Automated planning optimization

Intelligent preprocessing of the state space using Graph Neural Networks

Master Thesis

CS-23-MI-10-12

Aalborg University
Electronics and IT

This project has benefitted from the use of OpenAI's GPT-4, known as ChatGPT, for grammar correction and readability improvement. The AI model was instrumental in enhancing the readability and clarity of the text, ensuring grammatical accuracy, and assisting in the rephrasing of complex sentences.

AALBORG UNIVERSITY
STUDENT REPORT

Electronics and IT
Aalborg University
http://www.aau.dk

Title:
Automated planning optimization

Theme:
Intelligent preprocessing of the state space using Graph Neural Networks

Project Period:
Spring semester 2023

Project Group:
CS-23-MI-10-12

Participant(s):
Bartosz Piotr Lachowicz
Piotr Rafał Gzubicki

Supervisor(s):
Alvaro Torralba

Copies: 1

Page Numbers: 72

Date of Completion:
June 8, 2023

Abstract:

This master's thesis presents the outcomes of our project work during the 10th semester of the Computer Science program at Aalborg University. Our project focused on developing a tool using Graph Neural Networks to optimize automated planning processes. Building upon a previous prototype, our objective was to create a complete tool with hyperparameter tuning and feature engineering capabilities. The resulting tool significantly reduces the search state space for automated planners while retaining essential plan computation information. Our work was motivated by approaches that leverage experience gained from solving smaller planning instances to tackle larger ones.

Contents

Summary

This document serves as a master thesis detailing the outcomes of the project undertaken by our study group during the 10th semester of the Computer Science program at Aalborg University. The primary objective of this project was to develop an advanced tool capable of optimizing automated planning processes. To achieve this, we leveraged the cutting-edge technology of Graph Neural Networks.

Building upon a prototype that was developed in the previous semester, our development process aimed to create a comprehensive tool encompassing hyperparameter tuning and feature engineering. The ultimate goal was to produce a fully functional tool that could significantly reduce the search state space for automated planners while retaining the essential information required for efficient plan computation.

To achieve hyperparameter optimization, we employed an automated tool called SMAC in our project. SMAC allowed us to conduct experiments with various training parameters, enabling us to narrow down and select a number of relevant configurations based on accumulated experience.

In this project, we enhanced the previously developed prototype by introducing valuable features. Our focus was on identifying information that proves useful during the plan search and could potentially guide the Graph Neural Network towards improved classification. As a result, we incorporated landmarks and relaxed plans into the model.

Our motivation for this work stemmed from existing approaches that sought to gain valuable experience by solving smaller planning instances and then applying that knowledge to larger, more complex planning scenarios. By leveraging this accumulated knowledge, our tool aimed to enhance the efficiency and effectiveness of automated planning processes.

Throughout the project, we conducted extensive research, implemented innovative methodologies, and performed rigorous testing to ensure the effectiveness and reliability of our solution. The final outcome is a complete tool that successfully optimizes automated planning processes, contributing to the advancement of this field.

We believe that our work provides valuable insights and contributes to the growing body of knowledge in the area of automated planning optimization. By developing this tool, we aim to facilitate more efficient planning processes, opening up possibilities for improved problem-solving in various domains.

Aalborg University, June 8, 2023

Chapter 1

Introduction

This project is oriented on the area of automated planning and its improvement. It is based on previous research and strives to enhance the existing solution with feature engineering and with the use of a hyperparameters tunning tool. The concept of automated planning focuses on finding an optimal sequence of actions that leads to achieving all predefined goals, given a specific environment and initial state. The formalized form of the problems is described in the further part of the report.

The automated planning area can be described as a discipline of Artificial Intelligence that aims at the development of generic algorithms allowing autonomous systems to choose and organize their actions to achieve a goal by anticipating their effects [10]. It is oriented on providing tools to solve planning problems among versatile problems. Example domains in which such a solution can be applied are:

- Logistics: delivery companies assigning vehicles the routes and packages
- Agriculture: correct distribution of workers and machines for harvesting
- Satellite operations: deciding on a set of operations to take images of various objects
- Network security: having information about the internal network and automated discovery of possible exploits

The above-mentioned domains are only a few of many that can be tackled using automated planning tools. However, all of them suffer from the same issue which is scalability. With the size of the problems, the computational difficulty grows exponentially. One of the research areas is focused on exploring the options for generic tools which could improve the efficiency of planners and which could be applied regardless of the domain the planner works with.

