UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DSIC
DEPARTAMENT DE SISTEMES INFORMÀTICS I COMPUTACIÓ

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Dept. of Computer Systems and Computation

Approaching Generalized Planning using Deep Reinforcement Learning and Graph Neural Networks

Master's Thesis

Master's Degree in Artificial Intelligence, Pattern Recognition and Digital Imaging

AUTHOR: Aso Mollar, Ángel

Tutor: Onaindia de la Rivaherrera, Eva

ACADEMIC YEAR: 2021/2022

DSIC DEPARTAMENTO DE SISTEMAS INFORMÁTICOS Y COMPUTACIÓN

Universitat Politècnica de València

Department of Computer Systems and Computation

MASTER'S THESIS

Approaching Generalized Planning using Deep Reinforcement Learning and Graph Neural Networks

---

Author:
Ángel Aso Mollar

Supervisor:
Eva Onaindía de la Rivaherrera

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Abstract

One of the greatest challenges of Artificial Intelligence has been and still is the ability to infer solutions from small problems or even subproblems that can generalize over larger instances. The goal of Generalized Planning in the field of Automated Planning is precisely to come up with general principles that are valid beyond the data used to infer such principles.

In this work we offer a perspective based on Deep Reinforcement Learning (DRL), in which we try to learn a policy capable of solving complex planning problem instances by learning the underlying structure through small problems of the same domain. This is achieved by using an ideal representation of the planning tasks in the form of a graph that encodes their structure.

Particularly, we use Graph Neural Networks (GNN), which are a type of Neural Network capable of working directly on graphs and taking advantage of their structure to get the most out of the information. With all this, it will be shown that the policy learned by means of DRL and GNN generalizes well over instances of several orders of magnitude higher than those for which it has been trained.

Keywords: Artificial Intelligence, Automated Planning, Generalized Planning, Deep Reinforcement Learning, Graph Neural Networks

Resumen

Uno de los grandes retos de la Inteligencia Artificial ha sido y sigue siendo la capacidad de inferir soluciones dadas instancias pequeñas de un problema, o incluso subproblemas, que puedan generalizar hacia otras de tamaño mayor. El objetivo de la Planificación Generalista en el campo de la Planificación Automática es, precisamente, encontrar principios generales válidos más allá de los datos que se han utilizado para encontrarlos.

En este trabajo, ofrecemos un acercamiento basado en Deep Reinforcement Learning (DRL), en el cual se ha intentado aprender una política capaz de resolver instancias de problemas complejos a partir del aprendizaje de la estructura subyacente del dominio con problemas más pequeños del mismo. Esto se ha conseguido utilizando una representación concreta de las tareas de planificación a través de un grafo, el cual codifica su estructura.

En particular, se han utilizado Graph Neural Networks (GNN), que son un tipo concreto de Red Neuronal capaz de trabajar directamente con grafos y que usa esta misma estructura para sacar la máxima información del problema. Con todo esto, se verá que la política aprendida gracias mediante las técnicas DRL y GNN generaliza bien hacia instancias de varios órdenes de magnitud superiores que con los que se había entrenado.

Palabras clave: Inteligencia Artificial, Planificación Automática, Planificación Generalista, Aprendizaje por Refuerzo Profundo, Redes Neuronales de Grafos

Resum

Un dels grans reptes de la Intel·ligència Artificial ha sigut i continua sent la capacitat d'inferir solucions donades petites instàncies d'un problema, o inclús subproblemes, que puguen generalitzar cap a altres de grandària major. L'objectiu de la Planificació Generalista en el camp de la Planificació Automàtica és, precisament, trobar principis generals vàlids més enllà de les dades que s'han utilitzat per a trobar-los.

En este treball, oferim una proposta basada en Deep Reinforcement Learning (DRL), en el qual s'ha intentat aprendre una política capaç de resoldre instàncies de problemes complexos a partir de l'aprenentatge de l'estructura subjacent del domini amb problemes més xicotets del mateix. Açò s'ha aconseguit utilitzant una representació concreta de les tasques de planificació a través d'un graf, el qual codifica la seua estructura.

En particular, s'han utilitzat Graph Neural Networks (GNN), que són un tipus concret de Xarxa Neuronal capaç de treballar directament amb grafs i que usa esta mateixa estructura per a traure la màxima informació del problema. Amb tot açò, es veurà que la política apresa per mitjà de les tècniques de DRL i GNN generalitza bé cap a instàncies de diversos ordes de magnitud superiors que amb els que s'havia entrenat.

Paraules clau: Intel·ligència Artificial, Planificació Automàtica, Planificació Generalista, Aprenentatge Profund per Reforç, Xarxes Neuronals de Grafs

Contents

Chapter 1

Introduction

In the field of Artificial Intelligence, Automated Planning is a discipline that attempts to find strategies of actions that solve a given problem, generally within a description domain. These are usually search problems in very large state spaces that require defining heuristics or other strategies to facilitate their resolution. As stated in [GHA16]:

"We are interested here in the study of computational deliberation capabilities that allow an artificial agent to reason about its actions, choose them, organize them purposefully, and act deliberately to achieve an objective."

With the rise of Machine Learning and, especially, with the development of Neural Networks, many are the works that try to combine or introduce knowledge of this area within the field of Automated Planning. We, in particular, place ourselves within Generalized Planning, which tries to infer knowledge that is capable of creating strategies that exceed the orders of magnitude in the problems and creates generalist sequences of actions valid for all types of instances.

Specifically, we make use of Graph Neural Networks, which are a novel type of Neural Networks that work on graph structures to infer knowledge, and we also use them in conjunction with a Reinforcement Learning module, with which we will create a policy that will allow us to infer such strategies.

In chapter two we will talk about different lines of work related to what is going to be developed in this Master's Final Project, while in chapter three we will describe the concepts and models that we are going to use for this development. Chapter four will focus on explaining the architecture of the system on which all the work has been based, and then in chapter five we will develop a series of experiments to test the usefulness of the system. Chapter six will focus on an analysis of the results of these experiments to see where there is room for improvement, and will also present modifications to help improve the system. Finally, chapter seven will focus on the conclusions drawn from all the work and future lines of research that open up with this work.

1.1 Motivation

This project is motivated by the reading of the paper [RIV20]. We found the approach very interesting, especially the part of the PDDL problem representation as a graph structure and the use of Graph Neural Networks to create the Reinforcement Learning policy. That is why we were interested in understanding and analyzing the work, trying to draw more complete conclusions from what was stated there.

Since the article is a preprint of a workshop, we thought it was interesting to make an introspection inside it, understanding how the representation works, how it performs within the domains proposed in the paper and in others that we added later on, and why it works well in some but not in others.

To do this, we created a taxonomy with which we classified the proposed domains and tried to establish a hypothesis about the performance of the proposal within them. This hypothesis tries to be justified with the definition of another architecture a little more elaborate, which is tested with the domains that have given the worst results and it is observed that it improves the performance.

1.2 Objectives

With the above, we state the following specific objectives that we want to achieve in the development of this work:

- Understand the technology developed in the article and explain it in detail.
- Assemble the architecture successfully within the latest existing libraries in the field.
- Choose the domains that we want to use to experiment with this architecture.
- Develop a classification taxonomy for these domains.
- Experiment within these domains and draw conclusions regarding the taxonomy presented.
- Analyze points of potential improvement and develop new architectures to improve results.
- Conclude whether such experiments actually improve results and understand why.

Chapter 2

Related work

In the area of Artificial Intelligence (AI), a controller is a system that receives situations or states occurring in a dynamic and not completely predictable environment, and selects the most appropriate actions for achieving some goals. There are two main approaches to the task of designing a controller: the planning approach that builds the controller based on the specification of a model of the environment or domain, and the learning approach where the control is inferred from a set of experiences gathered from the environment.

These approaches, also known as *model-based solvers* and *model-free learners* [GEF18] respectively, highlight the advantages and limitations of both perspectives. The planning approach generates systems that are transparent, flexible and generic, but require a model designed by an expert and are computationally expensive. The learning approach, however, generates systems categorized as black-boxes that work for a fixed size input, but their techniques are much faster and capable of inferring hidden relationships within the data.

In the last few years, some techniques that integrate planning and learning have been developed in order to identify the dimensions of a problem in which it is more convenient to apply one algorithm or another.

2.1 Planning controllers

One of the big limitations of the planning approach is the need to have a model that describes the state variables of the problem and the executable actions in the environment [GHA04]. In this line there exist works whose objective is to learn the model of actions from observations of the environment. ARMS [YAN07] and SLAF [AMI08] are two of the first algorithms capable of learning the actions of a domain from plan traces with partial or null observation of intermediate states. ARMS offers no guarantee that the learned model can correctly explain the test set traces, and SLAF does not exhibit high accuracy in learning the preconditions of actions.

These two systems were followed by others such as LAMP [ZHU10] or AMAN [ZHU13], which introduce the concept of noise or probability of observing actions incorrectly and LOUGA [KUC18] or LOCM [CRE13], whose distinguishing feature is its ability to learn state variables. Finally, FAMA is one of the most recent approaches for learning action models with minimal observability [AIN19]. An example of this type of controller can be seen in Figure 2.1.

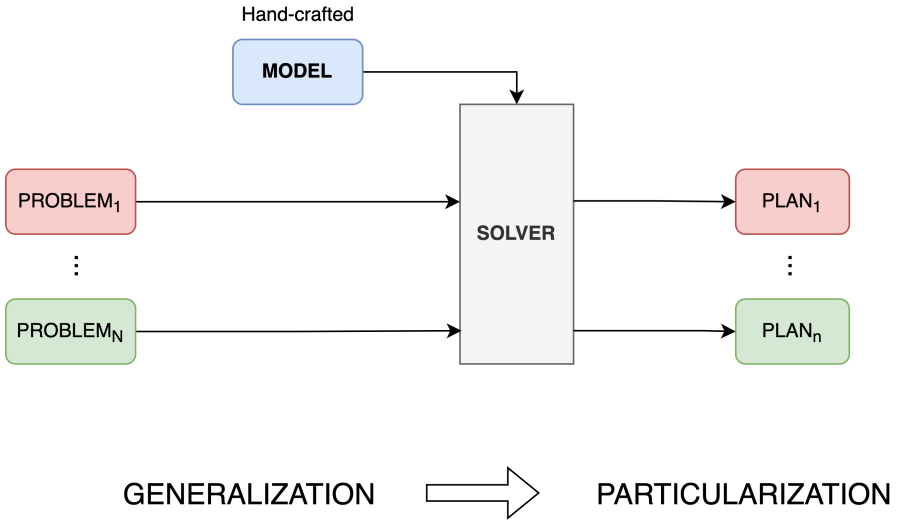

Figure 2.1: An example of a controller within the planning perspective. With a single hand-crafted model of the domain, the solver (planner) is able to compute a plan for each problem. This perspective uses a generalized model to create particular solutions.

2.2 Learning controllers

We now move to the learning approach where one of the most successful techniques for learning controllers is Deep Reinforcement Learning (DRL). Research using planning in a DRL environment, which is known as Model-based Deep Reinforcement Learning (MDRL), aims to build a predictive model of the environment. MDRL uses the learned model not only to obtain a reward but to predict the state resulting from applying an action. This allows understanding the world, predicting and controlling the agent's experience when acting in the environment [SUT18].

Models in MDRL are mathematical models of sequential decision making in non-deterministic environments such as Markov Decision Processes (MDPs) [PUT94]. The integration of these models in MDRL has led to very successful applications in robotics [NGU11] [EBE18], gaming [SIL18] [KAI20] and autonomous driving [YUR20] [PER22].

More recent work in this area focuses on exploring the benefits of incorporating planning into a MDRL agent such as studying the degree of generalization that a planner can provide (it is worth recalling that DRL is efficient for solving tasks of a

given type and size, but finds it difficult to generalize the learned model to another set of tasks) [HAM21], as well as proposals to design a common framework that integrates planning and learning [MOE22]. An example of this type of controller can be seen in Figure 2.2.

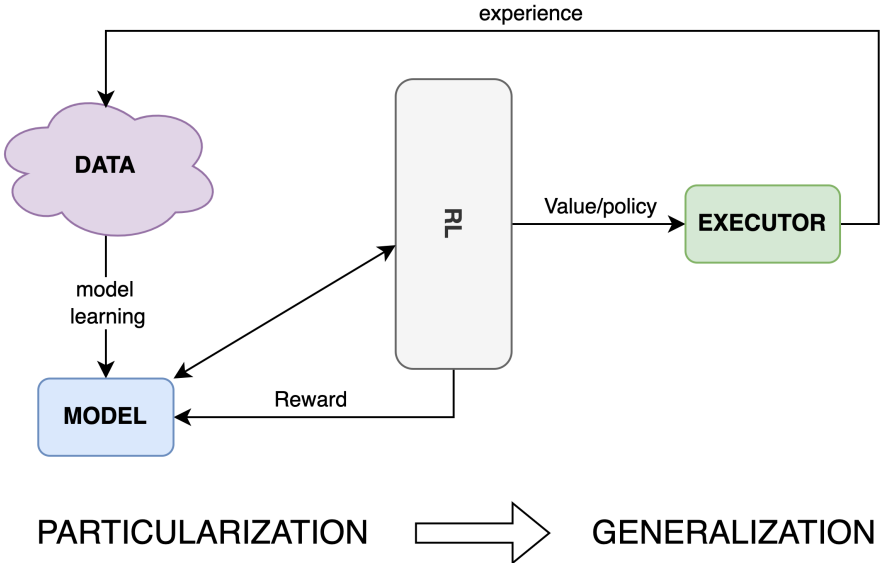

Figure 2.2: An example of a controller within the Reinforcement Learning perspective. From a collection of data a predictive model is learnt which can then be used to calculate a policy. Under this perspective a particular model is learned (particularly adapted to the data) from which a general solution (policy) is inferred.

2.3 Learning to plan controllers

So far we have presented two different lines of integration of planning and learning for the design of controllers in AI. The first line exploits techniques of a Machine Learning module in order to learn a deterministic action model that will later be used by a classical planner. The second line integrates a non-deterministic action planning module into a learning agent.

While classical planning models do not allow expressing the nondeterminism of an environment, high-level representation languages such as PDDL [MCD98] [FOX03] can be used to specify the model of a domain, thus avoiding the need of an explicit representation of the entire space of states and actions, as it is typically the case in most nondeterministic models.

Moreover, the advances in classical planning in recent years allow to solve satisfactorily any task of a domain represented with PDDL, which confers a high degree of generalization as it does not depend on the size of the input problem. However, the generalization in the input data does not carry over to the output since a classical planner returns a plan solution for a particular planning task and this solution is not generalizable to other tasks. Interestingly, this contrasts with

stochastic model solvers, which return a policy, i.e. the best action to execute in each state, and this policy is applicable to any task. All this highlights the benefits of using each approach to solve one type of problem.

Taking advantage of these synergies, we can find a recent line of research using MDRL techniques applied to a domain model specified in PDDL and several task instances with the goal of learning a policy and/or value function that is subsequently used to design a heuristic function for a classical planner [RIV20] [GEH22]. This line of work leads to what is called "Generalized Planning", techniques capable of returning a solution that solves several planning tasks instead of a single one. This type of methods require the specification of a model of the environment, as in all methods that fall under the umbrella of the planning approach, and apply Relational Neural Network techniques to exploit the structure of the planning domain such as GNNs (Graph Neural Networks) [SEJ18] or NLMs (Neural Logic Machines) [DON19].

Chapter 3

Background

In this chapter we will describe the techniques used in our work, going through each one and defining them in a general way, with the purpose of making this work as self-contained as possible. We will talk about Planning, Graph Neural Networks, Reinforcement Learning and ways to train a reinforcement learning modules.

3.1 Planning

Classical planning is the problem of finding a sequence of actions that applied to an initial state lead to the achievement of a goal or a set of goals. The two main components of a planning task are the domain and the problem.

A **planning domain** represents the hand-crafted model of the planning perspective presented in the previous chapter (see Figure 2.1). Formally speaking, a planning domain is defined as a set $D = (F, A)$, where F is a set of fluents that describe properties of the domain objects and their relations, and A is a set of actions in which every $a \in A$ is defined by a triplet $(Pre(a), Add(a), Del(a))$; $Pre(a)$ are the preconditions that must be true in a state for the action to be applicable; $Add(a)$ are the effects of the action that assert a positive literal in the state that results after the application of a; and $Del(a)$ is the set of negative effects, i.e., the set of literals which become false after the action is applied.

A **planning problem instance** is defined as a tuple (I, G) linked to a domain $D = (F, A)$, in which $I, G \subseteq F$ are, respectively, the initial state and goals of the problem. We can think of every problem in Figure 2.1 as a different tuple (I_i, G_i). The purpose of planning is to find a solution to a problem (I_i, G_i). A solution is a sequence of actions or plan $\pi_i = \langle a_1, a_2, \ldots, a_k \rangle$, $a_i \in A \ \forall i$ such that the result of the application of the action sequence $\langle a_1, a_2, \ldots, a_k \rangle$ to the initial state I_i leads to a state $S \subseteq F$ that satisfies $G_i \subseteq S$, that is to say, that the state S contains every goal in G_i.

Classical planning uses a formal language called Planning Domain Definition

Language (PDDL, [MCD98]) to define domains and problems. More specifically, while the above definition of planning domain uses a grounded interpretation, a domain in PDDL is written as a lifted or first-order specification. This means that the properties of the domain objects and their relations are expressed as first-order predicates, and the set of actions A is actually specified as operators composed of an action name and a series of variable arguments. As an example, consider the following excerpt of a simple transportation domain in which we have objects of type `location`, `truck` and `box`:

```
(:predicates
      (at ?t - truck ?l - location)
      (at ?b - box ?l - location)
      (in ?b - box ?t - truck)
      (free ?t - truck)
)
```

The predicate `at` is used to specify a locatable object (`truck` or `box`) in a location. The predicate `in` denotes whether a box is inside a truck and the predicate `free` indicates whether a truck is free.

Let's suppose now a planning problem instance with two trucks identified as `t1` and `t2`, two boxes referred as `b1` and `b2` and two locations `locA` and `locB`. Assuming a state of the problem in which the truck `t1` is at `locA`, the box `b1` is inside the truck, the truck `t2` is at `locB`, and the box `b2` is at `locB`, the literals `(at t1 locA)`, `(in b1 t1)`, `(at t2 locB)`, `(free t2)` and `(at b2 locB)` would make up the state.

Planning problems have predominantly been approached from a state space search perspective. After all, a planning problem is nothing more than finding paths, usually as short as possible, within a structure. The search is usually guided by some kind of heuristic function, which makes this operation computationally less hard. We limit ourselves to classical planning that takes into account only instantaneous actions without duration, perfect information without losses and sequential planning.

One of the purposes of this work is to learn strategies or policies that solve multiple problems of a give domain D. That is, rather than finding a solution plan π_i to each problem (I_i, G_i) of a domain D, we aim to find a general plan Π that explains the behaviour of each individual π_i. The idea is to learn a general strategy for problems of domain D, from small problems to larger problems using neural networks.

3.2 Graph Neural Networks

Graphs are structures that model relations between a set of elements, using the concept of node (an object) and edge (a relationship between nodes). The structural property of graphs combined with Machine Learning (ML) has proven to be quite

useful in several fields because of its great expressive power, for example in social science [WU-L20], natural science [SAN18], natural language processing [WU21] and many others.

Formally speaking, as stated in [SAN18], a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is a duple of nodes $\mathcal{V} = \{1, ..., n\}$ and edges $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$, where $(i, j) \in \mathcal{E}$ denotes an edge from a node i to a node j. A node can represent whatever we want; from the ML perspective, an element (node, edge, etc.) is described with a vector called *feature vector*.

It is interesting to think about how we could represent a situation within a planning domain in the form of a graph. For example, considering the logistics example presented in the previous section, we can create a simple graphical representation of the scenario, as shown in Figure 3.1. We can see that the graph explains the aforementioned situation: truck `t1` is at `locA` and has the box `b1` inside it, so it is not `free`, while truck `t2` is `free`, and it is at `locB` like the box `b2`.

