to share experimental results and data. Generating the imitation learning data with PROST is time-consuming and computationally expensive. As a result, hyperparameter tuning is often not an option, which may distort the results and potentially favour one's own results. This not only distorts the results but also reduces the quality of the imitation learning dataset, leading to potentially worse overall results.

One of the next steps is to integrate the upcoming enhancements of the symbolic toolkit of pyRDDLGym. Currently, the XADD compilation does not evaluate state-invariants, which leads to incorrect DBN and XADD representations. As they are the current foundation for extracting the adjacency matrix of the used graph structure, we obtain deviating graph representations compared to SymNet3.0, leading to different results.

Bibliography

[1] Aniket Bajpai, Sankalp Garg, and Mausam. Transfer of deep reactive policies for MDP planning. *CoRR*, abs/1810.11488, 2018. URL http://arxiv.org/abs/1810.11488.

[2] Craig Boutilier, Raymond Reiter, and Bob Price. Symbolic Dynamic Programming for First-Order MDPs. In *International Joint Conference on Artificial Intelligence*, 2001. URL https://api.semanticscholar.org/CorpusID:14567948.

[3] Floris den Hengst, Vincent Francois, Mark Hoogendoorn, and Frank Harmelen. Planning for potential: efficient safe reinforcement learning. *Machine Learning*, 111, 06 2022. doi: 10.1007/s10994-022-06143-6.

[4] Sankalp Garg, Aniket Bajpai, and Mausam. Size independent neural transfer for RDDL planning. *CoRR*, abs/1902.03081, 2019. URL http://arxiv.org/abs/1902.03081.

[5] Sankalp Garg, Aniket Bajpai, and Mausam . Symbolic network: Generalized neural policies for relational MDPs. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 3397–3407. PMLR, 13–18 Jul 2020. URL https://proceedings.mlr.press/v119/garg20a.html.

[6] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In *International Conference on Machine Learning*, 2017. URL https://api.semanticscholar.org/CorpusID:9665943.

[7] Mike Gimelfarb. pyRDDLGym JAX Planner. https://github.com/pyrddlgym-project/pyRDDLGym-jax, 2024. Accessed: 2024-03-22.

[8] Prof. Dr. Jörg Hoffmann. Ai planning: Applications - a few problems we can solve (and which some people care about). University of Saarland, 2018. URL https://fai.cs.uni-saarland.de/teaching/winter18-19/planning-material/planning04-applications-post-handout.pdf.

[9] Murugeswari Issakkimuthu, Alan Fern, and Prasad Tadepalli. Training Deep Reactive Policies for Probabilistic Planning Problems. *Proceedings of the International Conference on Automated Planning and Scheduling*, 28(1):422–430, Jun. 2018. doi: 10.1609/icaps.v28i1.13873. URL https://ojs.aaai.org/index.php/ICAPS/article/view/13873.

[10] Thomas Keller and Patrick Eyerich. PROST: Probabilistic Planning Based on UCT. In *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling (ICAPS 2012)*, pages 119–127. AAAI Press, 2012.

[11] Thomas Keller and Malte Helmert. Trial-based Heuristic Tree Search for Finite Horizon MDPs. In *Proceedings of the Twenty-Third International Conference on Automated Planning and Scheduling (ICAPS 2013)*, pages 135–143. AAAI Press, 2013.

[12] Wenzhe Li, Hao Luo, Zichuan Lin, Chongjie Zhang, Zongqing Lu, and Deheng Ye. A survey on transformers in reinforcement learning, 2023.

[13] Daniel Mankowitz, Andrea Michi, Anton Zhernov, Marco Gelmi, Marco Selvi, Cosmin Paduraru, Edouard Leurent, Shariq Iqbal, Jean-Baptiste Lespiau, Alex Ahern, Thomas Köppe, Kevin Millikin, Stephen Gaffney, Sophie Elster, Jackson Broshear, Chris Gamble, Kieran Milan, Robert Tung, Minjae Hwang, and David Silver. Faster sorting algorithms discovered using deep reinforcement learning. *Nature*, 618:257–263, 06 2023. doi: 10.1038/s41586-023-06004-9.

[14] Mausam and Daniel S. Weld. Solving relational mdps with first-order machine learning, 2004. URL `https://api.semanticscholar.org/CorpusID:16297298`.

[15] Nicola J. Müller. DSMC Evaluation Stages with Restart. Master's thesis, Saarland University, Saarbrücken, October 2022.

[16] Ladislav Rampášek, Mikhail Galkin, Vijay Prakash Dwivedi, Anh Tuan Luu, Guy Wolf, and Dominique Beaini. Recipe for a general, powerful, scalable graph transformer, 2023.

[17] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall Press, USA, 3rd edition, 2009. ISBN 0136042597.

[18] Scott Sanner. Relational Dynamic Influence Diagram Language (RDDL): Language Description, 2011.

[19] Scott Sanner and Mike Gimelfarb. pyRDDLGym RDDL repository. `https://github.com/pyrddlgym-project/rddlrepository`, 2023. Accessed: 2023-11-17.

[20] Scott Sanner and Mike Gimelfarb. pyRDDLGym Gurobi Planner. `https://github.com/pyrddlgym-project/pyRDDLGym-gurobi`, 2024. Accessed: 2024-03-22.

[21] Scott Sanner and Ayal Taitler. LH2: Introduction to MDP Modeling and Interaction via RDDL and pyRDDLGym. `https://pyrddlgym-project.github.io/AAAI24-lab`, 2024. Accessed: 2024-03-22.