The inspiration for the project and the base concept expanded in this research comes from the work of Daniel Gnad, Álvaro Torralba, Martín Ariel Domínguez, Carlos Areces and Facundo Bustos in the paper *"Learning How to Ground a Plan - Partial Grounding in Classical Planning"* [11]. The solution proposed by the above-mentioned researchers takes advantage of machine learning algorithms which are trained using specific domains. Afterwards, the learned knowledge can be applied in real-world scenarios. Conceptually, experiences learned from small problems can be used to help solve much bigger instances. Consequently, the model is trained on small and medium size problems and later it is applied to much more computationally expensive examples. Intuitively, knowledge gained from solving problems of delivering 10 packages can prove to be beneficial when applied to much bigger contexts of 100 packages and more. The planning systems thanks to the improvements could be applied more broadly in modern solutions at a bigger scale.

This project builds upon our previous work, which introduced an innovative technique for optimizing automated planning. The approach involves utilizing a pre-processor and harnessing the power of graph representation and Graph Neural Network (GNN) architecture. In the subsequent sections of this report, we will delve into a detailed description of this process.

The system after the introduction of the GNN enhancement presents promising results. However, as the solution was only a prototype created to investigate potential benefits, it was not explored throughout. Therefore, in this project by performing further analysis and introducing new components of the training and planning pipeline a more complete solution will be developed.

The project will confront its initial challenge directly by focusing on enhancing the knowledge input provided to the Graph Neural Network. Although GNNs can derive valuable insights from the graph structure alone, their capabilities expand significantly when supplemented with additional feature knowledge.

The success of Graph Neural Networks (GNNs), much like that of traditional Neural Networks, partially relies on optimal network configuration. In the case of the prototype solution, this was a weak point. The selection of network parameters was based solely on empirical evidence, resulting in suboptimal training efficiency and overall task performance. Given that the tool's objective is to optimize planning problems across diverse domains, relying on a single preselected parameter configuration poses a problem. As a result, significant discrepancies in performance may be observed, highlighting the need for a more adaptive approach.

To address the identified challenges in this project in this project additional components will be added to the training and planning pipelines. Firstly, the features will be introduced which may unlock the full potential of the GNN. Through the experimental process, the progress will be monitored and the impact on the performance will be analyzed. This will push the solution one step closer towards achieving optimal results. Additionally to address the second identified challenge a hyperparameter tuning tool will be used for training. This will ensure optimal model settings across all domains the GNN will work with. The enhanced solution may enable the GNNs' full potential and provide planners with a more accurate and computationally simpler environment description, even before the first attempt at solving the planning problem.

Chapter 2

Background

Over time, the field of planning research has implemented standardization measures, which involve adopting a shared terminology for describing planning problems and their respective domains. To facilitate the representation of problem instances, planners have also developed a uniform approach to reasoning about these issues known as translation. This involves parsing and converting instances into a universal format that can be readily comprehended by the planner, and is referred to as "grounding". To dive deeper into the topic of planning, it is necessary to first clarify these formalities.

2.1 Planning Introduction

The classical planning problem P_c is defined as a tuple $P_c = (Dom, Ins)$, where Dom represents a specific domain and Ins stands for a given instance of the problem.

The domain is composed of various elements, including *object types*, *predicate definitions*, and *action definitions*. Each predicate includes a name and arguments that can be substituted with objects from a set that is defined by problem instance Ins. By evaluating these predicates with the specified arguments, it can be determined whether they are true or false.

An action definition specifies an action along with its arguments and conditions, which are a collection of predicates defining preconditions and effects. For example, in order to capture an image of a particular star using a satellite, certain preconditions must be met, such as the satellite being *"pointed"* at the star, having a *"calibrated"* instrument, and having *"power_available"*. The result of satisfying these preconditions would be obtaining an image of the star, which is the effect.

The problem instance is represented by a tuple $Ins = (Obj, Init, Goal)$, where Obj is the collection of objects that are specified in the problem definitions. This is then followed by a description of the initial and goal states of the environment in which the task is situated.