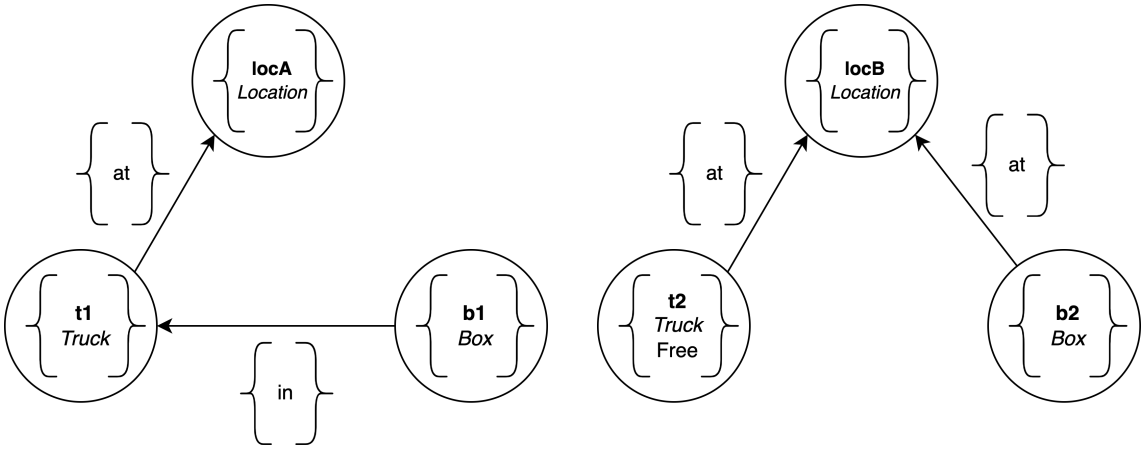

Figure 3.1: Example of a representation of a problem in a generic transport domain.

There are several aspects that must be taken into account if we want to follow this representation:

- The graph has to represent the taxonomy of the problem, i.e., has to be able to express types and objects.
- Features are not only important in nodes, but also in edges.
- As predicates are not symmetrical, the graph must be undirected.
- Nodes must represent properties intrinsic to objects while edges must represent properties between objects.
- There are also properties that are not intrinsic to an object, nor to a relation between two objects, so we have to be able to represent general information of the problem in some way.

In order to work with this structure, Graph Neural Networks (GNNs), have become a widely applied graph analysis method for the past few years, as stated in [ZHO20]. They are optimizable and permutational invariant transformations of all the elements in a graph, so their final purpose is to generate a suitable embedding of an element: a node, an edge or the graph itself. We can use GNNs to infer different types of information:

- At the node level, typically for tasks of node classification. For example, in the CORA dataset, a node-level task is to decide whether a paper (represented as a node) falls into one of seven categories based on the words inside it and the citations between other papers[1].
- At the edge level as in most recommender systems [WU-S20], where the objective is to predict the best relations between nodes in a graph of recommendation; i.e., we want to classify "good" and "bad" edges.
- At the graph level as in Figure 3.2, in which we want to know whether or not a graph contains two cycles.

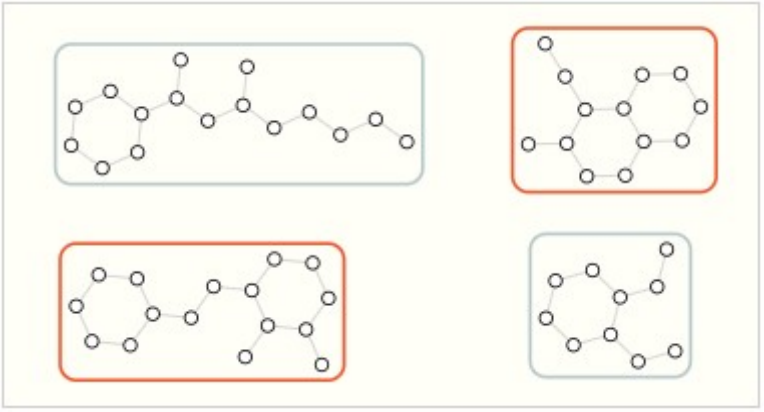

Figure 3.2: Graph classification task: Does the graph contain two rings?. In this situation we want to discriminate a network by whether or not it has two cycles within it. Source: `https://distill.pub/2021/gnn-intro/`. Accessed: 15-06-2022.

Graph Neural Networks use embeddings for each element of the graph, as we have already mentioned. This means that every node, edge and global property (these last two are optional) are represented by a vector. In the case of one-dimensional vectors, the representation would be a number.

A GNN layer is the operation of applying a transformation to the embeddings of the graph, generally by a process called message passing, which can be a convolution, a MLP or whatever flow of information we want to define. Figure 3.3 shows this transformation which intuitively follows the next steps for each node of the graph:

[1] `https://paperswithcode.com/sota/node-classification-on-cora`. Accessed: 15-06-2022.

it gets the neighbor embeddings of that specific node, then it aggregates them with a sort of aggregation operator, it transforms them by generally using a non-linear function and then it updates the information. Additional information of the edge and the graph embeddings can also be taken into account in this aggregation.

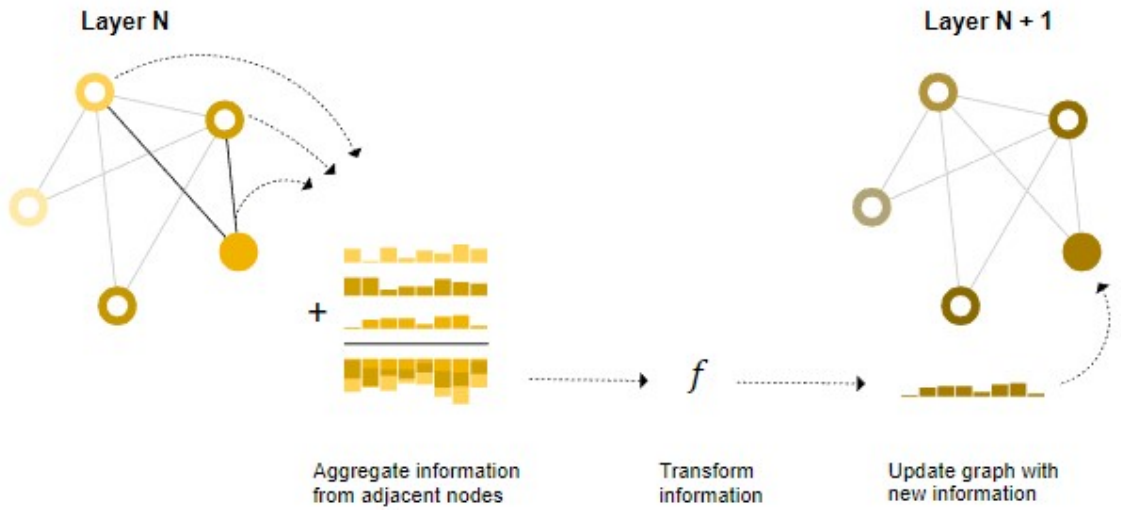

Figure 3.3: How do GNNs generate graph embeddings. Source: `https://distill.pub/2021/gnn-intro/`. Accessed: 15-06-2022.

Assuming that each node has an initial vector representation $h_i^{(0)} \in \mathbb{R}^{d_0}$, a Graph Neural Network layer has a set of node embeddings $\{h_i \in \mathbb{R}^d \mid i \in \mathcal{V}\}$ and a set of edges $\mathcal{E}$ as an input. The output of the layer is a new set of node representation $\{h'_i \in \mathbb{R}^{d'} \mid i \in \mathcal{V}\}$, where $\mathcal{N}_i = \{j \in \mathcal{V} \mid (j,i) \in \mathcal{E}\}$ is the neighborhood of i:

$$h'_i = f_\varphi(h_i, \text{AGGREGATE}(\{h_i \mid j \in \mathcal{N}_i\}))$$

where f_φ and AGGREGATE are what distinguish one GNN layer type from another. AGGREGATE is a permutationally invariant operation that aggregates messages from the local neighborhood of a node. Basic neighborhood aggregation operations are the **sum** or **average** of the neighbor embeddings.

Depending of how much we successively apply these layers to a graph we obtain different "states of information". For example, if we apply only one layer we have information combined from each node and its neighbors, or if we apply two layers we have information combined of a node, its neighbors and the neighbors of its neighbors.

3.3 Reinforcement Learning

Reinforcement Learning (RL) is a computational approach to learning from environmental interaction [SUT18]. The main objective of RL is to learn a policy that

defines the agent's behaviour at a given time. In our case, we will learn a planner's behavior to solve problems in a particular domain.

In RL we pursue the agent to follow the optimal path (states/actions) in a given environment so that the agent collects the maximal reward. Figure 3.4 shows an example in which the agent's objective is to take the diamond without falling into the trap (maximization of the total reward).

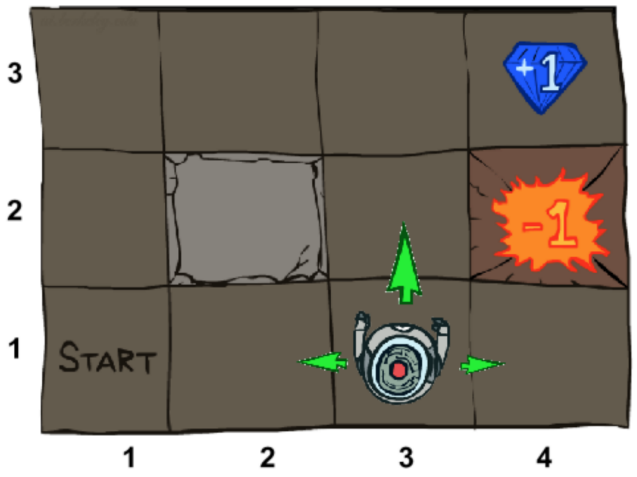

Figure 3.4: Typical example of a problem modeled with Reinforcement Learning in which an agent wants to obtain the maximum benefit. Source: `https://towardsdatascience.com/function-approximation-in-reinforcement-learning-85a4864d566`. Accessed: 22-08-2022.

RL algorithms are often modelled as finite Markov Decision Processes (MDP) [HEY90]. An MDP maps a current state to an action where the agent continuously interacts with the environment in order to produce new solutions and receive rewards. An MDP is defined as follows:

$$(S, A, R, P, T, \rho)$$

where S is a set of states, A is a set of actions, R is a reward function, P a transition probability function such as $P(s', s, a) = p(s'|s, a)$ and ρ is the distribution over initial states. In the example of Figure 3.4, a state could be, for example, a robot's position inside a cell of that grid. We can add more information to the state, for example, specifying the tools the robot is carrying.

At each time step t, an agent takes an action following a policy, represented by π, from its current state s_t:

$$\pi : S \rightarrow p(A = a_t \mid S = s_t)$$

Applying the policy to the state s_t it transitions to a newer state $\pi(s_t) = s_{t+1}$,

and receives a reward $r_t = R(s_t, a_t)$. In the example of Figure 3.4, the actions could be, for instance, the movement of the robot (up, down, left and right) and, if we use tools, the lighting of a flashlight.

When modeling a Reinforcement Learning problem, we need to define the reward function. In the example of Figure 3.4, the agent receives a positive reward, +1, if the robot reaches the diamond, a negative reward , -1, if it falls into the hole, and a neutral reward, 0, in any other case. By maximizing this reward function we would obtain that any path that starts where the robot is and reaches the diamond without passing through the hole would be optimal.

A more sophisticated reward function would return, for example, a higher reward as the robot gets closer and closer to the diamond using a distance metric. This would make the optimal path not just any path, but the shortest path to the diamond.

We can then define the **value of a policy** π, $J(\pi)$, as the expected value of the reward of all the possible paths τ induced by π within a task horizon T:

$$J(\pi) = \mathbb{E}_{\tau\sim\pi}\left[\sum_{t=0}^{T} r_t\right]$$

Thus, the learning problem is simply an optimization problem that seeks the policy with the best possible reward:

$$\pi^* = \arg\max_{\pi} J(\pi)$$

In spaces where the set of actions and states is reasonably small, the policy can be represented as a table where rows denote the discrete probability of actions that can be taken in a fixed state. However, in the case of large state and action spaces as it is the case we address in this work we are forced to represent the policy as some function approximator π_θ, with θ being the parameters for the approximation.

We can also define different value functions for policies, changing then the optimization problem. One very common approximation is, given $\gamma \in [0, 1]$:

$$J(\pi) = \mathbb{E}_{\tau\sim\pi}\left[\sum_{t=0}^{T} \gamma^t r_t\right] = \mathbb{E}_{\tau\sim\pi}\left[R(t)\right]$$

Which is the one that is used in this work and induces a decay for the last actions taken in a trajectory. The parameter γ is the discount factor and is used to determine the weight that the RL agent gives to rewards in the distant future versus those in the immediate future. If $\gamma = 0$ then the RL agent wil be completely myopic and will only learn actions that produce an immediate reward. The $R(t)$ function is called the *return to-go* function and represents the sum of each reward plus the discount factor.

3.4 Training a RL function approximator

Since the state space becomes too large and enumerating all the possible states or actions is unfeasible, a new approach based on the feature vector of each state is conceived. The aim is to use the features to generalize the value estimate of states that have similar features.

The above statement reflects we do not pursue the true value of a state but an estimation or approximation as this will allow us to achieve faster computations and a broader range of generalizations. This section is thus devoted to explaining how to compute these approximations, that is, function approximators.

In this work we will focus on stochastic policies. As we mentioned earlier, we assume that the problem can be modelled as an MDP, and therefore we do not have a clear action to take. Instead, stochastic policies offer a probability distribution over actions, and the act of sampling from that distribution is called *rolling the policy*.

For training a RL function approximator π_θ, policy gradient methods are used with Monte-Carlo sampling, which are methods that rely on randomness in order to solve generally deterministic problems, for the estimation of parameters θ. The gradient of the RL objective function is, according to [WIL92]:

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau\sim\pi}\left[\sum_{t=0}^{T} \nabla_\pi \log \pi_\theta(a_t|s_t) \sum_{t'=t}^{T} r(s_{t'}, a_{t'})\right] = \mathbb{E}_{\tau\sim\pi}\left[\sum_{t=0}^{T} \nabla_\pi \log \pi_\theta(a_t|s_t) R(t)\right]$$

Where $R(t)$ is the *return to-go* function. Nonetheless, this gradient is often estimated by choosing certain sampled trajectories D_k and computing the gradient of a *pseudo-loss* function with them:

$$L_{pseudo} = \frac{1}{|D_k|} \sum_{r\in D_k} \sum_{t=0}^{T} \log \pi_\theta(a_t|s_t) R(t)$$

The D_k trajectories are sampled at iteration k of the algorithm and each step the parameter θ is updated:

$$\theta_{k+1} = \theta_k + \alpha \nabla_\theta L_{pseudo}$$

This algorithm is highly inefficient because it needs to discard the data collected at each iteration and collect new data for the next update. In other words, the data used to update the policy must also be generated by the same policy parameters.

Other algorithms such as Proximal Policy Optimization (PPO) are used to better exploit the data collected at previous iterations of the learning process. PPO, in particular, performs several updates using the collected data before discarding it, thus allowing a better usage of resources.

This situation could lead to the algorithm being biased by the data at each iteration, so different strategies are used to avoid divergence between the current policy and the one that collected the data. More specifically, a special clipped objective is used in order to minimize this divergence. The algorithm described in [SCH17] describes the following optimization problem:

$$\theta_{k+1} = \arg\max_{\theta} \frac{1}{|D_k|} \sum_{r \in D_k} \sum_{t=0}^{T} M$$

$$M := \max \left\{ \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} A(a_t|s_t)^{\pi_{\theta_k}},\ clip\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)}, 1-\varepsilon, 1+\varepsilon \right) A(a_t|s_t)^{\pi_{\theta_k}} \right\}$$

Where *clip* is a function that prevents the value of the first parameter from exceeding the values of the interval compounded by the other two parameters, in this case $[1-\epsilon, 1+\epsilon]$, π_θ is the current policy that is being optimized, π_{θ_k} is the previous policy before updating, and $A(a_t|s_t)^{\pi_{\theta_k}}$ is the advantage of an action a_t given a state s_t inside a policy π_{θ_k}:

$$A(a_t|s_t)^{\pi_{\theta_k}} = R(t)^{\pi_{\theta_k}} - V_{\phi_k}(s)$$

Where $R(t)^{\pi_{\theta_k}}$ is the *return to-go* value, which depends on the actions taken by the policy, and $V_{\phi_k}(s)$ is a state value function predicted by some approximator with parameters ϕ_k, obtained by solving:

$$\phi_{k+1} = \arg\min_{\phi} \frac{1}{|D_k|} \sum_{r \in D_k} \sum_{t=0}^{T} |V_\phi(s_t) - R(s_t)|^2$$

In our work, the value of ϕ_{k+1} is extracted from the GNN.

Chapter 4

Generalized Planning model

In this chapter we will explain our approach to build a Generalized Planning model. The base structure that has been chosen for the generalized planning system is the one presented in the article [RIV20]. In the cited paper, Graph Neural Networks (GNNs) are used to represent the state of the problem by means of embeddings that come from the propagation of the problem information through the graph, via the layers of the GNN. This information is later used to create an adequate representation of the actions.

Once the data is processed, an MLP representing a Reinforcement Learning policy is trained, which will return a probability distribution over the actions. The final objective is, therefore, to develop a heuristic which, included in a greedy search algorithm combined with the RL policy, will return adequate results. In the following sections, we will present the components of the model architecture as well as explaining the information flow of the generalized planning system.

4.1 Acknowledgement

We got in contact with the authors of the named article requesting their implementation to do this work. They provided us with the implementation used for the experimentation shown in the paper, turning out that the code, which was written back in 2016, was completely outdated and much of it remained incompatible. This affected specially the GNN module as it used a proto-version of the Pytorch Geometric library [FEY19] which though it is a relatively new library, it has undergone many changes in the last few versions. The work within this chapter has therefore been divided into several parts:

- To understand how the code works and, with that, to migrate the code to the newer libraries and put together all the pieces of each block (GNN, DRL, PPO, etc.) so that they all work together.

- To understand and experiment with the provided architecture so as to analyze what type of strategies are learned and why.
- To modify the architecture by including new blocks that give rise to new alternatives and study the results.

4.2 Overview of the Generalized Planning model

The generalized planning model is based on a state space heuristic greedy search algorithm that solves planning problem instances. Given a domain D, and by solving various problems instances of D, we aim to learn a strategy or policy that solves "any problem" of the domain. We stress that we aim to train a model by solving instances of different size with the purpose of learning a model capable of solving instance of any number of objects and relationships between the objects.

The central component of the model architecture is thus a state-based search process combined with the computation of a RL module which is used to devise a heuristic function to guide the search process. Roughly speaking:

- A node of the search tree, which represents a planning state, is encoded as a graph and processed through a GNN.
- The info obtained from the GNN is fed to the RL module, which uses it to compute an approximate value for the state as well as a policy over the actions applicable in the state.
- With the policy and value functions returned by the RL, a heuristic function is designed to compute a heuristic value for each node during search.

4.3 State representation

As commented in section 4.2, our model builds a search tree for solving a planning problem of a particular domain, and nodes of the search tree represent a planning state. Particularly, a planning state is represented in the form of a graph, so that we use the perspective explained in Section 3.2 with the intention of encoding properties and relations between the objects in a proper way. The planning state comprised in a node of the tree is specifically represented as *state-goal* graphs, a graph that comprises both the state of the node as well as the goal of the problem.

A state-goal graph encodes the literals of the planning state of the node plus the literals that form the goal of the problem. Initially, for a particular problem (I, G), the initial state-goal graph will represent the literals of the initial state I and the literals of the goal G. For successor nodes, the graph will contain the state of the corresponding node and the goal G, which will be present in every state-goal

graph of the search tree. It is important to highlight not to mistake the nodes of the heuristic search tree with the nodes of the graph.