[22] Scott Sanner, Ayal Taitler, and Mike Gimelfarb. pyRDDLGym Documentation: RDDL Language Description. https://pyrddlgym.readthedocs.io/en/latest/rddl.html, 2024. Accessed: 2024-03-22.

[23] Scott Sanner, Ayal Taitler, and Mike Gimelfarb. pyRDDLGym Github Repository. https://github.com/pyrddlgym-project/pyRDDLGym, 2024. Accessed: 2024-03-22.

[24] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, Timothy P. Lillicrap, and David Silver. Mastering atari, go, chess and shogi by planning with a learned model. *CoRR*, abs/1911.08265, 2019. URL http://arxiv.org/abs/1911.08265.

[25] Vishal Sharma, Daman Arora, Florian Geißer, Mausam, and Parag Singla. SymNet 2.0: Effectively handling Non-Fluents and Actions in Generalized Neural Policies for RDDL Relational MDPs. In James Cussens and Kun Zhang, editors, *Proceedings of the Thirty-Eighth Conference on Uncertainty in Artificial Intelligence*, volume 180 of *Proceedings of Machine Learning Research*, pages 1771–1781. PMLR, 01–05 Aug 2022. URL https://proceedings.mlr.press/v180/sharma22a.html.

[26] Vishal Sharma, Daman Arora, Mausam ., and Parag Singla. SymNet 3.0: Exploiting Long-Range Influences in Learning Generalized Neural Policies for Relational MDPs. In *The 39th Conference on Uncertainty in Artificial Intelligence*, 2023. URL https://openreview.net/forum?id=sWzgZUmJich.

[27] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy P. Lillicrap, Karen Simonyan, and Demis Hassabis. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *CoRR*, abs/1712.01815, 2017. URL http://arxiv.org/abs/1712.01815.

[28] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of go without human knowledge. *Nature*, 550:354–359, 10 2017. doi: 10.1038/nature24270.

[29] Simon Ståhlberg, Blai Bonet, and Hector Geffner. Learning Generalized Policies Without Supervision Using GNNs, 2022.

[30] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018. URL http://incompleteideas.net/book/the-book-2nd.html.

[31] Petar Velickovic. TikZ: Graph Convolution. https://github.com/PetarV-/TikZ/tree/master/Graph%20convolution, 2024. Accessed: 2024-01-13.

[32] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio', and Yoshua Bengio. Graph attention networks. *ArXiv*, abs/1710.10903, 2017. URL https://api.semanticscholar.org/CorpusID:3292002.

[33] Zhanghao Wu, Paras Jain, Matthew A. Wright, Azalia Mirhoseini, Joseph E. Gonzalez, and Ion Stoica. Representing long-range context for graph neural networks with global attention. *CoRR*, abs/2201.08821, 2022. URL https://arxiv.org/abs/2201.08821.

GitHub (Issue) References

[34] Daman Arora. Missing and deprecated dependencies. https://github.com/dair-iitd/symnet3/issues/1, 2023. Accessed: 2023-08-14.

[35] Mike Gimelfarb. ArithmeticError: unable to short-circuit in vectorized environments. https://github.com/pyrddlgym-project/pyRDDLGym/issues/235, 2024. Accessed: 2024-01-22.

[36] Mike Gimelfarb. Incorrect immediate reward returned to PROST. https://github.com/pyrddlgym-project/pyRDDLGym/pull/240, 2024. Accessed: 2024-02-05.

[37] Mike Gimelfarb. RDDLSimServer: Truncated log files. https://github.com/pyrddlgym-project/pyRDDLGym/issues/254, 2024. Accessed: 2024-03-22.

[38] Scott Sanner. XADD compilation of CPFs ignores state-invariants. https://github.com/pyrddlgym-project/pyRDDLGym-symbolic/issues/3, 2024. Accessed: 2024-02-18.

[39] Scott Sanner and Mike Gimelfarb. Missing PROST integration. https://github.com/pyrddlgym-project/pyRDDLGym/issues/214, 2023. Accessed: 2023-12-23.

[40] Scott Sanner and Jihwan Jeong. XADD DBN representation ignores state-invariants. https://github.com/pyrddlgym-project/pyRDDLGym-symbolic/issues/2, 2024. Accessed: 2024-02-18.

[41] Scott Sanner, Ayal Taitler, and Mike Gimelfarb. JAX Planner Performance Issues on Discrete Control Problems. https://github.com/pyrddlgym-project/pyRDDLGym/issues/224, 2023. Accessed: 2023-12-29.

[42] Vishal Sharma and Daman Arora. Missing source code due to incorrect .gitignore file. https://github.com/dair-iitd/symnet3/issues/2, 2023. Accessed: 2023-08-15.

[43] Vishal Sharma, Daman Arora, and Mausam. Segmentation fault (core dumped) error. https://github.com/dair-iitd/symnet3/issues/3, 2023. Accessed: 2023-10-04.

Appendices

A. Appendix

A.0.1. Hardware Specifications

CPU	
Model name	AMD Ryzen 9 5950X 16-Core Processor
Architecture	x86_64
CPU Cores	16
Threads	32
Base Clock	3.4GHz
L1 Cache	1MB
L2 Cache	8MB
L3 Cache	64MB
GPU	
Model name	NVIDIA GeForce RTX 3060 LHR
GPU Model	GA106-302 (GA106)
CUDA Cores	3584
Tensor Cores	112
Base Clock	1320MHz
Memory	12GB G6
Memory	
Total size	128GB
Model name	Kingston KF3600C18D4/32GX
Type	DDR4 DIMM
Capacity	32GB
Clock speed	2400MHz

Table A.1.: Hardware Specifications