2.1.1 PDDL Problem Representation

The Planning Domain Definition Language (PDDL) is a group of languages that standardizes the representation of planning problems. A planning task is typically defined in two files, where one encodes general knowledge about the planning environment (domain) and the other instantiates the actual problem. This definition is referred to as a lifted representation of the problem.[9].

The lifted PDDL task Π is represented by a tuple $(P, A, \sum, I, G)$, where P is a collection of predicates, A is a collection of action schemas, $\sum$ is a collection of objects, and I and G represent the initial and goal states of the task. This representation can be easily translated to the previous notion of classical planning problems since predicates and action schemas are part of the domain and are common for all instances in a given environment. The remaining components $(\sum, I, G)$ belong to the problem specification and are unique to each instance. In this report for simplicity we use examples specified in STRIPS [7] subset of PDDL.

2.1.2 Grounding

Planning models are typically represented using the PDDL notation. However, planners do not directly understand problems structured using this method and require a grounded representation of the instances, such as the STRIPS notation. In this section, we will explain the grounding process, which is one of the most essential steps in the planning process.

Grounding is a process of translating lifted PDDL representation into an instantiated set of actions and predicates using the objects from $\sum$ To produce the grounded representation the computation of all valid instantiates with objects is needed. The result of the grounding is the set of prepositions also called facts or atoms and grounded actions referred to as operators. This yields the state space of the search which is explored by the planner 2.2.1. Moreover, the output consists of all variations even though only a small subset will be necessary to solve a specific planning task.

A STRIPS instance is a tuple (F, O, I, G), where F is a set of grounded predicates, O is the set of grounded actions, and I and G are the initial and goal states.

Instantiated predicates hold binary values and describe the state of the environment at a given time. A state $s \subseteq F$ is a set of facts (also called atom), with the initial state being $I \subseteq F$ and goal state $G \subseteq F$. Instantiated action schemas are called operators. An operator o is applicable in a state s if its precondition $pre(o) \subseteq s$. If the operator is applicable, then the state can be changed by applying it, resulting in the transition from s to s' via o, denoted by $s \xrightarrow{o} s'$. A sequence of operators that leads from the initial state to the goal state is called a plan. If such a sequence is found, then the planning task is considered solved. The plan is optimal if the discovered sequence has the lowest possible cost.

2.2 Planners

Planners utilize a translated and grounded representation of the problem, such as STRIPS, and employ various algorithms to search for a solution. The typical approach involves using search algorithms to traverse a tree consisting of all reachable states via operator applications. The root of the tree is the initial state, and one or more leaves represent the desired goal states. The intermediate nodes correspond to states, and the edges connect states with operators applicable in a given state. The complexity of the traversal grows as the number of reachable states increases, making the planning process more challenging.

2.2.1 Search State Space Problem

As previously mentioned, planners face the challenge of scalability. The translation process generates a set of all possible combinations of instantiated objects, even though only a small fraction of them are relevant to the solution. As the number of components in the domain increases, the time required to find a solution also increases. In fact, the time required to solve a problem grows exponentially in relation to the growth of arguments. Real-world planning challenges, such as logistics tasks, can have a state space so large that the efficiency and usability of planners become limited. When considering all the available packages, trucks, and possible destinations, the number of combinations becomes impossible to handle. Therefore, the research challenge is focused on addressing the exponential growth of grounded actions.

The primary tool used for solving the planning instances in this research is Fast Downward (FD) [16], which is a state-of-the-art planner that employs a heuristic approach to compute optimal plans. It is compatible with planning problems defined in PDDL and supports various search algorithms like $A*$[15], LAMA[33], and LM-cut[17].

2.2.2 Preprocessing

The limitation of scalability of planning problems can be approached in various ways, and this report presents a solution inspired by prepossessing tools. The proposed method involves optimizing planning problem instances before the planner attempts to solve them. One notable example is the Scorpion planner [37], which incorporates a preprocessor that analyzes the problem and strives to reduce task complexity. The Scorpion planner integrates the $h^2 - preprocessor$ out of the box [1], which aims to simplify the planning task model by examining mutex pairs that cannot belong to any plan. After analysis, unreachable facts are removed, thereby reducing the search state space. Considering the similar characteristics of the $h^2 - preprocessor$ and the proposed GNN solution, these two approaches will be compared against each other as well as combined together. The Scorpion planner with $h^2 - preprocessor$ setup will serve as the baseline for comparison.