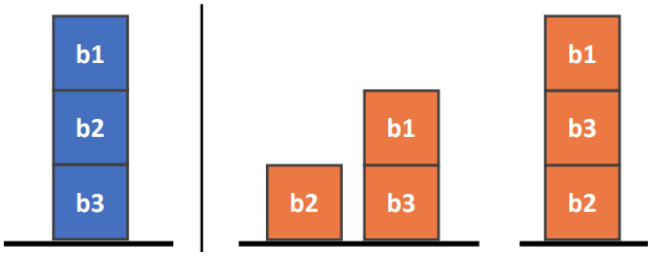

Figure 4.1: Example of a planning problem of the *blocksworld* domain. Source: [RIV20].

Let's take the planning problem of the *Blocksworld* domain in Figure 4.1. In this domain, the objects are blocks and a robotarm to handle the blocks. Given an initial configuration of the blocks on the table, the goal is to arrange the blocks as specified in the goal state. The problem of Figure 4.1 features three blocks `b1`, `b2` and `b3`. The initial state of the problem is shown on the left side (in blue), with `b2` being *on* b3 (`on b2 b3`) and `b1` *on* `b2` (`on b1 b2`). The goal of the problem is to have `b1` *on* `b3` (`on b1 b3`). The two configurations on the right (in orange) represent two possible final states that both achieve (`on b1 b3`).

We explain now how the graph is constructed. Let O be the set of objects of a planning instance, a graph will always have exactly $|O|$ nodes, one for each object of the planning instance, and two edges for each pair of objects connecting them. **The structure of the graph**, then, **remains invariant** through the search, what changes is the embedding representation of the features.

Moreover, relations are restricted to arity 2 because in a graph we can only relate two nodes. This is not a major issue as almost every relation with arity higher than 2 can be represented in terms of several lower arity relations. In the following, we will detail how to represent the state of a node in the planning search tree as a graph using the example shown in Figure 4.2.

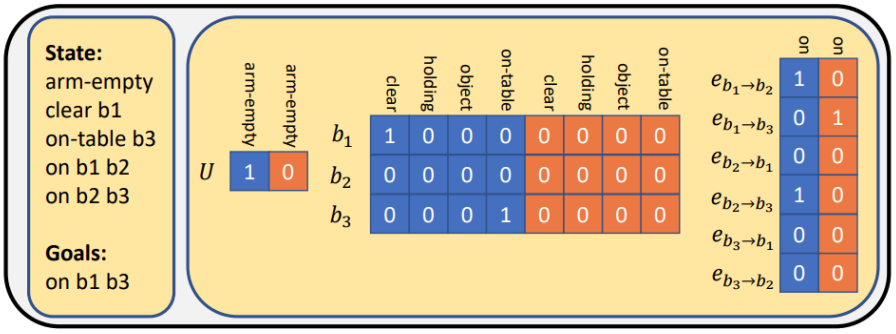

Figure 4.2: Embedding representation of the problem of Figure 4.1. Source: [RIV20].

A graph contains node (which we will call vertex from now on so it is not

mistaken with the search tree) features, edge features, and global features, which represent the 1, 2 and 0-arity predicates, respectively. The 0-arity predicates are global features of the problem, while the 1-arity ones represent properties of the objects and the 2-arity ones, the relations between them.

In the example of Figure 4.2 `clear` is a vertex feature, which is true (value 1) if there is nothing on top of that block; `on` is an edge feature, which represents if one block is on top of another; and `arm-empty` is a global feature representing whether the robot-arm that moves the blocks is empty or not.

We have already stated that for each state generated during the search process of solving a planning problem a graph is produced. The graph is **complete**, that is, each pair of graph nodes is connected.

As the network is a state-goal graph, we will first explain the construction of the "state" part of the graph, and then explain the "goal" one. We will base this explanation with the example of Figure 4.2, the state being the blue part and the goal being the orange part.

A *state* (graph) contains a vertex for each object defined in the problem. For each vertex, edge and global feature we have a one-hot encoding vector, with a value 1 if the corresponding feature is present in the current state of the problem, and 0 if it is not. For example, in Figure 4.2 we can see that the `clear` property of the vertex feature of `clear b1` is 1 because we have `clear b1` in the state, and as `on b1 b2` is on the state that we are modelling, the edge $e_{b_1 \to b_2}$ has a 1 in that part.

As far as the goal part is concerned, it is included by concatenating another one-hot encoding vector next to the current state one, but in this case the zero value indicates that the feature does not contribute to the goal. For example, in Figure 4.2 we can see that the only goal that is present, i.e., `on b1 b3`, is the only 1 present in the orange part of the encoding, in $e_{b_1 \to b_3}$.

A vertex in a state-goal network reflects the current state of the search node and where we want to go, so that the flow of information within the GNN will reflect these two realities: **where I am and where I want to go**. This information is essential when building a planner, since without information on where we want to go we would not be doing much more than a blind search. In Figure 4.3 we can see the explicit graph representation of the previous defined problem in *blocksworld* domain.

To sum up:

- Each tree state represents a step in the planning procedure using a graph, whose structure is fixed.
- Each tree state has information of the current situation and the goal stored in vertex, edge and global vectors.
- The graph produced is complete: every pair of graph nodes is connected, even if we do not have an explicit predicate that relates two nodes.

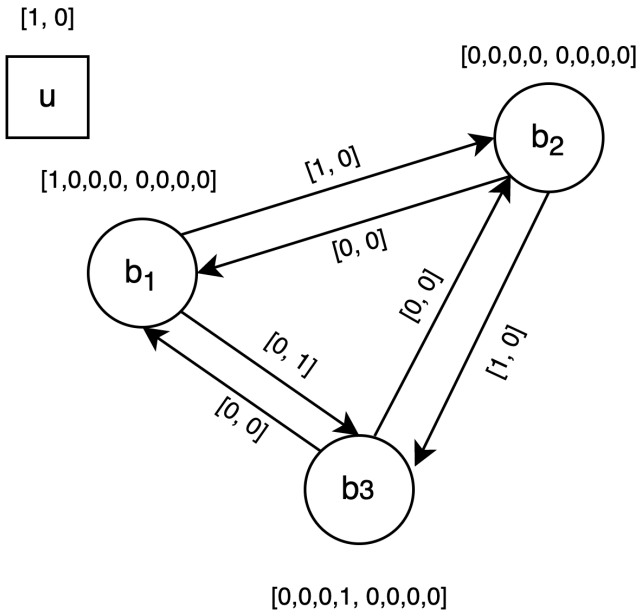

Figure 4.3: State-goal graph representation of the tree state that is seen in Figure 4.2.

4.4 GNN layers

Two different types of GNN layers are used in [RIV20] to propagate and produce embeddings reflecting the neighbors' information: the Graph Network block (GN block) and the Graph Network Attention Block (GNAT block). We have also created some other layers based on the attention one, called GNATv2 and GNATv3. All this GNN layers do not alter the structure of the graph, i.e. they convert a graph into another graph isomorphic to the input.

4.4.1 GN block

This block generates embeddings following the intuitive schema of Figure 4.4, in which we can observe that information is indiscriminately propagated through all the graph. The update of the graph features is performed in three phases: the edge embedding update, the vertex embedding update and finally the global embedding update.

Mathematically, the GN block layer applies the following operations to all the embeddings of the graph, which are based on the work of [BAT18] but simplified to a non multi-graph environment. Being e_{ij}, v_i, u the current edge, vertex and global embeddings, respectively, we start by updating the edge embedding with a Multilayer Perceptron, using its last embedding and the source vertex, as in Equation 4.1

$$\tilde{e}_{ij} = \phi(W^e[e_{ij}, v_i] + b^e) \tag{4.1}$$

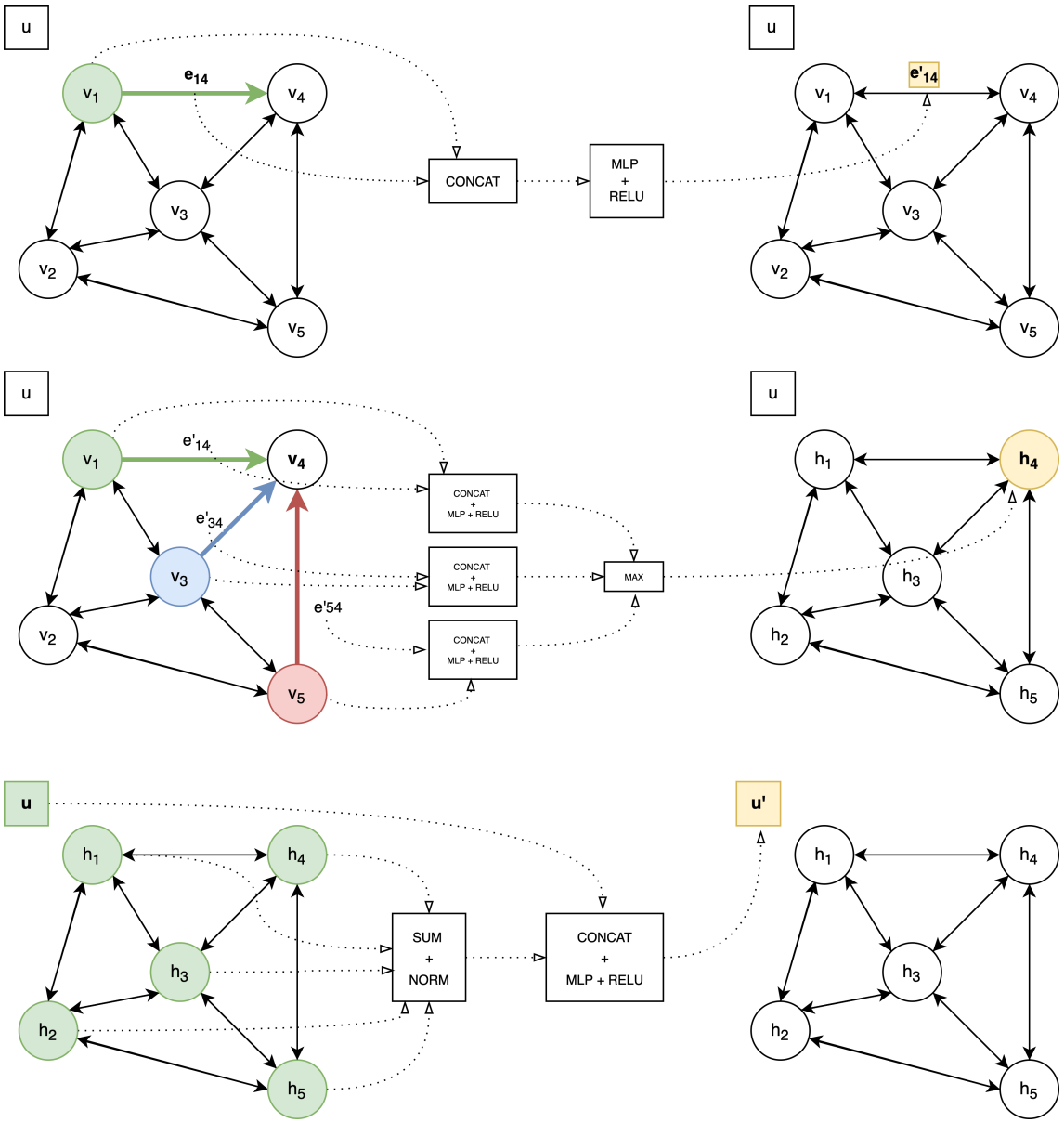

Figure 4.4: Propagation of the information via the GN block. It is divided in three phases: the edge embedding generation, the vertex embedding generation and the global embedding generation.

where W^e is the weight matrix, b^e is the bias and ϕ a RELU function.

Consecutively, focusing on vertex v_i in the vertex updating phase, we generate intermediate embeddings h_{ij} for each v_j in the neighborhood of v_i also with a MLP as in Equation 4.2

$$h_{ij} = \phi(W_1^v[v_j, \tilde{e}_{ji}] + b_1^v), v_j \in V_i \tag{4.2}$$

where W_1^v and b_1^v are also weights and biases, respectively. With this, we calculate the edge embedding by getting the maximum of the values in the neighborhood

(which is internally achieved with a node-wise max-pooling operator ψ) and then transforming it with the global features via another MLP, as in Equations 4.3 (W_2^v and b_2^v are weights and biases).

$$\begin{aligned} h_i &= \psi_j(h_{ij}) \\ \tilde{v}_i &= \phi(W_2^v[h_i, u] + b_2^v) \end{aligned} \tag{4.3}$$

Finally, the global embeddings are updated with information from the previous embeddings and an aggregation plus normalization of all the generated vertex embeddings, via also a MLP as in Equation 4.4, being W^u and b^u the last weights and biases.

$$\tilde{u} = \phi\left(W^u\left[u, \frac{1}{|V|}\sum_{i=1}^{|V|}\tilde{v}_i\right] + b^u\right) \tag{4.4}$$

This implementation differs from that of [BAT18] in some aspects that, in our opinion, are relevant:

- The edge embedding generation is only calculated using the sender vertex, not the receiver of the edge, and it does not use the global embedding for the update, unlike in [BAT18]. In the context of an edge e_{ij} the sender vertex is v_i and the receiver is v_j.
- The aggregation in the edge embedding generation section in [BAT18] is performed using edges and not vertexes.
- The global embedding generation in [BAT18] aggregates edges and vertexes, but we use only vertexes.

The GN block works well propagating general information amongst the graph, but makes it difficult to transfer specific information when needed. The transmission of information from all to all reminds us of an almost convolutional operation, in which every neighbor contributes to some extent to the updating of each other. This, as we have already said, could lead to certain problems for domains where specific information must be transmitted.

4.4.2 GNAT block

The GNAT layer is created by using the Graph Attention Network from [VEL17] but with an attention mechanism similar to the Transformer model of [VAS17], which is a renowned model in ML based on self-attention, i.e., attention on its own context. The only thing that differs from the GN block is the vertex embedding generation, which uses attention mechanisms in order to allow specific information to be transmitted, and it is what we are going to explain.

Mathematically, we use a key-query attention operation from the Transformer [VAS17] in order to get those embeddings. Firstly, we produce a transformation of the vertex embeddings using a MLP, as in Equation 4.5

$$h_i = \phi(W_1^v v_i + b_1^v) \tag{4.5}$$

with W_1^v and b_1^v being the learnable weights and biases from the MLP.

Secondly, we calculate the key and query of the attention, being respectively a transformation of the vertexes and edge embeddings (the new ones) via MLPs and its respective weights and biases. The operation can be followed in Equations 4.6.

$$\begin{aligned} k_i &= \phi(W^k v_i + b^k) \\ q_{ij} &= \phi(W^q \tilde{e}_{ij} + b^q) \end{aligned} \tag{4.6}$$

Then, we calculate an attention coefficient by performing a node-wise softmax function in the neighborhood of a vertex v_i and the respective keys and queries regarding to it, as in Equation 4.7.

$$\alpha_{ij} = \frac{e^{k_i^T q_{ij}}}{\sum_{p \in N(v_i)} e^{k_i^T q_{ip}}} \tag{4.7}$$

Following this we multiply the attention coefficient α_{ij} cell-wisely by the embedding that we want to attend to (the edge embedding), and then we aggregating it vertex-wise within the neighborhood again, as in Equation 4.8.

$$m_i = \varphi_j(\alpha_{ij} \cdot \tilde{e}_{ij}) \tag{4.8}$$

Lastly, the vertex embedding is then generated by transforming m_i, with the initial transformation of the vertex, h_i, and the global embedding, u, as shown in Equation 4.9.

$$\tilde{v}_i = \phi(W^u[h_i, m_i, u] + b^v) \tag{4.9}$$

The GNAT block allows important and concrete information to travel along the graph, and also focuses more in specific kinds of messages. The main difference with the [VEL17] paper is that here the attention does not use the LeakyReLU operation, but instead it uses the Transformer one, as it has already been mentioned.

It is also noteworthy that the α_{ij} coefficient is generated, unlike in the original paper, as the multiplication of the sender vertex and the edge, leaving out the receiver vertex. From a global perspective, this makes sense, since the vertex information should be updated with what is relevant, and in a 2-ary predicate the subject of the action is usually the first object or, in our case, the initial vertex.

4.4.3 GNATv2 block

The change in the attention mechanism made in the preceding section to the one described in the Transformer's paper made us wonder why the one in the original paper of [VEL17] had not been used to generate the coefficient α_{ij}. After some research into the different types of graph layers and relying on the Pytorch Geometric layer library [FEY19], we discovered thanks to the recent paper [BRO22] that the attention described in [VEL17] is, as they call it, static, and features some limitations.

Broadly speaking, understanding attention as a mechanism for computing a distribution within a set of key vectors given another vector which is called query, an attention mechanism is static if such mechanism always weighs one key greater than others, independently of the query. This is very limited because every function has a key that is always selected and thus we want to seek attention functions that are dynamic. That is, a function where each key could potentially excel depending on the query.

In [BRO22], it is proven that [VEL17] attention is static. A very slightly modification of that attention mechanism is provided, which makes it dynamic. The mechanism goes as follows: a learnable parameter a is introduced, which is then multiplied by a transformation of the sender and receiver vertexes concatenated with the edge, as in Equation 4.10.

$$\tilde{e}_{ij} = a^T \text{LeakyReLU}(W[v_i \mid\mid v_j \mid\mid e_{ij}]) \tag{4.10}$$

And then a softmax operation is introduced, followed by the aggregation plus multiplication which is typical in any attention operator. The result is seen in Equation 4.11

$$\begin{aligned} \alpha_{ij} &= \frac{\tilde{e}_{ij}}{\sum_{j' \in N(v_i)} \tilde{e}_{ij'}} \\ . \\ h_i &= \text{ReLU}\left(\sum_{v_j \in N(v_i))} \alpha_{ij} \cdot W v_j \right) \end{aligned} \tag{4.11}$$

Note that the only thing that distinguishes this implementation from the one exposed in [VEL17] is that now the a parameter is outside the LeakyReLU function, which is a very small change for such a big change in behavior.

4.4.4 GNATv3 block

Directly following this line, another layer called GNATv3 was also created, which uses the same attention mechanism but multi-headed. In multi-headed attention we perform several attention operations and then we concatenate them. For this last model we used the already existing layer *GATv2Conv* of Pytorch Geometric [FEY19], on top of which we implemented the multi-headed operation. This simply consists of

several attention operations running in parallel and then concatenated and linearly transformed to the desired dimension. We also found this interesting since the network could theoretically choose which attention to pick within each computation in parallel inside each head.

4.5 Action and policy representation

In this section we will explain how, from the embeddings that have been updated with the GNN layers, we can produce a representation of the actions in order to use it within the context of Reinforcement Learning. The goal is therefore to create a vector representation of the domain actions and then to use them.

There is a fundamental difference between common Reinforcement Learning approaches and the RL-based planning. Unlike ordinary RL benchmarks where the set of actions that can be taken from a state is fixed, in planning this set of actions is state-dependant and varies in size from one to another. This is not a problem, because we are following the approach of the policy as a function approximator.

Each action type receives a set of arguments, and for it to be applicable it must fulfil a set of preconditions. This means that, being A the set of actions, it can occur that $|\{a \in A \mid \pi(a|s) > 0\}| \neq |\{a \in A \mid \pi(a|s') > 0\}|$, i.e., given some states of the problem s and s', the number of applicable actions of one state can be different from another. They also have a set of effects that change the state, which can be positive, meaning that some information or evidence is added to the problem, or negative, meaning that some evidence is removed from the state because it no longer exists or fulfills anything.

In the [RIV20] approach they do not bother about preconditions, as they define a successor-state generator that automatically displays the current state's applicable actions. In practice this ends being the renowned STRIPS planner Pyperplan [ALK20]. In order to represent actions in a meaningful and useful way, another network is defined in order to generate which they call action embeddings.

In Figure 4.5 is illustrated how this process works. First of all, an action is defined as previously mentioned with its effects, represented by each gray block. For example, A_1 has three effects, A_2 has two and A_3 has four. The effects are then reordered and clustered by their type (and, consequently, their arity) and concatenated with a one-hot encoding vector of the corresponding dimension (for example, in Figure 4.5 there is only one global predicate, while there are three unary predicates and two 2-ary) with a 1 if the effect is positive and -1 if it is negative in the correspondent predicate. Each effect block is transformed by a *MLP* and then is scattered back to its original position, their aggregation being the final action embedding.

The policy is then defined as another *MLP* that outputs a scalar for each action, $\pi(a|s)$, normalized by a `softmax` operation. Moreover, the global embeddings of the

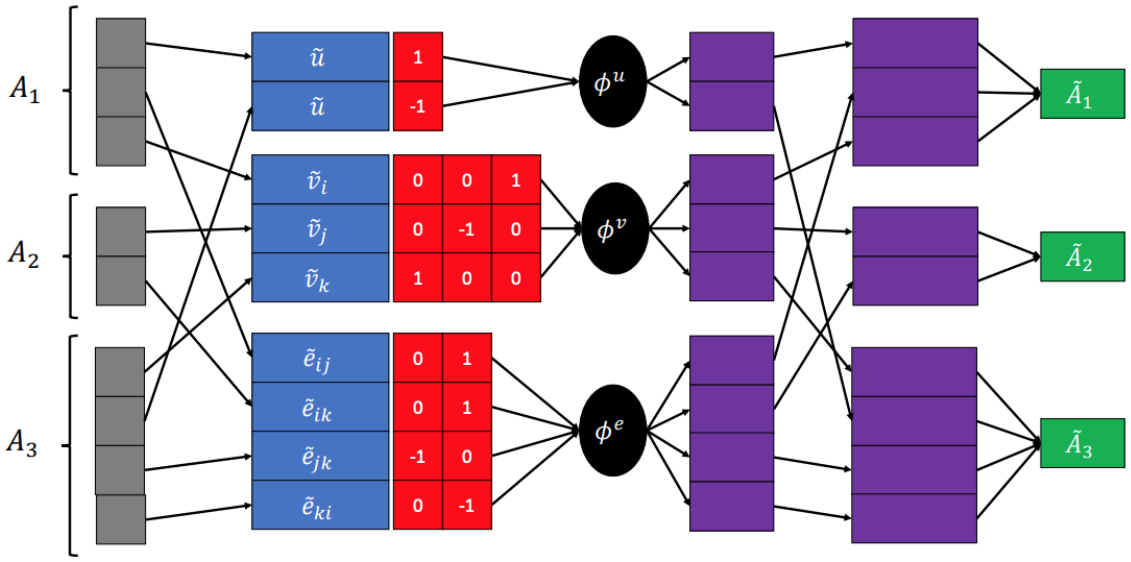

Figure 4.5: Action embedding generation. Source: [RIV20].

graph are taken in order to generate a "state importance" value with another *MLP*, $V(s)$, which is used in the RL algorithm as we already mentioned in the Background section. All in all, the structure follows the schema of Figure 4.6.

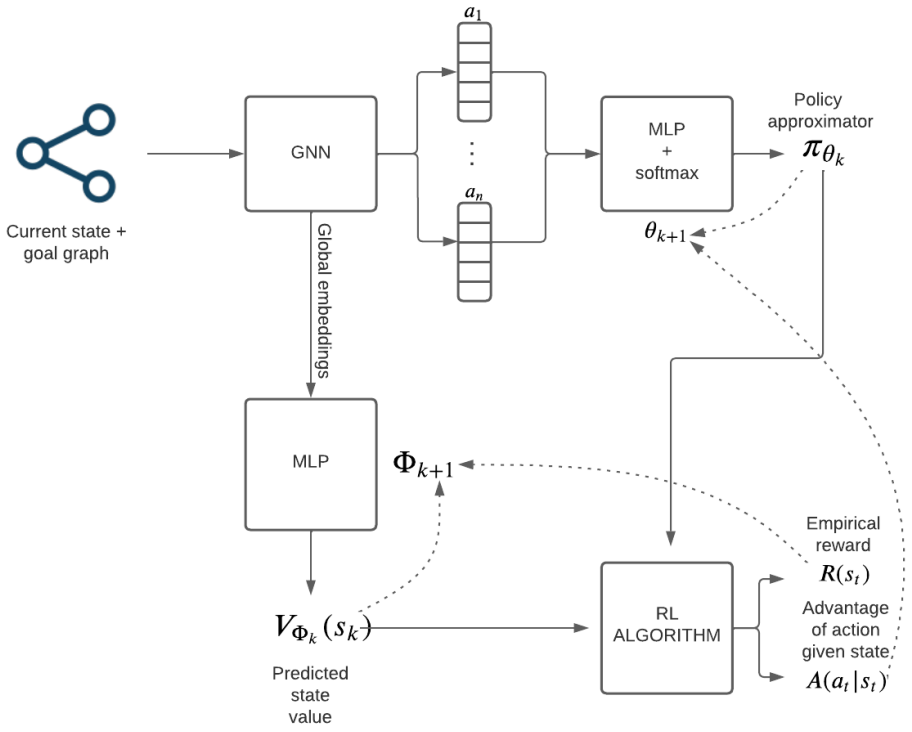

Figure 4.6: General schema of the structure of the system. Dashed arrows represent the training flow, while line arrows represent the update of each value.

4.6 Training procedure

The problem is modeled as a sparse reward problem, in which there are few states that return a feedback signal, with a binary reward: the RL algorithm will return a reward of 1 if the planning agent satisfies the goals of the problem within an horizon length, and if not it gets no reward at all. The appropriate horizon length is calculated using the *hff* heuristic [HOF01], which solves a relaxed version of the problem in linear time (removing all the negative effects). In order to get the horizon length, we use this value multiplied by a factor of 5.

The policy is then trained using Proximal Policy Optimization (PPO), and for it to be learned from scratch as a sparse binary reward problem it was only necessary to include problems with such small size that can be solved by a randomly initiated policy. This situation allows the policy to train itself and to eventually solve much larger instances without the need of manual tuning. In the training procedure, the policy is rolled out until termination before updating the model parameters, then using empirical return values instead of simulated ones and helping with this the stabilization of the algorithm. Rolling out a policy implies, without changing it, finding a sequence of operations that lead to a state in which we cannot apply any action or to a final state. In this work, they use 100 roll out episodes and the collected data to update the model.

4.7 Planning with a learnable heuristic

It is usually useful to combine policies with a search algorithm to improve planning systems, as in [SIL19], for example. Several elements of the structure of Figure 4.6 are useful in order to generate a heuristic for some search algorithm. In [RIV20] they base their search on a slightly modified Greedy Best First Search algorithm [FRA19].

Instead of constructing a search tree from the root node (initial state) and expanding a node with the best heuristic estimation and so on, they use the parameters of the RL algorithm in order to compute a heuristic value for each node, performing then a full roll-out for each expanded node as we already mentioned in last section. If we represent each node as a pair of state and action, the heuristic estimation is defined as follows:

$$g(s,a) = \frac{\pi_\theta(a|s) \cdot V_\Phi(s)}{1 + H(\pi_\theta(\cdot|s))}$$

With $\pi_\theta(a|s)$ being the probability of an action a from a state s given by our trainable policy, $V_\Phi(s)$ being the estimated state value coming from the GNN, which is also trained with the RL algorithm as stated in Figure 4.6, and $H(\pi(\cdot|s))$ being the entropy of the policy's distribution over actions at a given state s.

Chapter 5

Experimentation

The purpose of this chapter is to observe what we can explain or infer with the proposed generalized planning architecture and what aspects are more complicated or even impossible to generalize. To do this, we have created experiments with a number of domains that are well known in the planning literature, built a taxonomy within them, and identified the type of GNN that works best for each domain, correlating GNN layers with the domain taxonomy.

5.1 Domains

We will work with some of the most common classical planning domains that feature predicates of arity no greater than two. The domains have been extracted from the IPC (International Planning Competition)[1].

The objective is to test domains that feature different characteristics in order to check if general strategies can or cannot be learned for such domains. We will test how well the GNN models are able to generalize strategies and in which cases they are best learned, depending on the complexity and nature of the domain.

5.1.1 Blocksworld

In this domain there is a potentially infinite smooth surface, something like a table, and a set of blocks identified by letters. There is also a mechanical arm capable of picking up a block and dropping it in another position, either on top of another block or on the table, and only one block can be moved at a time. The objective is to change the arrangement of blocks using the arm, as in Figure 5.1. The actions that can be taken at each step are:

- `Pickup`. The arm picks up a block that is clear and on the table, i.e. that has

[1] https://www.cs.cmu.edu/afs/cs/project/jair/pub/volume20/long03a-html/node1.html

nothing on top of it.

- `Putdown`. The arm leaves the block that is holding on top of the table.
- `Stack`. The arm leaves the block that is holding on top of another block.
- `Unstack`. The arm picks up a block that is clear and on top of another block.

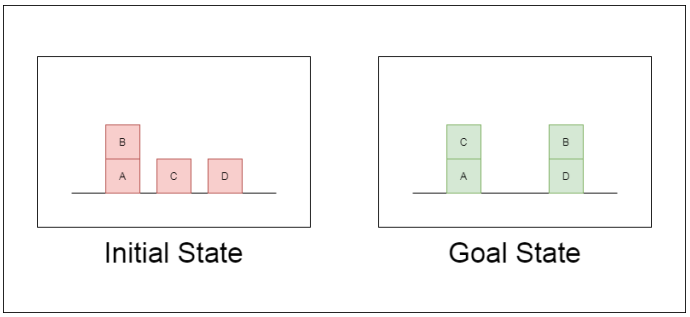

Figure 5.1: Example of a problem in the blocksworld domain.

5.1.2 Satellite

This domain models several satellites that collect data from different targets. Each satellite has different (probably overlapping) instruments that support several imaging modes. Instruments need calibration to take a picture. A problem goal consists of capturing images from certain positions, also called targets, in a specific mode (spectrometer, infrared, etc.). The instruments need energy to be able to take a picture in any mode. A satellite will only be able to power one type of instrument at a time. The available actions are:

- `Turn-to`. A satellite changes its direction.
- `Switch-on`. An instrument inside a satellite is switched on with the power of the satellite, but it is not initially calibrated.
- `Switch-off`. An instrument inside a satellite is switched off, freeing then the power of the satellite.
- `Calibrate`. The instrument is calibrated for a certain target in order to be able to take a photography. Not every instrument can be calibrated for every target.
- `Take-image`. Once the instrument is calibrated, it can take a picture of the target, but only with one of the modes that the instrument can work with.

5.1.3 Gripper

This domain involves a robot with two grippers that must move balls between two rooms. The *Gripper* domain is similar to the *blocksworld* domain, but in this case the robot can move between rooms and no structure is needed for the goal, as the balls are just dropped to the floor. A gripper can only grasp one ball at a time, so the robot can transport up to two balls between rooms in one move, one ball in each gripper. The robot has freedom of movement between rooms. The actions are:

- `Move`. The robot moves from one room to another.
- `Pick`. The robot picks up a ball with one of its grippers, only if it is available.
- `Drop`. The robot drops the ball in a room from one of its grippers, freeing it immediately.

5.1.4 Ferry

Similar to the *Gripper* domain, but in this case we have a ferry that must transport cars between potentially more than two different locations, instead of a robot. The ferry can transport only one vehicle at a time. The actions of the domain are:

- `Sail`. The ferry moves from one location to another.
- `Board`. The ferry boards a car. In this scenario, a ferry can only have one car per sail.
- `Debark`. The ferry unloads a car in a location and becomes empty.

5.1.5 Logistics

Logistics is a domain in which we have different objects placed in certain locations, that must be transported to other locations by land or air. Ground transportation uses trucks and can only happen between locations that are within the same city, while air transportation is between airports, which are special locations inside cities. The destination of a package is a location either within the same city or in a different city. In general, ground transportation is required to take a package to the city's airport (if the package is not at the airport), then air transportation between cities, and finally ground transportation again to take the package to the final destination, if it is not the same airport. The actions of the *Logistics* domain are:

- `Load-truck`. This action loads an object inside a truck, both of them being at the same location.

- `Load-airplane`. The object, in this case, is loaded inside an airplane that is located in an airport.
- `Unload-truck`. Unloads an object inside a truck in the location where the truck is.
- `Unload-airplane`. Unloads an object inside an airplane in the airport where the plane is.
- `Drive-truck`. A truck moves between two locations inside the same city.
- `Fly-airplane`. An airplane moves between two airports from different cities.

5.1.6 Depots

This domain consists of trucks for loading and unloading crates, and hoists to handle the crates in pallets. Hoists are only available at certain locations and they are static. Crates can be stacked/unstacked onto a fixed set of pallets. Trucks do not store the boxes in any particular order[2]. This domain can be seen as an hybrid between *logistics* and *blocksworld*. The actions are:

- `Drive`. A truck drives between two places.
- `Lift`. A hoist lifts a crate which is on a surface, simmilar to the *blocksworld* domain.
- `Drop`. A hoist drops the crate onto a surface.
- `Load`. Some crate lifted by a hoist is loaded into a truck at a certain place.
- `Unload`. Some crate that is into a truck is unloaded by a hoist at a certain place.

5.1.7 Elevators

In this domain we have a building with a specific number of floors, people who want to move between these same floors and a series of elevators spread throughout the building. In addition, the elevators are in blocks, which means that they only have access to a certain number of floors. In general, the building is divided according to the blocks, which are contiguous and coinciding only at the ends, with the exception of a fast block where there is a fast elevator that can go throughout the building with the restriction of only being able to stop at the even numbered floors. In this approach elevators do not have limited capacity, so they can transport as many people as needed. The actions of the domain are the following:

[2] `https://www.cs.cmu.edu/afs/cs/project/jair/pub/volume20/long03a-html/node38.html`. Accessed: 25-06-2022.

- `Passenger-enters`. Some passenger enters some elevator at a certain floor.
- `Passenger-leaves`. Some passenger leaves some elevator at a certain floor.
- `Move-elevator`. Some elevator moves from one floor to another.

5.1.8 Hanoi

This domain represents the well-known game of the Hanoi towers, in which we have three sticks and a series of disks of different sizes, stacked starting with the largest and ending with the smallest on the left stick. The objective, in this case, is fixed: to transport the stack of discs to the right stick, with the particularity that the moves that are allowed are those that do not involve putting a larger disc on top of a smaller one. The actions are:

- `Move`. Some disc moves on top to another which is bigger or directly to a empty stick.

5.2 Taxonomy

Our objective is to experiment with the different types of domains defined above and observe how the different GNN architectures behave when learning valid policies. As it has been made clear in the previous section, domains present specific characteristics that allow us to divide them into different groups. We can analyze several aspects:

- **Goal dependency**: Whether goals are dependent from each other or not, i.e., if a goal can be reached without interacting with any other goal.
- **Agency**: It describes the number of agents involved in the domain. A *single* agency means that there is only one agent that executes or produces the plan; *multi* agency means that there is more than one executing agent.
- **Typology**: A concept from [TOR15], a domain is *loosely-coupled* if there is little or no interaction between the agents in order to reach the set of goals, and is *tightly-coupled* if there is a lot of interaction and/or dependence between agents in order to fulfill a goal. It only applies, obviously, for multi-agent domains.

Once we have described the aspects that we want to focus on, we can make Table 5.1, which identifies each domain with each aspect.

The *blocksworld* domain is single-agent because there is only one arm that moves blocks, and it is goal dependent because if a goal is block A on top of block

Domain	Goal dependency	Agency	Typology
Blocksworld	Yes	Single	-
Satellite	No	Multi	Loosely-coupled
Gripper	No	Single	-
Ferry	No	Single	-
Logistics	No	Multi	Tightly-coupled
Depots	Yes	Multi	Tightly-coupled
Elevators	No	Multi	Tightly-coupled
Hanoi	Yes	Single	-

Table 5.1: Taxonomy of each domain.

B there is more to take into account, as block B could need to be on top of another block.

Logistics is not goal dependent, because one package delivery is independent from another, but has a multi agency because it involves trucks and airplanes, which are also tightly coupled as they need to collaborate in order to get the package to the destination.

The goals in the *satellite* domain are independent because they only involve pictures of certain entities, and is multi-agent because there is more than one satellite. Moreover, it is also loosely-coupled as each satellite can get the job done independently; in fact, one satellite could even take all the pictures if it had the required equipment.

The *gripper* and the *ferry* domains are both goal independent and single agent, because they are both transport domains with one entity (a gripper and a ferry) and they just leave entities at each location without any specific order.

Depots is goal dependent because it involves some piling mechanisms as in *blocksworld* domain, but is multi-agent and tightly-coupled as there exist several trucks and grippers that need to coordinate.

The *elevators* domain is not goal dependent because each person wants to go to a specific floor and that does not interact with any other person's desire. Moreover, it is also in a multi-agent environment, as there are various elevators, and tightly-coupled, as they all have to collaborate in order to reach the goal state. If a passenger wanted to potentially go from one floor to another, he/she would have to make as many transfers as there are elevators, or take the express elevator if he/she can.

The *hanoi* domain is goal dependent since the position of each disk interferes with that of the others, similar to what happens in the *blocksworld* domain. Moreover, it is single-agent since the movements are all made from the same virtual structure that we could think of as a hand or a gripper.

5.3 Training

For the experiments we have used the architecture presented in previous chapters. It has been decided to use, based on [RIV20] experiments, a two-layer system in which the second layer is always a GN block and the first layer is varied between the different layers of attention that have been developed. In this way we have the following approximations:

1. **GN block + GN block**
2. **GNAT block + GN block**
3. **GNATv2 block + GN block**
4. **GNATv3 block + GN block**

Since Reinforcement Learning based methods require large amounts of information in order to be trained, we have made use of a series of PDDL generators available at [SEI22] in order to fulfil this task, and some original Python programmed generators such as the one for the *elevators* domain.

The policies are trained with 1000 iterations, with an evaluation frequency of 15 and each iteration with a number of 100 training episodes. We used a learning rate of 10^{-4}, a discount factor for the RL return to-go function of 0.99 and a clipping ratio ε of 0.2. The models are validated with 100 PDDL problems that are fixed at the start of the procedure and automatically generated.

The experiments have been performed on a machine with a Nvidia GeForce RTX 3090 GPU, a 12th Gen Intel(R) Core(TM) i9-12900KF CPU and Ubuntu 22.04 LTS operating system. In Table 5.2 we display the sizes used for training and validation of the RL algorithm.

The success rate has been used as a control measure, i.e., the ratio of whether the policy is able to reach the goal from the initial state. Results from validation are shown in Table 5.3.

Due to the large number of experiments carried out, with each experiment taking several days to complete, it was impossible to generate a more exhaustive error analysis, with confidence intervals, for the success rate due to the limited amount of time and resources available. In future works within this field we would like to carry out a more complex experimentation, including these confidence intervals.

We see no significant differences between the last two proposed architectures, GNATv2-GN and GNATv3-GN, and the original attention one in the paper, GNAT-GN, beyond slightly improving performance in certain models where other models had already achieved a better result. There is a clear difference between domains that perform better for architectures with non-attention layers and domains that perform better for architectures with attention layers. This is what we want to

Domain	Train size	Validation size
Blocksworld	4-5 blocks	10-11 blocks
Satellite	1-4 satellites 1-4 instruments 1-4 modes 1-4 targets 2-4 observations	7-8 satellites 6-7 instruments 4-5 modes 8-9 targets 15-16 observations
Gripper	3-4 balls	15-16 balls
Ferry	3-5 locations 2-4 cars	9-10 locations 20-21 cars
Logistics	2-4 airplanes 2-4 cities 2-4 trucks 2-4 locations per city 1-3 packages	3-4 airplanes 6-7 cities 6-7 trucks 3-4 locations per city 6-7 packages
Depots	1-2 depots 2-3 distributors 2-3 trucks 3-5 pallets 2-4 hoists 3-5 crates	5-6 depots 5-6 distributors 5-6 trucks 5-6 pallets 5-6 hoists 5-6 crates
Elevators	6-10 floors 2-3 passengers 1-2 blocks	8-12 floors 5-10 passengers 1-3 blocks
Hanoi	3-4 discs	5-7 discs

Table 5.2: Sizes used for each domain in training and validation.