2.3 Previous Work

The techniques and choices outlined in this report derive from the team's prior efforts in the previous research [13]. Specifically, a prototype was constructed to validate the initial hypotheses, proving that the Graph Neural Networks can be applied successfully in the automated planning problems' optimization, thereby encouraging the ongoing refinement of the approach.

In order to fully comprehend the solution first the basic concepts of graphs and problem mappings must be explored. A *Graph G* can be defined as a collection of nodes V and edges E. Each node $v \in V$ is characterized by its specific features, denoted as x_v. In a graph with n nodes, the relationships between nodes are represented by a binary adjacency matrix $A = V \times V$, where a value of 1 in the $A(i,j)$ entry indicates that nodes i and j are connected by an edge. Additionally, two nodes v_1 and v_2 are considered neighbors if $A(v_1, v_2) = 1$.

2.3.1 Mapping Problems onto PDG

Various kinds of graphs are available to represent planning tasks. In the previous iteration of the research the classification performance of using both the solution based on *Causal Graph*[34] and the one based on *Problem Description Graph (PDG)*[30] was explored. Through experimentation, the PDG was found to be the most promising option. Consequently, to streamline the project and avoid unnecessary complexity, the Causal Graph will not be further explored in this research. Because of certain practical nuances, the PDG was slightly modified and does not correspond exactly to what was proposed in [30] however the main structure of the graph is the same.

A Problem Description Graph is a graph representation of a planning problem exploiting the relationship between actions and predicates. To create the *Problem Description Graph*, the relationships between operators and predicates are leveraged. Furthermore, the PDG is considered a heterogeneous graph due to its inclusion of multiple types of nodes and links. Specifically, four distinct node types are identified in the graph:

- **operator nodes**: grounded actions, generate operator nodes, for instance:
  - *turn_to(satellite1, star1)*
- **value nodes**: single fact:
  - *pointing(star1, satellite2)*

  Certain sets of facts are exclusive meaning that one only of the can be true at the time. This allows for the generation of a logical grouping of such sets - Variables.
- **variable nodes**: logical grouping of value nodes reasoning about one specific object. For variables bound to *satellite1* for a predicate *pointing* could group:
  - *pointing(satellite1, star1)*
  - *pointing(satellite1, star2)*
  - *pointing(satellite1, star3)*

Four distinct types of edges can be observed in the graph, each of which reflects the relationships between objects in their grounded representation. These edges are mapped to the PDG in the following manner:

- **Value-Variable edge**: A Value and Variable have an edge if the Value belongs to the Variable, the edge is not directed. The variable can have multiple values but a value node can only be connected to one variable node.
- **Value->Operator (Precondition)** A precondition Value and an Operator are related if the Value is in the preconditions of the action from which the operator was derived. This edge is pointing from value towards the operator
- **Operator->Value (Effect)** An effect Value and Operator are related if the Value is in the effects of the action from which the operator was derived. For this type of connection the operator node point towards the value node.

Value nodes are connected to one or more operator nodes (by either precondition or effect edge), if said values appear in either the precondition or effect of the action definition associated with the operator. This relationship can be visualized in the figure below.