Domain	GN-GN	GNAT-GN	GNATv2-GN	GNATv3-GN
blocksworld	**100%**	69%	72%	76%
satellite	57%	**100%**	51%	97%
gripper	**100%**	**100%**	**100%**	**100%**
ferry	92%	**100%**	90%	**100%**
logistics	4%	**28%**	3%	5%
depots	16%	**54%**	17%	30%
elevators	62%	53%	**63%**	61%
hanoi	0%	0%	0%	0%

Table 5.3: Success rate in validation. We highlight the best result for each domain in bold.

discuss in the next section of this paper. In fact, we can come to the following conclusions by focusing on the taxonomy we have created:

- Loosely-coupled or single-agency domains in which the goals are independent, such as *satellite*, *gripper* or *ferry*, achieve better results with attention layers, because they need a more focused flow of information.

- A goal-dependent domain such as *blocksworld* achieves better results with the GN-GN model. In this case the indiscriminate exchange of information that characterizes this model benefits the results because of the nature of the domain.
- Tightly-coupled domains such as *logistics*, *depots* or *elevators* do not reach 100% in success rate, probably because the two GNN layers are insufficient to be able to correctly transfer the information, since in addition to a need for attention to the flow of information, the order of execution of the actions also needs to be taken into account.
- Domains such as *hanoi* fail to achieve any degree of generalization.

Chapter 6

Learned policy analysis

The purpose of this chapter is to analyze the learned policy for each domain presented in the previous chapter, if any, and look into the difficulties to learn it. We will discuss the characteristics that make some domains be easier than others at the time of learning a generalized policy.

6.1 Overview

In the previous chapter we trained a variety of models (GN-GN, GNAT-GN, GNATv2-GN and GNATv3-GN) for every domain, each resulting in different performance. Subsequently, we validated the models to get a first indication on the behavior of each trained model on slightly larger problem sizes. In this chapter, we will discuss, domain by domain and using the model that output the highest rate in validation, the reasons for such performance values. To do so, we tested the best model with even larger size problems, and we analyzed the plans (policy) obtained for specific problems.

After training the four models for each domain, what we will do in the testing phase is to use the policy that emerged from the best model to find a plan for various large-size problems. We are interested in analyzing whether the learned policy is able to reach generalization, that is, we want to observe if the learned policy also solves problems of a larger size.

Whether or not the policy succeeds in reaching this generalization, we will use specific problems from each domain that will help us understand and justify the reasons why or why not this generalization is reached.

Additionally, we will compare the results of the best model for each domain with a state-of-the-art planner: Fast Downward [HEL11]. We carried out this comparison in terms of computation time, to see how fast our approach is in comparison to FD, and in terms of expanded states, to see how much "greedy" our algorithm behaves, i.e., how big the search is in terms of memory, using 100 different problems generated

from the distribution shown in Table 6.1.

In Table 6.1 we can see the size of the problems that have been chosen to work with in the testing phase within each domain. It can be seen that some domains, the simplest ones such as *blocksworld* or *gripper*, allow us to test with very large sizes, while in other more complicated domains such as *logistics*, *depots* or *elevators* we need to work with problems of smaller size since the spatial complexity of the problem increases greatly as we increase the number of objects within the problem. In Table 6.1 we can observe the differences in size between train and test.

Domain	**Train size**	**Test size**
Blocksworld	4-5 blocks	5-100 blocks
Satellite	1-4 satellites 1-4 instruments 1-4 modes 1-4 targets 2-4 observations	11-17 satellites 12-13 instruments 6-7 modes 12-17 targets 47-60 observations
Gripper	3-4 balls	5-200 balls
Ferry	3-5 locations 2-4 cars	15-20 locations 20-50 cars
Logistics	2-4 airplanes 2-4 cities 2-4 trucks 2-4 locations per city 1-3 packages	3-4 airplanes 5-6 cities 5-6 trucks 3-4 locations per city 5-6 packages
Depots	1-2 depots 2-3 distributors 2-3 trucks 3-5 pallets 2-4 hoists 3-5 crates	5-6 depots 5-6 distributors 5-6 trucks 7-9 pallets 7-9 hoists 7-9 crates
Elevators	6-10 floors 2-3 passengers 1-2 blocks	8-10 floors 5-8 passengers 1-2 blocks
Hanoi	3-4 discs	-

Table 6.1: Sizes of problems used for each domain in testing.

6.2 Blocksworld

In this section we will analyze the testing results for the *blocksworld* domain when using the configuration that returned the best results in validation; i.e., the GN-GN configuration.

First, we will take a look at the plans that the algorithm generates as a result of learning the policy, and examine them in order to see if we are able to find out what it is really learning. We will take as an example the *blocksworld* problem shown in the excerpt Problem 1.

As we can see, the code in Problem 1 features a situation with 10 blocks, whose initial configuration is given by the facts in the initial situation (:init). The goal is to achieve two towers: `b8` on top of `b9` on top of `b1` on top of `b2` on top of `b3` on top of `b4`; and a second tower `b6` on top of `b10` on top of `b7`.

Analyzing the plan in Problem 1 we can see that the algorithm tends to first put all the blocks on the table to then stack them as specified in the goal. Surprisingly, the network has managed to learn a policy valid for any instance of this domain, one that is well known in the planning world: **put all the blocks on the table and subsequently stack them in the correct order indicated in the goal**. Obviously this will not be the optimal strategy in most scenarios, but the semantic knowledge that the network has learned is what makes this interpretation rich, since by using this policy the algorithm would potentially be able to solve any scenario.

A more exhaustive experimentation has been carried out in the Figure 6.1, serving also as a comparison with the Fast Downward planner and using, as mentioned above, 100 problems automatically generated from a previously defined probability distribution. We observe that we have two graphs: one corresponding to time and the other to expanded states. We can see in the figure the success rate of our model in testing phase (GN block - GN block) against Fast Downward in *blocksworld* in terms of time and memory consumed for problems of 5-100 blocks.

The left graph shows how many of these problems have been solved at each time; i.e., a success rate of 0.8 at 60s means that 80% of the problems have been solved (the goals have been reached) within 60 seconds. The right graph shows how many states were needed in order to solve each problem, i.e., a success rate of 0.8 at 100 expanded states means that 80% of the problems have been solved without visiting more than 100 states.

It can also be observed in the Figure 6.1 that our approach greatly improves in terms of the number of expanded states. This means that the information it learns is very focused, in the sense that the algorithm acts approximating as a greedy heuristic in which we would reach the solution by simply going forward in the search. This is quite interesting because it makes us think that the heuristic is quite informed and sufficient.

Problem 1 Plan of a problem for the *blocksworld* domain using the GN-GN model. The learned policy performs: put every block on the table and then stack them in the correct order accordingly to the goal.

```
(define (problem BW) (:domain blocksworld)
(:objects b1 b2 b3 b4 b5 b6 b7 b8 b9 b10 )

(:init                                  (clear b8))
(arm-empty)                             (:goal
(on b1 b2)                              (and
(on b2 b3)                              (on b1 b2)
(on b3 b4)                              (on b2 b3)
(on b4 b9)                              (on b3 b4)
(on-table b5)                           (on b5 b8)
(on-table b6)                           (on b6 b10)
(on b7 b6)                              (on b8 b9)
(on b8 b10)                             (on b9 b1)
(on b9 b7)                              (on b10 b7))))
(on b10 b5)
(clear b1)

PLAN:

1.  (unstack b8 b10)                    18. (putdown b7)
2.  (putdown b8)                        19. (pickup b3)
3.  (unstack b10 b5)                    20. (stack b3 b4)
4.  (stack b10 b1)                      21. (pickup b2)
5.  (unstack b10 b1)                    22. (stack b2 b3)
6.  (putdown b10)                       23. (pickup b1)
7.  (unstack b1 b2)                     24. (stack b1 b2)
8.  (putdown b1)                        25. (pickup b10)
9.  (unstack b2 b3)                     26. (stack b10 b7)
10. (putdown b2)                        27. (pickup b6)
11. (unstack b3 b4)                     28. (stack b6 b10)
12. (putdown b3)                        29. (pickup b9)
13. (unstack b4 b9)                     30. (stack b9 b1)
14. (putdown b4)                        31. (pickup b8)
15. (unstack b9 b7)                     32. (stack b8 b9)
16. (putdown b9)                        33. (pickup b5)
17. (unstack b7 b6)                     34. (stack b5 b8)
```

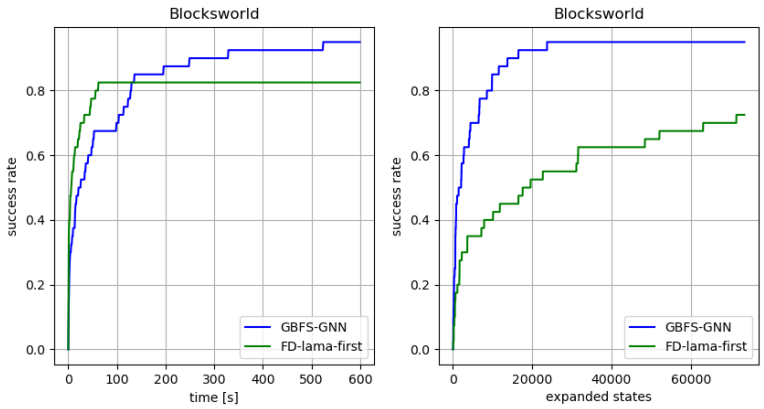

Figure 6.1: Success rate of our model in testing phase against Fast Downward in *blocksworld* domain.

We can conclude the following remarks from the *blocksworld* domain:

- Our approach was able to learn a policy that solves any size problem, thus guaranteeing the generality of the policy.
- As there is only one robot arm in this domain, problem solving completely relies on a sequential decision-making. This favors the learning of a policy for this domain.
- The most interesting aspect lies in that the strategy correctly deduces the order of achieving the goals once they are all independently achievable. That is, when all the goals `(on X Y)` are individually achievable, the algorithm is able to infer the correct order for stacking the blocks. This fact raises a question as whether this is a learnable strategy in all the domains.

6.3 Satellite

In this section we will discuss the testing results for the *satellite* domain, using the configuration that returned the best results in validation; that is, the GNAT-GN architecture. We will also discuss the *satellite* problem example seen in Problem 2 and its generated plan, and we will try to infer the general policy from it.

The problem defined in Problem 2 refers to a situation in which we have two satellites, `sat0` and `sat1`, with two instruments, `ins0` and `ins1`. The instrument `ins0` is inside `sat0` and `ins1` is inside `sat1`. There are two modes of taking pictures, infrared `infra0` and spectometer `spc1`, and the goal is to take six pictures of six different objects in the sky: `phn2`, `Star3`, `plt4`, `plt5`, `phn6` and `Star7` with any of the two modes. We note that `sat0` would be able to do all the work.

Taking a look to the plan displayed in Problem 2, we can see that the algorithm assigns all the work to one single satellite, since it is capable of taking all the pictures (as it can work in both necessary modes). What our model tends to do here is, as expected, to converge towards the most generalized strategy possible: **a satellite is chosen and does all the imaging work itself**. Note that if one satellite is not capable of taking all the pictures, **the least amount of needed satellites will be used**.

This strategy generalizes very well, as seen in Figure 6.2, in which we see the success rate of our model (GNAT-GN) in the testing phase against Fast Downward in terms of time and memory consumed, for 11-17 satellites, 12-13 instruments per satellite, 6-7 modes, 12-17 targets and 47-60 observations. Our approach outperforms again Fast Downward in terms of expanded states, reaching a 100% success rate. The use of the attention layer allows specific information to travel along the graph, which in this case is the use of only one or few satellites.

We decided to also run the GN-GN model to see the differences of the learned policy. The plan shown in Extract 1 is generated with the GN-GN model. The program starts an infinite cycle of turning the satellites: it does not have the ability to focus the work to the few satellites that can perform all the tasks. We can see that the `turn_to` action repeats infinitely until manually termination of the inferring process. Below we can see the same problem but with the plan inferred by using the GNAT-GN model, which finally finds a solution.

We can conclude the following from the experiments in the *satellite* domain:

- Our approach (GNAT-GN) was able to learn a policy that solves problems of any size, thus guaranteeing generalization.
- In this case the use of an attention layer helps the model focus on a single satellite, as long as it is able to accomplish all the goals. If not, several satellites are chosen, always as few as possible.
- When it is compulsory that two or more satellites take the pictures, the non-interacting nature of the goals allows the model to “attend” to one satellite at

a time, that is to say, a satellite taking a picture does not influence another taking another picture by any means.

- In this domain, the goals are all individually achievable and so there is no a specific order for the pictures to be taken, so the strategy does not need to worry about the order.

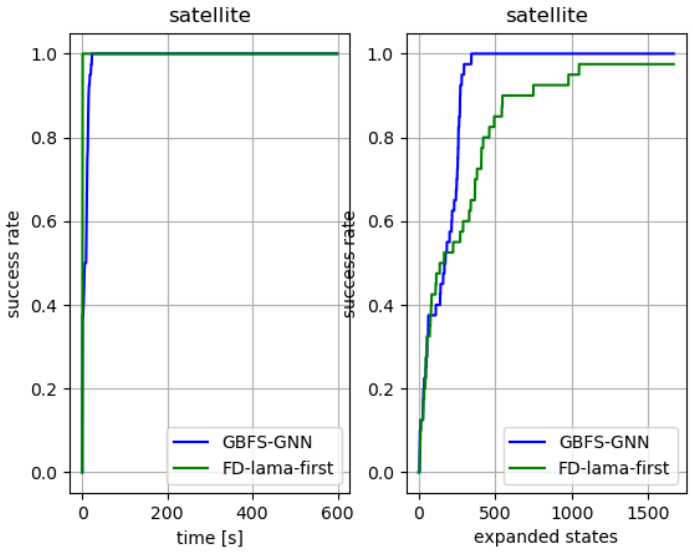

Figure 6.2: Success rate of our model in testing phase against Fast Downward in *satellite* domain.

Problem 2 Example of a plan in the *satellite* domain using the GNAT-GN model. We see that what the policy learns in this case is letting a single satellite, usually the first one, do all the tasks. In case a single satellite cannot take every image, it tries to use the least amount of satellites possible.

```
(define (problem sat)
(:domain satellite)
(:objects
sat0 - satellite
ins0 - instrument
sat1 - satellite
ins1 - instrument
infra0 - mode
spc1 - mode
Star1 - direction
grstt0 - direction
phn2 - direction
Star3 - direction
plt4 - direction
plt5 - direction
phn6 - direction
Star7 - direction)
```

```
(:init
(supports ins0 infra0)
(supports ins0 spc1)
(calibration_target ins0 Star1)
(on_board ins0 sat0)
(power_avail sat0)
(pointing sat0 grstt0)
(supports ins1 spc1)
(calibration_target ins1 grstt0)
(on_board ins1 sat1)
(power_avail sat1)
(pointing sat1 phn2))

(:goal (and
(have_image phn2 spc1)
(have_image Star3 infra0)
(have_image plt4 infra0)
(have_image plt5 spc1)
(have_image phn6 spc1)
(have_image Star7 spc1) )))
```

```
PLAN:

1.  (switch_on ins0 sat0)
2.  (turn_to sat0 star1 grstt0)
3.  (calibrate sat0 ins0 star1)
4.  (turn_to sat0 plt5 star1)
5.  (take_img sat0 plt5 ins0 spc1)
6.  (turn_to sat0 phn2 plt5)
7.  (take_img sat0 phn2 ins0 spc1)
8.  (turn_to sat0 plt4 phn2)
9.  (turn_to sat0 star7 plt4)
10.  (take_img sat0 star7 ins0 spc1)
11.  (turn_to sat0 phn6 star7)
12.  (take_img sat0 phn6 ins0 spc1)
13.  (turn_to sat0 star3 phn6)
14.  (take_img sat0 star3 ins0 infra0)
15.  (turn_to sat0 plt4 star3)
16.  (take_img sat0 plt4 ins0 infra0)
```

Extract 1 Extract of a plan for some *satellite* problem using the GN-GN model versus the GNAT-GN one.

Extract from the GN-GN model:

```
(switch_on ins35 sat7)              (turn_to sat7 pnt15 grnd5)
(turn_to sat7 grnd6 star2)          (turn_to sat6 star8 pnt18)
(turn_to sat7 star0 grnd6)          (switch_on ins6 sat1)
(turn_to sat7 star7 star0)          (turn_to sat6 grnd3 star8)
(turn_to sat7 star4 star7)          (turn_to sat7 pnt17 pnt15)
(turn_to sat7 grnd5 star4)          (turn_to sat1 pnt17 pnt9)
(calibrate sat7 ins35 grnd5)        (turn_to sat7 pnt11 pnt17)
(turn_to sat7 pnt21 grnd5)          (turn_to sat1 star4 pnt17)
(take_img sat7 pnt21 ins35 spc)     (turn_to sat7 star0 pnt11)
(turn_to sat7 pnt16 pnt21)          (turn_to sat1 star2 star4)
(turn_to sat7 pnt12 pnt16)          (turn_to sat6 pnt22 grnd3)
(switch_on ins30 sat6)              (turn_to sat7 phn13 star0)
(turn_to sat7 pnt18 pnt12)          (turn_to sat6 pnt15 pnt22)
(turn_to sat7 grnd5 pnt18)          (turn_to sat1 star14 star2)
(switch_on ins22 sat5)              (turn_to sat6 star2 pnt15)
(turn_to sat5 pnt21 pnt22)          [...]
```

Extract from the GNAT-GN model (note that this is the full plan):

```
(switch_on ins36 sat7)              (turn_to sat7 pnt23 grnd5)
(calibrate sat7 ins36 star2)        (switch_on ins30 sat6)
(turn_to sat7 phn13 star2)          (turn_to sat6 star4 pnt18)
(take_img sat7 phn13 ins36 infra0)  (calibrate sat6 ins30 star4)
(turn_to sat7 pnt18 phn13)          (turn_to sat6 pnt17 star4)
(take_img sat7 pnt18 ins36 infra0)  (take_img sat6 pnt17 ins30 img2)
(turn_to sat7 star14 pnt18)         (turn_to sat6 star19 pnt17)
(take_img sat7 star14 ins36 infra0) (take_img sat6 star19 ins30 img2)
(turn_to sat7 pnt9 star14)          (turn_to sat6 pnt16 star19)
(take_img sat7 pnt9 ins36 infra0)   (take_img sat6 pnt16 ins30 img2)
(turn_to sat7 grnd5 pnt9)           (turn_to sat6 pnt22 pnt16)
(switch_on ins26 sat5)              (take_img sat6 pnt22 ins30 img3)
(turn_to sat5 grnd3 pnt22)          (turn_to sat6 pnt10 pnt22)
(calibrate sat5 ins26 grnd3)        (take_img sat6 pnt10 ins30 img2)
(turn_to sat5 phn20 grnd3)          (turn_to sat6 pnt15 pnt10)
(take_img sat5 phn20 ins26 infra1)  (take_img sat6 pnt15 ins30 img2)
(turn_to sat5 pnt11 phn20)          (turn_to sat6 pnt12 pnt15)
(take_img sat5 pnt11 ins26 infra1)  (turn_to sat6 pnt21 pnt12)
(turn_to sat5 pnt23 pnt11)          (take_img sat6 pnt21 ins30 spc4)
(take_img sat5 pnt23 ins26 infra1)
```

6.4 Gripper

In this section we will analyze the behavior of the GN-GN model for the *gripper* domain. In fact, all the tested models returned a 100% success rate for this domain. The reason for the high success across all modes may be due to the very nature of this domain, because the only thing that can be done is to transport balls from left to right, which leads to a quite simple strategy.

We want to examine a plan for a relatively big problem in the *gripper* domain, which is shown in the excerpt Problem 3. The problem is very simple: we have five balls that we want to move from one room (`rooma`) to another (`roomb`), using a robot with two grippers.

As we can see in the plan, the algorithm learns that the best strategy is that the robot **grabs two balls from the initial room, each with one gripper, transport them to the other room, drop them and go back to the first room**. This situation is repeated until there are no more balls.

Note that in Problem 3 the plan is not optimal because actions 4 and 5 are unnecessary. This may be happening because, in the end, what the algorithm does is to sample actions inside a probability distribution induced by the policy that it has learned, perhaps sampling an action that should not have been executed at that time but that had a probability value greater than zero.

In that sense, when we are in `roomb` we might also want to go to `rooma` (if we had no ball in our grippers), even though in that state we do have two balls in our grippers. The probability of taking that action is, then, not zero, so it could potentially be chosen – which is what happens in this scenario. That is inefficient because we first want to drop the two balls before going back again to `rooma`.

The inference of the aforementioned strategy leads to a very satisfactory generalization, as seen in Figure 6.3. The architecture GN-GN is sufficient to learn a generalized policy. We can analyze the success rate of our GN-GN model in the testing phase against Fast Downward in terms of time and memory consumed, for problems of 5-200 balls.

The success rate is almost identical to that of FD as it is a simple domain for a planner, although we observe that our approach improves success rate in terms of expanded states against the Fast Downward planner. It is then shown that it is an easy planning domain that any planner can easily solve and which leads to a very "repetitive" policy.

Problem 3 Example of the policy learned in the gripper domain for the GN-GN model

```
(define (problem gripper)
(:domain gripper)
(:objects  rooma roomb left right ball1 ball2 ball3 ball4 ball5 )

(:init                                  (at ball2 rooma)
(room rooma)                            (at ball3 rooma)
(room roomb)                            (at ball4 rooma)
(gripper left)                          (at ball5 rooma)
(gripper right)                         (at-robby rooma))
(ball ball1)
(ball ball2)                            (:goal
(ball ball3)                            (and
(ball ball4)                            (at ball1 roomb)
(ball ball5)                            (at ball2 roomb)
(free left)                             (at ball3 roomb)
(free right)                            (at ball4 roomb)
(at ball1 rooma)                        (at ball5 roomb))))

PLAN:
                                        9.  (pick ball3 rooma right)
1.  (pick ball4 rooma right)            10. (pick ball1 rooma left)
2.  (pick ball5 rooma left)             11. (move rooma roomb)
3.  (move rooma roomb)                  12. (drop ball1 roomb left)
4.  (move roomb rooma)                  13. (drop ball3 roomb right)
5.  (move rooma roomb)                  14. (move roomb rooma)
6.  (drop ball4 roomb right)            15. (pick ball2 rooma left)
7.  (drop ball5 roomb left)             16. (move rooma roomb)
8.  (move roomb rooma)                  17. (drop ball2 roomb left)
```

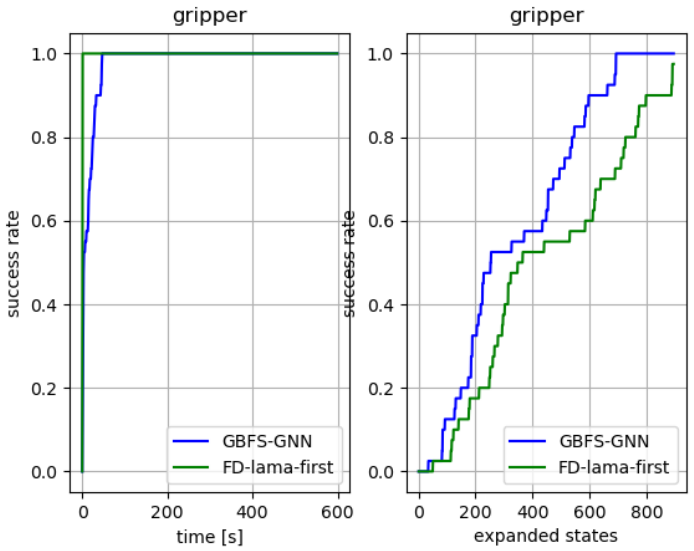

Figure 6.3: Success rate of our model in testing phase against Fast Downward in *gripper* domain.

We can conclude the following from the experiments in the *gripper* domain:

- Generalization is guaranteed by our approach, which was again able to learn a policy that potentially solves any problem instance.
- This domain has only one robot, so we are again in a sequential decision-making process, as in the *blocksworld* domain. This helps the learning of a policy.
- It is interesting to note the appearance of small cycles that are potentially useless, favored by the stochastic nature of our policy.
- The goals are all individually achievable from the initial state of the problem and no order is needed, as in the *satellite* domain.

6.5 Ferry

We will now analyze the results obtained from the *ferry* domain using the GNATv3-GN model. Although the architecture GNAT-GN also obtained a 100% success rate in validation, we opted for testing GNATv3-GN as this is the only domain along with the *gripper* domain that scored the maximal success rate in GNATv3-GN.

The *ferry* domain is similar though slightly more elaborated than the *gripper* domain. In the *ferry* there is also only one transport vehicle, a ship, that is used to transport objects (cars, in this case) that need to be moved to a city, and the ship

can only transport one car at a time. Unlike the *gripper* domain, there are more than two locations wherein the ferry can move to.

The PDDL code in Problem 4 shows a problem and a plan for this domain. The problem is an scenario of four locations: `l0`, `l1`, `l2` and `l3`, and six cars, `c0`, `c1`, `c2`, `c3`, `c4` and `c5`, that are initially in one location and have to be transported to another one. The plan in Problem 4 shows the movements of the ferry to transport the cars from one location to another.

The strategy of the *ferry* domain is that **the ship goes to one location, picks up a car and leaves it at its destination; then, if there is a car to transport at the destination city, it picks it up and takes it to its goal, and so on. If there is no car, the ferry will move to another location that has a car to be transported**. Here we can also see that the policy sometimes makes unnecessary trips similar as the robot in *gripper*, but in this case as there are more locations, instead of going back to the departure room it goes to an intermediate place.

There is an inherent serialization in the problem since there is only one ship, but in this case the location to which this ship can travel is not unique as in the *gripper* domain. This is why we believe that the attention layers generally favors the ship to focus the journey to a specific place, reducing cycles and therefore increasing success rate, as we can see in Extract 2, in which we compare the plan extracted from the GN-GN model versus the one from GNATv3-GN model. We can observe that the `sail` actions define a more straightforward trip with less moving around with the attention model. In both scenarios a solution is achieved, but with the second model using much less steps.

Figure 6.4 compares the success rate of our model in testing phase (GNATv3 block - GN block) against Fast Downward in the *ferry* domain, in terms of time and memory consumed, for problems of 15-20 cities and 20-50 vehicles. We note that our approach equals the Fast Downward planner in terms of time, and greatly improves it in terms of memory.

The conclusions of the analysis for this domain are as follows:

- A strategy is learned through the attention algorithm, achieving the desired results for the test instances.
- In the plans we observe that the algorithm often fails to learn direct trips from one location to another, which may be due to having learned transitions from other problems that may not be useful in the specific case.
- The attention layer favors this search to be narrowed down, reaching solutions in up to one third of the steps than with non-attention domains. For this reason, in addition to achieving a slight increase in the success rate to 100% from the GN-GN model to the GNATv3-GN one, the model has also learned a better policy in terms of plan length.

Problem 4 Plan for a *ferry* problem using the GNATv3-GN model. We can see that the strategy that is followed here is that the ferry goes visiting each city, debarking and boarding cars whenever is needed. Unnecessary trips are marked with a *.

```
(define (problem ferry)
(:domain ferry)
(:objects l0 l1 l2 l3 c0 c1 c2 c3 c4 c5 )

(:init
(location l0)                    (at c3 l2)
(location l1)                    (at c4 l2)
(location l2)                    (at c5 l3)
(location l3)                    (at-ferry l1))
(car c0)
(car c1)                         (:goal
(car c2)                         (and
(car c3)                         (at c0 l1)
(car c4)                         (at c1 l1)
(car c5)                         (at c2 l0)
(empty-ferry)                    (at c3 l1)
(at c0 l2)                       (at c4 l3)
(at c1 l2)                       (at c5 l2))))
(at c2 l0)


PLAN:

1.  (sail l1 l2)                 12. (board c5 l3)
2.  (board c0 l2)                13. (sail l3 l2)
3.  (sail l2 l3) *               14. (debark c5 l2)
4.  (sail l3 l1)                 15. (board c4 l2)
5.  (debark c0 l1)               16. (sail l2 l3)
6.  (sail l1 l2)                 17. (debark c4 l3)
7.  (board c3 l2)                18. (sail l3 l2)
8.  (sail l2 l3) *               19. (board c1 l2)
9.  (sail l3 l1)                 20. (sail l2 l0) *
10. (debark c3 l1)               21. (sail l0 l1)
11. (sail l1 l3)                 22. (debark c1 l1)
```

Extract 2 Extract of a plan for some *ferry* problem using the GN-GN model versus using the GNATv3-GN model.

Extract from the GN-GN model:

```
[...]                (debark c3 l4)        (sail l1 l6)
(debark c16 l5)      (board c1 l4)         (sail l6 l9)
(sail l5 l8)         [...]                 (sail l9 l8)
(sail l8 l1)         (debark c10 l3)       (sail l8 l6)
(board c3 l1)        (sail l3 l7)          (sail l6 l3)
(sail l1 l5)         (sail l7 l2)          (sail l3 l7)
(sail l5 l7)         (sail l2 l1)          (sail l7 l2)
(sail l7 l1)         (board c5 l1)         (sail l2 l8)
(sail l1 l5)         (sail l1 l9)          (sail l8 l7)
(sail l5 l9)         (sail l9 l0)          (sail l7 l3)
(sail l9 l0)         (sail l0 l1)          (sail l3 l1)
(sail l0 l3)         (sail l1 l6)          (sail l1 l6)
(sail l3 l2)         (sail l6 l0)          (sail l6 l5)
(sail l2 l8)         (sail l0 l1)          (debark c5 l5)
(sail l8 l3)         (sail l1 l7)          (sail l5 l3)
(sail l3 l7)         (sail l7 l4)          (sail l3 l6)
(sail l7 l6)         (sail l4 l0)          (sail l6 l2)
(sail l6 l1)         (sail l0 l3)          (sail l2 l4)
(sail l1 l5)         (sail l3 l8)          [...]
(sail l5 l4)         (sail l8 l1)
```

Extract from the GNATv3-GN model (fewer sail actions):

```
(sail l5 l4)         (sail l1 l4)          (board c3 l1)
(board c14 l4)       (sail l4 l5)          (sail l1 l4)
[...]                (debark c16 l5)       (debark c3 l4)
(debark c4 l3)       (sail l5 l4)          (board c9 l4)
(board c13 l3)       (board c19 l4)        (sail l4 l1)
(sail l3 l4)         (sail l4 l8)          (sail l1 l5)
(sail l4 l2)         (debark c19 l8)       (sail l5 l1)
(debark c13 l2)      (board c0 l8)         (sail l1 l9)
(board c16 l2)       (sail l8 l0)          (debark c9 l9)
(sail l2 l4)         [...]                 (sail l9 l7)
(sail l4 l8)         (debark c10 l3)       (sail l7 l1)
(sail l8 l9)         (sail l3 l1)          (board c5 l1)
(sail l9 l4)         (board c12 l1)        (sail l1 l7)
(sail l4 l7)         (sail l1 l5)          (sail l7 l5)
(sail l7 l8)         (debark c12 l5)       [...]
(sail l8 l1)         (sail l5 l1)          (debark c15 l1)
```

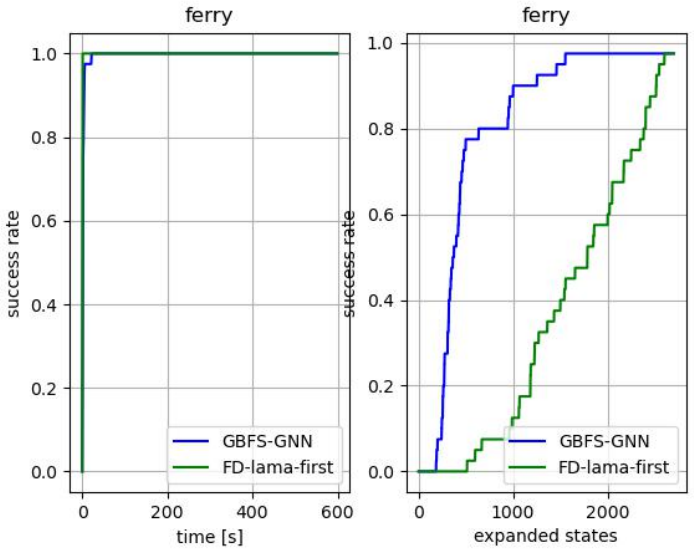

Figure 6.4: Success rate of our model in testing phase against Fast Downward in *ferry* domain.

6.6 Logistics

According to table 5.3, for the *logistics* domain, we found that all four approaches failed to achieve an adequate success rates in validation. In this section, we aim to deeply analyze why we got these results.

To begin with, tightly-coupled domains, and especially one such as *logistics*, are known for having sequences of actions that can be parallelized, i.e., that could theoretically be executed in parallel. This is the case, for instance, of several packages being transported simultaneously. Given that the generalized planning model is for sequential decision making, policies define a total ordering of the actions.

In the situation where these sequences are independent from one other the order can be arbitrary. For example, when a truck moves a package from one location to another within the same city, and another truck wants delivers a different package. In this case, no matter the order of the two independent sub-sequences of actions.

The problem comes when these sequences have to reach a point where they must be executed in a specific order. For example, a truck moves to a location within a city at the same time as an airplane moves from one airport to another. However, if the goal is to take one package from one city to another city, the truck and the airplane must coordinate (interact) at some point. That is, the parallel sequences have to be "sequentialized" at the moment when a package is transferred from a truck to an airplane.

This is where we see the problem: the necessary overlapping that happens, for example, when we need to take a package on an airplane, is what makes the network unable to generalize satisfactorily. Since this is a perspective based on

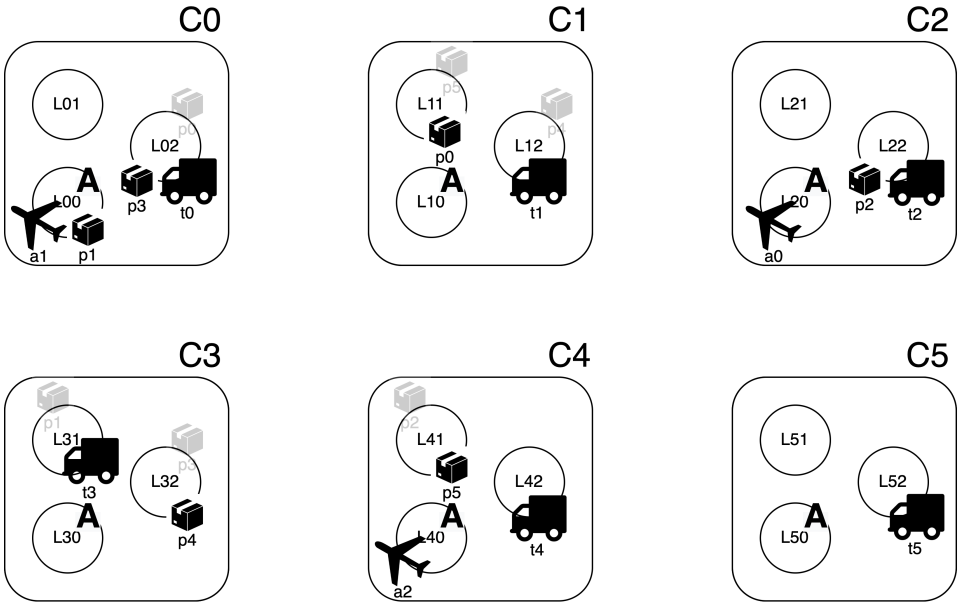

Figure 6.5: Problem of *logistics* domain. Grey packages are the goal, Ci represent each city and Lij each location inside the city. A represents whether a location is an airport.

Deep Reinforcement Learning and with the way the graph is defined with which we represent the problem, it is impossible to obtain any valuable information about the correct order of the plan, in the training with the current architecture.

An example of this can be seen with the problem described in Figure 6.5, in which we have six cities: `c0`, `c1`, `c2`, `c3`, `c4` and `c5`, that have an airport with which they communicate and have several trucks, one for each city, that transport packages between locations from within the city itself. In Problem 5 we have the plan extracted from the GNAT-GN model (the one that returned the highest success rate in validation), and the aforementioned problem comes to light in the `drive-truck` action, which remains in an infinite loop until termination.

Since the architecture prevents us from establishing a partial order in the actions as mentioned above, a policy is learned that overly rewards the actions of moving the trucks, which makes sense since it is the first thing to be done in any *logistics* plan and what would make an airplane able to pick up the package to move it out of town.

As a result of our experience with the GNAT-GN architecture, we decided to experiment with a new model with the aim to correct this problem. An important aspect to be taken into account is the number of layers used in the GNN models. Given that we are working with complete networks, the information flows from all to all, so we must be careful not to increase the number of layers too much since this could produce an "overgeneralization" effect, uniforming all the embeddings and thus losing specific information.

Because of this it seemed reasonable to think of an architecture with two layers

Problem 5 Plan for the logistics problem seen in Figure 6.5 using the GNAT-GN model. Here we can see that a lot of cycles in driving start to appear, hence preventing the planner from converging.

```
1.  (load-truck p2 t2 l22)          22. (drive-truck t3 l31 l32 c3)
2.  (load-truck p3 t0 l02)          23. (drive-truck t1 l11 l12 c1)
3.  (drive-truck t0 l02 l00 c0)     24. (drive-truck t3 l32 l31 c3)
4.  (load-truck p1 t0 l00)          25. (drive-truck t1 l12 l11 c1)
5.  (drive-truck t3 l31 l32 c3)     26. (drive-truck t3 l31 l32 c3)
6.  (load-truck p4 t3 l32)          27. (drive-truck t1 l11 l12 c1)
7.  (drive-truck t3 l32 l31 c3)     [...] (Repeated a lot of times)
8.  (drive-truck t1 l12 l11 c1)     (drive-truck t3 l31 l32 c3)
9.  (load-truck p0 t1 l11)          (drive-truck t1 l11 l12 c1)
10. (drive-truck t3 l31 l32 c3)     (drive-truck t3 l32 l31 c3)
11. (drive-truck t1 l11 l12 c1)     (drive-truck t1 l12 l11 c1)
12. (drive-truck t3 l32 l31 c3)     (drive-truck t3 l31 l32 c3)
13. (drive-truck t1 l12 l11 c1)     (drive-truck t1 l11 l12 c1)
14. (drive-truck t3 l31 l32 c3)     (drive-truck t3 l32 l31 c3)
15. (drive-truck t1 l11 l12 c1)     (drive-truck t1 l12 l11 c1)
16. (drive-truck t3 l32 l31 c3)     (drive-truck t3 l31 l32 c3)
17. (drive-truck t1 l12 l11 c1)     (drive-truck t1 l11 l12 c1)
18. (drive-truck t3 l31 l32 c3)     (drive-truck t3 l32 l31 c3)
19. (drive-truck t1 l11 l12 c1)     (drive-truck t1 l12 l11 c1)
20. (drive-truck t3 l32 l31 c3)     (drive-truck t3 l31 l32 c3)
21. (drive-truck t1 l12 l11 c1)     [...]
```

of attention instead of one. We have already seen that loosely-coupled domains such as *satellite* make very good use of this single layer of attention in order to be able to focus the information where it is wanted, but we have seen that this is not sufficient in tightly-coupled domains where there is a lot of overlapping between agents. Therefore, we introduced another layer that intuitively focuses on attending to this overlap, while the other one focuses more on the information flow such as in *satellite*.

With this, we define a new GNN structure in which we have two GNAT layers followed by a GN, and we performed the same validation experiments as in the previous chapter. The previous best model was the GNAT-GN model with a **28%** success rate, reaching **31%** with the new model structure of GNAT-GNAT-GN.

As can be seen, although the results have improved slightly, they are still far from what we are theoretically looking for. We cannot thus speak of a generalized policy in this case – or at least we have not been able to find such a policy with the proposed models. This makes sense since the *logistics* domain is a rather complex domain in which the search for such a policy (if it existed) is an arduous task. In fact, [STA22] paper shows that it is not possible to learn a policy for the *logistics* domain.

However, we can talk about why we believe the new proposal has improved the outcome. As we have said, adding a new layer to the model helps to make the information more targeted and learning somewhat more satisfying. For example, in Extract 3 we can see how the inclusion of this new layer tends to ignore intermediate paths for the benefit of more directed ones. As we have said, this is not enough but it might explain the slight improvement since in the end this significantly shortens the search within the state space.

In the left column from Extract 3 we see an extract of the plan from the GNAT-GNAT-GN model , while in the second one we see an extract for the same problem but using the GNAT-GN model. Note that on the left side the `drive-truck` and `fly-airplane` actions go directly to the destination, while on the right side it usually makes a stop in some other location. There are some examples with *.

It was then decided to perform some tests for this new model GNAT-GNAT-GN, observing how it behaves with respect to the Fast Downward planner as has been done on previous occasions. In Figure 6.6 we can see the experiments that have been carried out, with the particularity that it has not been possible to further increase the size of the problems due to the limitations in terms of memory that the problem has, since the logistics domain is a domain with a considerable size already for relatively small problems. We note that our model improves on Fast Downward in the results obtained.

We analyzed the success rate of our model (GNAT block - GNAT block - GN block) against Fast Downward in *logistics* domain, in terms of time and memory consumed, for problems of 3-4 airplanes, 5-6 cities, 5-6 trucks, 3-4 locations per city and 5-6 packages.

Extract 3 Extract of a plan for some *logistics* problem using the GNAT-GNAT-GN model versus using the GNAT-GN model.

```
GNAT-GNAT-GN extract:

(load-truck p1 t4 l41)
(drive-truck t4 l41 l42 c4)*
(unload-truck p1 t4 l42)
(load-truck p3 t4 l42)
(load-truck p1 t4 l42)
(load-truck p2 t4 l42)
(drive-truck t3 l31 l33 c3)*
(load-truck p5 t3 l33)
(drive-truck t1 l12 l10 c1)
(drive-truck t5 l52 l51 c5)
(load-truck p4 t5 l51)
(drive-truck t5 l51 l53 c5)
(unload-truck p4 t5 l53)
(load-truck p4 t5 l53)
(drive-truck t3 l33 l32 c3)
(unload-truck p5 t3 l32)
(drive-truck t5 l53 l50 c5)
(drive-truck t4 l42 l40 c4)
(unload-truck p3 t4 l40)
(load-airplane p3 a1 l40)
(unload-truck p1 t4 l40)
(load-airplane p1 a1 l40)
(load-airplane p0 a1 l40)
(fly-airplane a1 l40 l10)
[...]
```

```
GNAT-GN extract:

(load-truck p1 t4 l41)
(drive-truck t4 l41 l40 c4)*
(drive-truck t4 l40 l42 c4)*
(load-truck p3 t4 l42)
(load-truck p2 t4 l42)
(drive-truck t4 l42 l40 c4)*
(load-truck p0 t4 l40)
(drive-truck t4 l40 l43 c4)*
(drive-truck t1 l12 l10 c1)
(drive-truck t5 l52 l51 c5)
(load-truck p4 t5 l51)
(drive-truck t5 l51 l53 c5)
(drive-truck t3 l31 l33 c3)
(load-truck p5 t3 l33)
(drive-truck t3 l33 l32 c3)
(unload-truck p5 t3 l32)
(drive-truck t5 l53 l50 c5)
(drive-truck t4 l43 l40 c4)
(fly-airplane a2 l50 l20)
(unload-truck p2 t4 l40)
(load-airplane p2 a1 l40)
(fly-airplane a1 l40 l10)*
(fly-airplane a1 l10 l00)*
(unload-airplane p2 a1 l00)
[...]
```

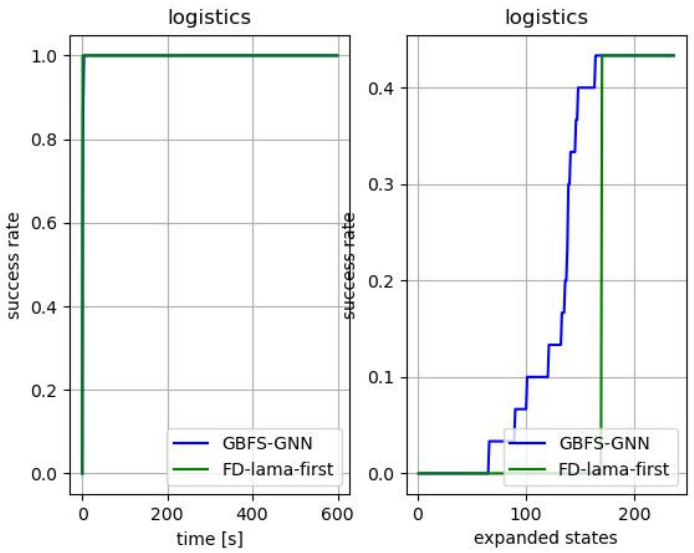

Figure 6.6: Success rate of our model against Fast Downward in *logistics* domain.

We can then conclude the following:

- The base models were not able to come to a generalizable strategy.
- We have presented a new model that improves the success rate of the aforementioned models, but it was again impossible to infer such strategy.
- This could be happening due to the nature of the problem, because *logistics* is quite a complex domain and being tightly-coupled there are a lot of subgoals that need to be ordered.
- Our model outperforms Fast Downward in terms of expanded states.

6.7 Depots

Since *depots* is also a tightly-coupled domain such as *logistics*, it is expected to exhibit a similar behavior. In this case, again, we have the same problem as in *logistics*: sequences of actions can run in parallel but at a certain point they must be sequentialized.

We analyze this situation by describing a problem and discussing its behavior with the GNAT-GN model, which was the one that output the best results in validation. In Figure 6.7, we can observe a complete definition of the problem, in which we have several distributors and several pallets, and the objective is to get crates on top of some pallets. For that, we have five trucks: `truck0`, `truck1`, `truck2`, `truck3` and `truck4`, which transport crates from one place to another and with the help of some hoists they move the crate from it to the pallet. Problem 6 shows the

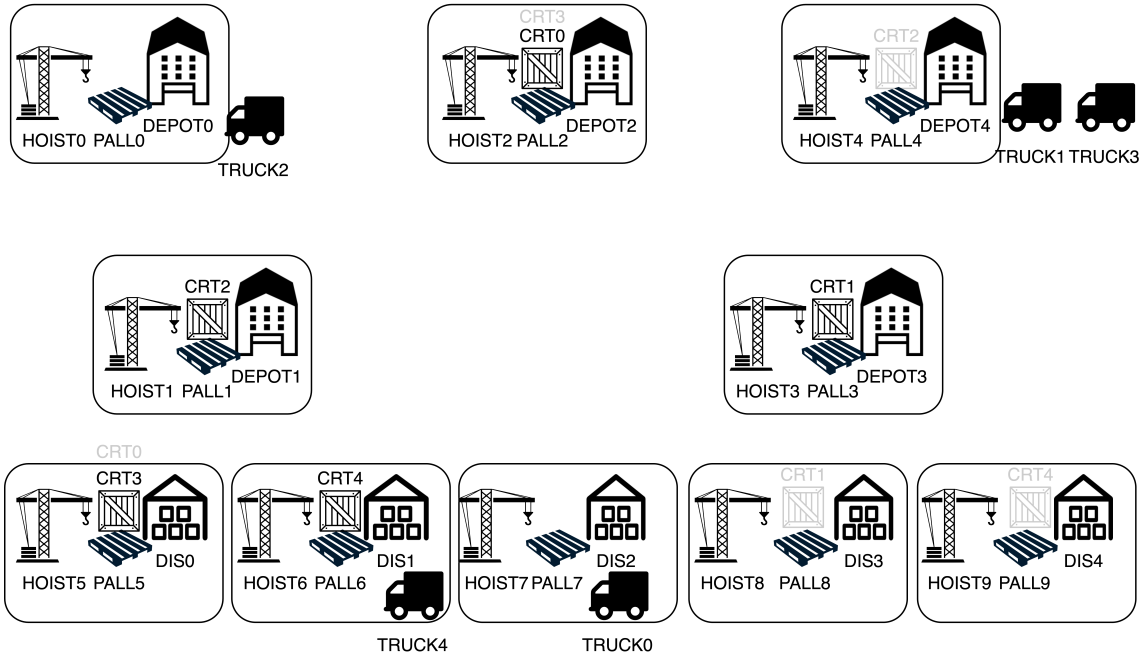

Figure 6.7: Definition of a problem from the *depots* domain. HOIST represents a hoist, PALL represents a pallet, DEPOT a depot, CRT a crate, DIS a distributor and TRUCK a truck. We can see in grey the goal objectives of the problem. The execution of the plan is in Problem 6.

Problem 6 Plan for the problem in Figure 6.7 using the GNAT-GN model. As we can see, it leads to infinite cycles in the `hoist0`, loading and unloading two crates.

```
PLAN:
                                        (load hst0 crt1 truck0 dpt0)
1.  (lift hst6 crt4 pall6 dis1)         (unload hst0 crt2 truck0 dpt0)
2.  (load hst6 crt4 truck4 dis1)        (drive truck3 dpt4 dis2)
3.  (drive truck4 dis1 dis3)            (load hst0 crt2 truck0 dpt0)
4.  (unload hst8 crt4 truck4 dis3)      (unload hst0 crt1 truck0 dpt0)
5.  (load hst8 crt4 truck4 dis3)        (load hst0 crt1 truck0 dpt0)
6.  (drive truck4 dis3 dis2)            (unload hst0 crt2 truck0 dpt0)
7.  (unload hst7 crt4 truck4 dis2)      (load hst0 crt2 truck0 dpt0)
8.  (load hst7 crt4 truck4 dis2)        (unload hst0 crt1 truck0 dpt0)
9.  (drive truck4 dis2 dpt4)            (load hst0 crt1 truck0 dpt0)
10. (unload hst4 crt4 truck4 dpt4)      (unload hst0 crt2 truck0 dpt0)
11. (load hst4 crt4 truck4 dpt4)        (load hst0 crt2 truck0 dpt0)
12. (drive truck4 dpt4 dpt0)            (unload hst0 crt1 truck0 dpt0)
13. (unload hst0 crt4 truck4 dpt0)      (load hst0 crt1 truck0 dpt0)
[...]                                   [...]
```

problematic of these domains that we talked about before: the appearance of infinite cycles because of the inability of properly learning the policy, which leads to a loopy behaviour that prevents the algorithm from converging.

In this case we tested again the new model that we proposed in the previous section, GNAT-GNAT-GN, obtaining an improvement in results such that from **54%** of success rate with the GNAT-GN model, we achieved **100%** in validation with the GNAT-GNAT-GN model. This huge improvement is very positive but must be analyzed.

In Extract 4 we see why the GNAT-GNAT-GN model largely improves the results. The new attention layer prevents the planner to get stuck in `load` and `unload` cycles. This helps the search converge and explains the huge improvement in validation. In Figure 6.8 we see how the testing went, outperforming again Fast Downward with our model in terms of expanded states. We were again not able to enlarge the testing sizes because of memory limitations in the nature of the domain.

In the left column from Extract 4 we see the full plan from the GNAT-GNAT-GN model, while in the second one we see an extract for the same problem but using the second model GNAT-GN. Note that on the left side the `load` and `unload` actions are non blocking, meaning that they do not produce cycles, while on the right side this is the main problem.

We analyzed the success rate of our model (GNAT block - GNAT block - GN block) against Fast Downward in *depots* domain, in terms of time and memory consumed, for problems of 5-6 depots, 5-6 distributors, 5-6 trucks, 7-9 pallets, 7-9 hoists and 7-9 crates.

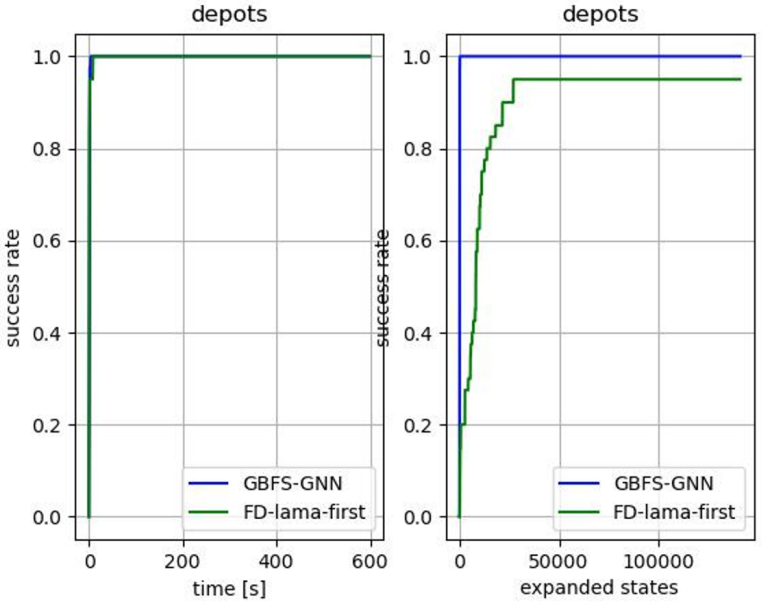

Figure 6.8: Success rate of our model against Fast Downward in *depots* domain.

Extract 4 Extract of a plan for some *depots* problem using the GNAT-GNAT-GN model versus using the GNAT-GN model.

```
GNAT-GNAT-GN extract (full plan):

(lift hst2 crt0 pall2 dpt2)
(lift hst5 crt3 pall5 dis0)
(drive truck4 dis1 dis0)
(drive truck4 dis0 dpt2)
(load hst2 crt0 truck4 dpt2)
(drive truck4 dpt2 dis0)
(load hst5 crt3 truck4 dis0)
(drive truck4 dis0 dpt2)
(drive truck4 dpt2 dpt3)
(unload hst3 crt3 truck4 dpt3)
(load hst3 crt3 truck4 dpt3)
(drive truck4 dpt3 dis0)
(unload hst5 crt3 truck4 dis0)
(load hst5 crt3 truck4 dis0)
(drive truck4 dis0 dpt2)
(drive truck0 dis2 dis0)
(unload hst2 crt3 truck4 dpt2)
(drop hst2 crt3 pall2 dpt2)
(unload hst2 crt0 truck4 dpt2)
(load hst2 crt0 truck4 dpt2)
(drive truck4 dpt2 dis0)
(unload hst5 crt0 truck4 dis0)
(drop hst5 crt0 pall5 dis0)
(lift hst3 crt1 pall3 dpt3)
(drive truck4 dis0 dpt3)
(load hst3 crt1 truck4 dpt3)
(drive truck4 dpt3 dis3)
(unload hst8 crt1 truck4 dis3)
(drop hst8 crt1 pall8 dis3)
(lift hst1 crt2 pall1 dpt1)
(drive truck2 dpt0 dpt1)
(load hst1 crt2 truck2 dpt1)
(drive truck2 dpt1 dpt4)
(unload hst4 crt2 truck2 dpt4)
(drop hst4 crt2 pall4 dpt4)
(lift hst6 crt4 pall6 dis1)
(drive truck2 dpt4 dis1)
(load hst6 crt4 truck2 dis1)
(drive truck2 dis1 dis4)
(unload hst9 crt4 truck2 dis4)
(drop hst9 crt4 pall9 dis4)
```

```
GNAT-GN extract (leads to a loop):

(lift hst6 crt4 pall6 dist1)
(load hst6 crt4 truck4 dist1)
(drive truck4 dist1 dist3)
[...]
(load hst1 crt2 truck4 dpt1)
(unload hst1 crt1 truck4 dpt1)
(load hst1 crt1 truck4 dpt1)
(unload hst1 crt2 truck4 dpt1)
(load hst1 crt2 truck4 dpt1)
(unload hst1 crt1 truck4 dpt1)
(load hst1 crt1 truck4 dpt1)
(unload hst1 crt2 truck4 dpt1)
(load hst1 crt2 truck4 dpt1)
(unload hst1 crt1 truck4 dpt1)
(load hst1 crt1 truck4 dpt1)
(unload hst1 crt2 truck4 dpt1)
(load hst1 crt2 truck4 dpt1)
(unload hst1 crt1 truck4 dpt1)
(load hst1 crt1 truck4 dpt1)
(unload hst1 crt2 truck4 dpt1)
(load hst1 crt2 truck4 dpt1)
(unload hst1 crt1 truck4 dpt1)
(load hst1 crt1 truck4 dpt1)
[...]
```

The conclusions of the analysis for this domain are as follows:

- Again, the base models were not able to come to a generalized strategy, but with our new GNAT-GNAT-GN model, the success rate raises up to 100%.
- This strategy is not as easy as in other domains, because even though with the new GNAT-GNAT-GN model we get the desired results we are not able to infer a strategy that is easily explainable. That is, it is able to find solutions to this problem where the search is guided by RL but there is not a clearly defined strategy or clearly identifiable pattern.
- Our model outperforms Fast Downward in terms of expanded states and equals it in terms of speed.

6.8 Elevators

For this domain, which is also tightly-coupled, we can expect the same situation as in the previous two. In this case the problem is that the elevators, since they do not pertain to all the blocks, must coordinate to take people from one floor to another whenever their destination is in a block other than the one from which they departed.

We analyze this situation using the problem of Figure 6.9, where we can see that we have a ten-floor building in which there are two blocks and an elevator for each block, `as0` and `as1`. In addition, there is a fast elevator `asr` which only uses the even floors but can move throughout the building and four people that want to get to different floors. In Problem 7 we can see how the best model GNATv2-GN behaves with this problem, resulting in an infinite cycle in the `move_elevator` operation.

This domain has turned out to be tougher than we thought. As we obtained poor results with the initial models, we decided to go a step further and try to analyze the plans using the GNAT-GNAT-GN model that we have explained in previous sections. However, the success rate results did not go any higher, maintaining the 63% that was initially offered with the new model as well.

We can analyze why this happens by looking at Extract 5, in which we have a small extract of a plan made with the GNAT-GNAT-GN model for the problem proposed above. As can be seen, this time the cycle has moved to another part of the actions: `passenger-enters` and `passenger-leaves`. We hypothesize that this happens because the model is still unable to explain the entire nature of the problem, since in this case the elevators have several points where subplans can converge.

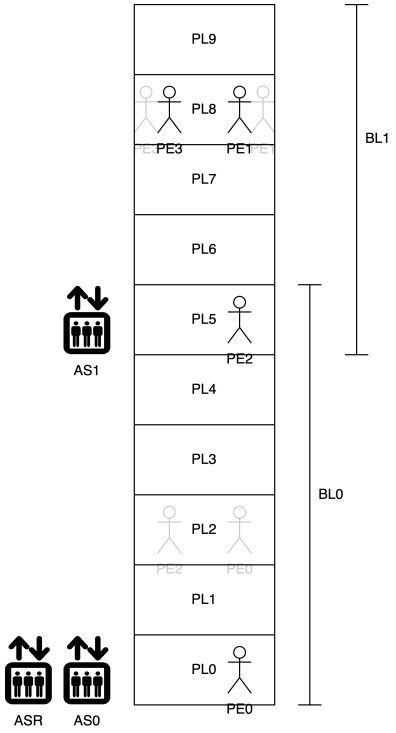

Figure 6.9: Definition of a problem of *elevators* domain. We have a building of 10 floors, two blocks inside the building and two elevators AS0, AS1 that move inside BL0 and BL1, respectively. Grey people are the goal, and black people represent the initial state. Plan is in Problem 7.

Problem 7 Plan for the problem in Figure 6.9 using the GNATv2-GN model. We can notice that it repeats the same strategy of moving elevators from one floor to another over and over again.

```
PLAN:

1.  (passenger-enters pe2 as1 pl5)  17. (move-elevat as0 pl5 pl2 bl0)
2.  (move-elevat as0 pl0 pl2 bl0)   18. (move-elevat as0 pl2 pl1 bl0)
3.  (move-elevat as0 pl2 pl3 bl0)   19. (move-elevat as0 pl1 pl4 bl0)
4.  (move-elevat as1 pl5 pl7 bl1)   20. (move-elevat as0 pl4 pl3 bl0)
5.  (move-elevat as0 pl3 pl4 bl0)   21. (move-elevat as0 pl3 pl2 bl0)
6.  (move-elevat as1 pl7 pl9 bl1)   22. (move-elevat as1 pl5 pl9 bl1)
7.  (move-elevat as0 pl4 pl2 bl0)   23. (move-elevat as0 pl2 pl3 bl0)
8.  (move-elevat as1 pl9 pl8 bl1)   24. (move-elevat as1 pl9 pl5 bl1)
9.  (move-elevat as0 pl2 pl1 bl0)   25. (move-elevat as1 pl5 pl7 bl1)
10. (move-elevat as0 pl1 pl3 bl0)   26. (move-elevat as0 pl3 pl5 bl0)
11. (move-elevat as1 pl5 pl9 bl1)   27. (move-elevat as1 pl7 pl9 bl1)
12. (move-elevat as1 pl9 pl6 bl1)   28. (move-elevat as0 pl5 pl1 bl0)
13. (move-elevat as0 pl0 pl3 bl0)   29. (move-elevat as0 pl1 pl2 bl0)
14. (move-elevat as0 pl3 pl2 bl0)   30. (move-elevat as0 pl2 pl4 bl0)
15. (move-elevat as0 pl2 pl1 bl0)   31. (move-elevat as0 pl4 pl5 bl0)
16. (move-elevat as0 pl1 pl5 bl0)   [...] (And so on)
```

In the *logistics* case, for example, the only point where each subplan had to be serialized was only in cities with airports, but in the *elevators* any floor can be susceptible to overlap: even floors overlap with the fast elevator and block intersections between block elevators. This is why we believe that the newly proposed model fails to achieve a good result, it would require searching for some other architecture or thinking in what way this situation could be dealt with.

Extract 5 Extract of a plan for the problem in Figure 6.9 , in which we can see that the actions get stuck in `passenger-leaves` and `passenger-enters`.

```
Extract from the GNAT-GNAT-GN model:

(passenger_enters pe2 as1 pl5)      (passenger_enters pe0 as0 pl2)
(move_elevat as0 pl0 pl5 bl0)       (passenger_leaves pe0 as0 pl2)
(move_elevat as1 pl5 pl8 bl1)       (passenger_enters pe0 as0 pl2)
(move_elevat as0 pl5 pl2 bl0)       (passenger_leaves pe0 as0 pl2)
(passenger_enters pe0 as0 pl2)      (passenger_enters pe0 as0 pl2)
(passenger_leaves pe0 as0 pl2)      (passenger_leaves pe0 as0 pl2)
(passenger_enters pe0 as0 pl2)      (move_elevat as0 pl0 pl5 bl0)
(passenger_leaves pe0 as0 pl2)      (move_elevat as0 pl5 pl2 bl0)
(passenger_enters pe0 as0 pl2)      (passenger_enters pe0 as0 pl2)
(passenger_leaves pe0 as0 pl2)      (passenger_leaves pe0 as0 pl2)
(passenger_enters pe0 as0 pl2)      (passenger_enters pe0 as0 pl2)
(passenger_leaves pe0 as0 pl2)      (passenger_leaves pe0 as0 pl2)
(passenger_enters pe0 as0 pl2)      (passenger_enters pe0 as0 pl2)
(passenger_leaves pe0 as0 pl2)      [...]
```

However, and taking into account that the expected result has not been achieved, the latter model still outperforms Fast Downward in terms of state expansion according to the results obtained with the test set. This can be seen in Figure 6.10, in which we analyzed the success rate of our model (GNAT block - GNAT block - GN block) against Fast Downward in *elevators* domain, in terms of time and memory consumed, for problems of 8-10 floors, 5-8 passengers, 1-2 blocks.

Therefore, we can conclude the following:

- We were not able to get a generalized strategy from this domain, not with the base models nor the proposed one.
- Even though we did not come to a solution, our model outperforms Fast Downward in all of our experiments.
- This problem has been difficult to approach with our models due to the high overlap existing in the nature of the domain.

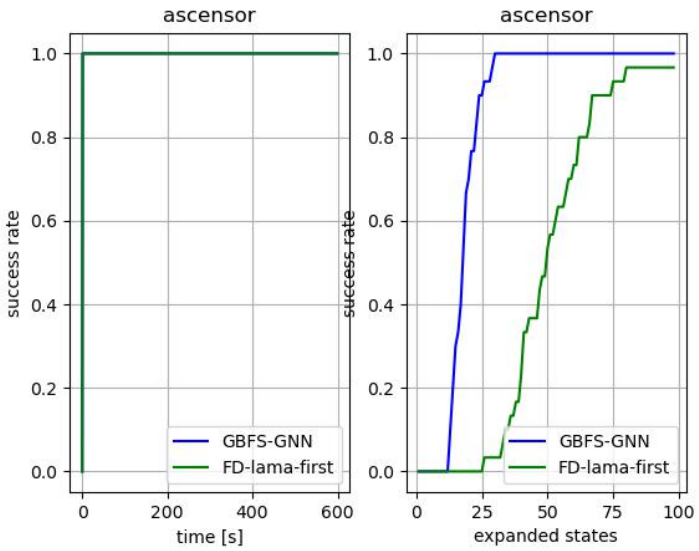

Figure 6.10: Success rate of our model against Fast Downward in *elevators* domain.

6.9 Hanoi

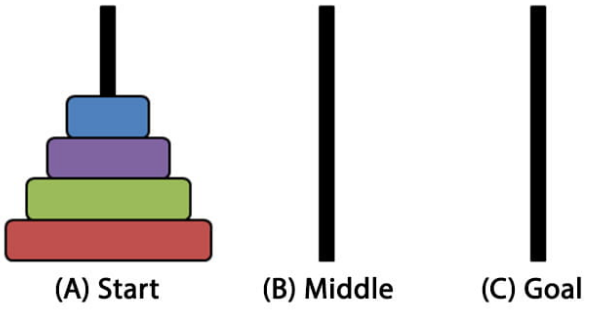

Figure 6.11: Basic setup of a hanoi problem.

This semantic knowledge that we talked about in previous sections, with which we can infer generalist strategies that solve much larger problems within a domain, does not make sense in the field of certain games such as hanoi. We can see an example of a problem in Figure 6.11.

Firstly, because we are in a domain in which there is a dominant strategy, but it is dependent on the problem size [KUM15]. And secondly, the number of moves that must be made to reach the solution grows exponentially with the number of disks there are. In fact, the problem needs $2^n - 1$ moves, with n being the number of disks.

For this same reason, despite reaching 100% success rate in training, it does not go above 0% in validation: the probability distribution that it is learning in training sizes is of no use at all for the validation size.

We can conclude the following:

- In the *hanoi* domain we were not able to come to a generalizable strategy.
- The success rate does not go above 0% because it makes no sense, as there is no strategy to be learned.
- The hard combinatorics of the domain makes it impossible to train a policy that is useful for higher order instances.
- The gaming domain is not something worth investigating from this perspective, since most games have these problems.

Chapter 7

Conclusion

In this chapter we will present the conclusions we have drawn from all the work that has been done, both from the purely theoretical-explanatory part and from the more experimental and analytical part of the models. In addition, we will discuss future lines of work that may arise from the development of this work, which will be developed in a future PhD project.

7.1 Achievements

First of all, we can say that the specific set of objectives of this work have been achieved. Specifically, we have explained the technologies of the model in a satisfactory way, we have presented them together with other models that we came up with, such as GNATv2-GN, GNATv3-GN or GNAT-GNAT-GN, and we have observed that they have equaled and even improved the results from what already existed previously in the paper from which we have started to do the work.

Secondly, we have managed to extract information about the proposed domains and their relationship with the taxonomy to which they belong. More specifically, we have observed that more general domains or domains with simpler strategies respond well to indiscriminate information propagation models, such as GN-GN, which has been explained in this paper.

On the other hand, attention models such as GNAT-GN, GNATv2-GN or GNATv3-GN respond well to the types of domains that we refer to as *loosely-coupled*, in which there are tasks that can be performed in parallel and that do not interfere with each other. We have seen that this attention helps to focus information on these virtually parallel tasks and allows fewer cycles to appear in the plans, which are the result of misinformation caused by inattention.

It has also been observed that tightly-coupled domains have problems with the initial two-layer GNN architectures, so we presented a new model, GNAT-GNAT-GN, which improves the results.

Finally, we were able to obtain generalized policies in most domains, which scaled well to problems of higher magnitude. However, it has been more difficult in some of the domains to find a policy that is explainable, i.e., there were domains for which it was not easy to find a regular pattern that explains the strategy or policy.

7.2 Future work

This work has been an approach to Generalized Planning using certain tools that we have found interesting. In future work, however, we will try to explore other ways in order to improve the performance we have obtained in this work. For instance:

- To explore different representations for the construction of the graph, so we can boost performance by including other useful information such as partial order for each planning state.
- To change the embedding generation, exploring other methods apart from the GNNs such as Logical Neural Networks [RIE20].
- To experiment within different types of GNN layers that include other types of information.
- To try and use several improvements to the model for it to boost its performance, such as using artificial nodes or pretraining the network.[1]
- To change or to improve RL algorithms, in order to collect more useful information at each training step.

[1] `https://towardsdatascience.com/how-to-boost-your-gnn-356f70086991`. Acessed: 11-09-2022.

Bibliography

[AIN19] Aineto D., Jiménez-Celorrio S., Onaindia E., 2019. *Learning action models with minimal observability*, Artificial Intelligence, 275, pp. 104-137.

[ALK20] Alkhazraji Y., Frorath M., Grützner M., Helmert M., Liebetraut T., Mattmüller R., Ortlieb M., Seipp J., Springenberg T., Stahl P., Wülfing J., 2020. *Pyperplan*, Zenodo, https://doi.org/10.5281/zenodo.3700819.

[AMI08] Amir E., Chang A., 2008. *Learning partially observable deterministic action models*, Journal of Artificial Intelligence Research 33, pp. 349–402.

[BAT18] Battaglia P. W., Hamrick J. B., Bapst V., Sanchez-Gonzalez A., Zambaldi V., Malinowski M., Tacchetti A., Raposo D., Santoro A., Faulkner R. et al, 2018. *Relational inductive biases, deep learning, and graph networks*, arXiv preprint, https://doi.org/10.48550/arXiv.1806.01261.

[BRO22] Brody S., Alon U., Yahav E., 2022. *How Attentive are Graph Attention Networks?*, ICLR, https://doi.org/10.48550/arXiv.2105.14491.

[CRE13] Cresswell S. N., McCluskey T. L., West M. M., 2013. *Acquiring planning domain models using LOCM*, Knowledge Engineering Review 28 (02), pp. 195–213.

[DON19] Dong H., Mao J., Lin T., Wang C., Li L., Zhou D., 2019. *Neural Logic Machines*, ICLR 2019.

[EBE18] Ebert F., Finn Ch., Dasari S., Xie A., Lee A., Levine S., 2018. *Visual foresight: Model-based deep reinforcement learning for visionbased robotic control.*, arXiv preprint arXiv:1812.00568.

[FEY19] Fey M., Lenssen J.E., 2019. *Fast Graph Representation Learning with PyTorch Geometric*, ICLR Computer Science, https://doi.org/10.48550/arXiv.1903.02428.

[FOX03] Fox M., Long D., 2003. *PDDL2.1: an extension to PDDL for expressing temporal planning domains*, J. Artif. Intell. Res. 20, pp. 61–124.

[FRA19] Frasinaru C., Raschip M., 2019. *Greedy Best-First Search for the Optimal-Size Sorting Network Problem*, Procedia computer science, 159:447-454.

[GEF18] Geffner H., 2018. *Model-free, Model-based, and General Intelligence*, IJCAI 2018: 10-17.

[GEH22] Gehring C., Asai M., Chitnis R., Silver T., Kaelbling L. P., Sohrabi S., Katz M., 2022. *Reinforcement Learning for Classical Planning: Viewing Heuristics as Dense Reward Generators.*, ICAPS 2022.

[GHA04] Ghallab M., Nau D., Traverso P., 2004. *Automated Planning. Theory and practice*, Morgan Kaufmann.

[GHA16] Ghallab M., Nau D., Traverso P., 2004. *Automated Planning and Acting*, Cambridge University Press. doi:10.1017/CBO9781139583923.

[HAM21] Hamrick J. B., Friesen A. L., Behbahani F., Guez A., Viola F., Witherspoon S., Anthony T., Buesing L. S., Velickovic P., Weber T., 2021. *On the role of planning in model-based deep reinforcement learning*, ICLR 2021.

[HEL11] Helmert M., 2011. *The Fast Downward Planning System*, Journal Of Artificial Intelligence Research, Volume 26, pages 191-246, `https://doi.org/10.48550/arXiv.1109.6051`.

[HEY90] Heyman D.P., Sobel M.J., 1990. *Markov Decision Processes*, Stochastic Models, Elsevier, Volume 2.

[HOF01] Hoffmann J., Nebel B., 2001. *The FF planning system: Fast plan generation through heuristic search*, Journal of Artificial Intelligence Research, 14:253-302.

[KAI20] Kaiser L., Babaeizadeh M., Milos P., Osinski B., Campbell R., Czechowski K., Erhan D., Finn C., Kozakowski P., Levine S., Mohiuddin A., Sepassi R., Tucker G., Michalewski H., 2020. *Model Based Reinforcement Learning for Atari.*, ICLR 2020.

[KUM15] Kumar-Mishra B., Vishnoi S., 2015. *Towers of Hanoi - An Iterative Solution for Parallel Computation.*, IJR 2015.

[KUC18] Kucera J., Barták R., 2018. *LOUGA: learning planning operators using genetic algorithms*, Pacific Rim Knowledge Acquisition Workshop, PKAW-18, 2018, pp. 124–138.

[MCD98] McDermott D., Ghallab M., Howe A., Knoblock C., Ram A., Veloso M., Weld D., Wilkins D., 1998. *PDDL - the planning domain definition language*, arXiv preprint, `https://doi.org/10.48550/arXiv.2005.02305`.

[MOE22] Moerland T. M., Broekens J., Plaat A., Jonker C. M., 2022. *A Unifying Framework for Reinforcement Learning and Planning.*, arXiv preprint, arXiv:2006.15009 [cs.LG].

[NGU11] Nguyen-Tuong D., Peters J., 2011. *Model learning for robot control: a survey*, Cognitive processing, 12(4), pp. 319–340.

[PER22] Pérez-Gil O., Barea R., López-Guillén E., et al., 2022. *Deep reinforcement learning based control for Autonomous Vehicles in CARLA.*, Multimed Tools Appl 81, pp. 3553–3576.

[PUT94] Putterman M., 1994. *Markov Decision Processes—Discrete Stochastic Dynamic Programming*, John Wiley and Sons, Inc., New York.

[RIE20] Riegel R., Gray A., Luus F., Khan N., Makondo N., Akhalwaya I. Y., Qian H., Fagin R., Barahona F., et al., 2020. *Logical Neural Networks*, NeurIPS 2020, https://doi.org/10.48550/arXiv.2006.13155.

[RIV20] Rivlin O., Hazan T., Karpas E., 2020. *Generalized Planning with Deep Reinforcement Learning*, arXiv preprint, https://doi.org/10.48550/arXiv.2005.02305.

[SAN18] Sanchez A., Hees N., Springenberg J.T., Merel J., Hadsell R., Riedmiller M.A., Battaglia P., 2018. *Graph networks as learnable physics engines for inference and control.*, In: Proceedings of ICML, pp. 4470-4479.

[SCH17] Schulman J., Wolski F., Dhariwal P., Radford A., Klimov O., 2017. *Proximal policy optimization algorithms*, arXiv preprint, https://doi.org/10.48550/arXiv.1707.06347.

[SEI22] Seipp J., Torralba A., Hoffmann J., 2022. *PDDL Generators*, Zenodo, https://doi.org/10.5281/zenodo.6382173.

[SEJ18] Sejr M., Kipf T. N., Bloem P., Van der Berg R., Titov I., Welling M., 2018. *Modeling Relational Data with Graph Convolutional Networks*, pp. 593-607, ESWC 2018.

[SIL18] Silver D., Hubert T., Schrittwieser J., Antonoglou I., Lai M., Guez A., Lanctot M., Sifre L., Kumaran D., Graepel T., et al, 2018. *A general reinforcement learning algorithm that masters Chess, Shogi, and Go through self-play.*, Science, 362 (6419), pp. 1140–1144.

[SIL19] Silver D., Huang A., Maddison C.J., Guez A., Sifre L., Van Den Driessche G., Schrittwieser J., Antonoglou I., Panneershelvam V., Lanctot M., et al., 2019. *Mastering the game of go with deep neural networks and tree search*, Nature, 529(7587):484.

[STA22] Stahlberg S., Bonet B., Geffner H., 2022. *Learning General Optimal Policies with Graph Neural Networks: Expressive Power, Transparency,*

and Limits, Proceedings of the International Conference on Automated Planning and Scheduling, 32(1), 629-637. Retrieved from `https://ojs.aaai.org/index.php/ICAPS/article/view/19851`.

[SUT18] Sutton R.S., Barto A.G., 2018. *Reinforcement learning: An introduction*, MIT Press 1998, ISBN 978-0-262-19398-6.

[TOR15] Torreño A., Onaindia E., Sapena O., 2015. *FMAP: Distributed Cooperative Multi-Agent Planning*, Applied Intelligence, Volume 41, Issue 2, pp. 606-626, `https://doi.org/10.48550/arXiv.1501.07250`.

[VAS17] Vaswani A., Shazeer N., Parmar N., Uszkoreit J., Jones Ll., Gomez A. N., Kaiser L., Polosukhin I., 2017. *Attention is all you need*, Advances in neural information processing systems, p.5998-6008, `https://doi.org/10.48550/arXiv.1706.03762`.

[VEL17] Velickovic P., Cucurull G., Casanova A., Romero A., Lio P., Bengio Y., 2017. *Graph Attention Networks*, arXiv preprint, `https://doi.org/10.48550/arXiv.1710.10903`.

[WIL92] Williams R. J., 1992. *Simple statistical gradient-following algorithms for connectionist reinforcement learning*, Machine learning, 8(3-4):229-256.

[WU21] Wu Y., Lian D., Xu Y., Wu L., Chen E, 2021. *Graph convolutional networks with markov random field reasoning for social spammer detection*, In: Proceedings of AAAI, 34, pp.1054-1061.

[WU-L20] Wu L., Chen Y., Shen K., Guo X., Gao H., Li S., Pei J., Long B., 2020. *Graph neural networks for Natural Language Processing: A survey*, Arxiv, `https://doi.org/10.48550/arXiv.2106.06090`.

[WU-S20] Wu S., Sun F., Zhang W., Xie X., Cui B., 2020. *Graph neural networks in Recommender Systems: A survey*, Arxiv, `https://doi.org/10.48550/arXiv.2011.02260`.

[YAN07] Yang Q., Wu K., Jiang Y., 2007. *Learning action models from plan examples using weighted MAX-SAT*, Artificial Intelligence 171 (2-3), pp. 107-143

[YUR20] Yurtsever E., Capito L., Redmill K. A., Özgüner Ü., 2020. *Integrating Deep Reinforcement Learning with Model-based Path Planners for Automated Driving.*, EEE Intelligent Vehicles Symposium, pp. 1311-1316.

[ZHO20] Zhou J., Cui G., Hu S., Zhang Z., Yang C., Liu Z., Wang L., Li C., Sun M., 2020. *Graph neural networks: A review of methods and applications*, AI Open, 1, pp. 57-81, `https://doi.org/10.1016/j.aiopen.2021.01.001`.

[ZHU10] Zhuo H. H., Yang Q., Hu D. H., Li L., 2010. *Learning complex action models with quantifiers and logical implications*, Artificial Intelligence, 174 (18), pp. 1540–1569.

[ZHU13] Zhuo H. H., Kambhampati S., 2013. *Action-model acquisition from noisy plan traces*, International Joint Conference on Artificial Intelligence, IJCAI-13, 2013, pp. 2444–2450.

List of Figures

List of Tables