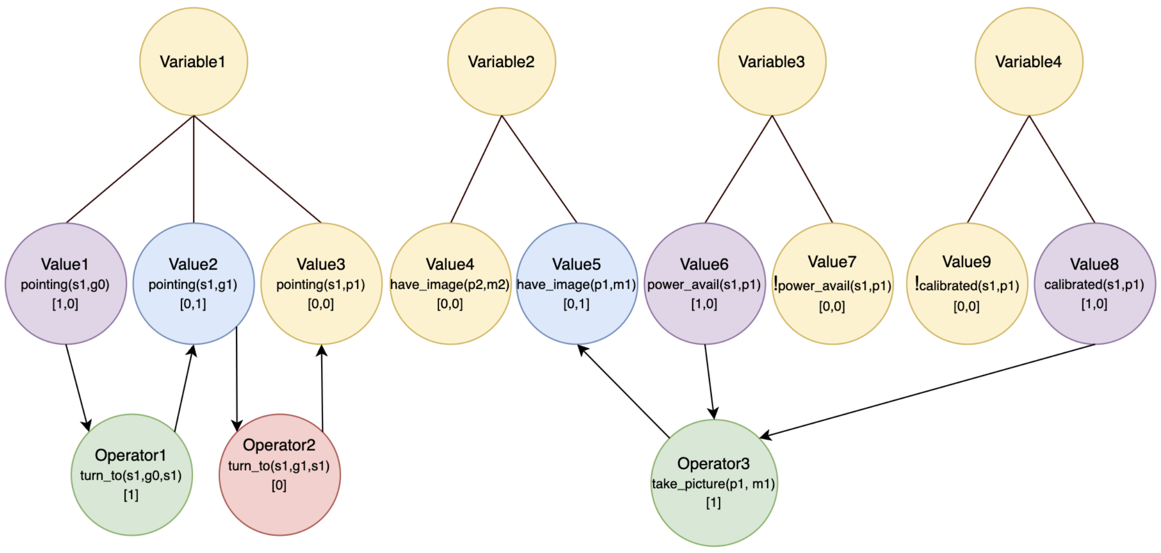

Variable1
Variable2
Variable3
Variable4
Value1 pointing(s1,g0) [1,0]
Value2 pointing(s1,g1) [0,1]
Value3 pointing(s1,p1) [0,0]
Value4 have_image(p2,m2) [0,0]
Value5 have_image(p1,m1) [0,1]
Value6 power_avail(s1,p1) [1,0]
Value7 !power_avail(s1,p1) [0,0]
Value9 !calibrated(s1,p1) [0,0]
Value8 calibrated(s1,p1) [1,0]
Operator1 turn_to(s1,g0,s1) [1]
Operator2 turn_to(s1,g1,s1) [0]
Operator3 take_picture(p1, m1) [1]

Figure 2.1: Problem Description Graph with features and labels. Green and red colors indicate whether an operator is representing a good or a bad action. Purple represents values in the initial state of the problem and blue represents values in the goal state of the problem.

2.3.2 Graph Neural Networks

Graph Neural Networks are a class of neural networks designed for data that is represented in graph form. These networks leverage the connections between nodes, utilizing the edges to propagate information between connected nodes. Graph Neural Networks have found diverse applications, including the modelling of social networks [48], physical systems in natural science [35], and protein-protein interface networks [8].

GNN updates

Bronstein, Bruna, Cohen, and Veličković provide a comprehensive analysis of Graph Neural Networks (GNNs) [3] in which they identify that all modern GNN architectures stem from one of three types: Convolutional, Attentional, and Message Passing. The relationship between these types is such that Convolutional GNNs are a subset of Attentional GNNs, which in turn are a subset of Message Passing GNNs. Since the Message Passing GNN is the most general, we will describe its logic further.

The Message Passing GNN consists of two functions: an aggregate function and an update function. To compute a hidden state h_u for a node u, the GNN first aggregates all feature vectors x_v for all neighbours $v \in N_u$ where N_u is the set of all nodes that have a direct edge to node u. Then, the GNN combines the node features x_u with the aggregated information from the neighbours and updates the hidden representation h_u as follows:

$$h_u^{t+1} = \phi\left(h_u^t, \bigoplus_{v \in N_u} \psi(h_v^t)\right) \tag{2.1}$$

The permutation invariant $\bigoplus$ controls the aggregation and accepts an arbitrary number of inputs (one per neighbour $v \in N_u$). The permutation invariant could be a sum or a pooling mechanism such as max, mean, or min similar to conventional convolutional networks. The trainable function $\psi(x_u, x_v)$ transforms the features of the node u and the feature vectors coming from each neighbouring node that are passed to the permutation invariant. Finally, to update the hidden representation h_u, the feature vector of node u is combined with the aggregated information z through a trainable non-linear function ϕ (e.g., ReLU).

In the context of Graph Neural Networks, the number of message-passing functions defined in the network architecture determines the depth of the network. Suppose we aim to construct a computational graph for a Graph Convolution Network (GCN), which is a specialized variant of a Message Passing GNN. In GCN, the function ψ is replaced with trainable weight matrix W, which is utilized to perform a matrix multiplication operation on the feature vectors of all neighbouring nodes for a given node, resulting in the final following formula.

$$\psi(h_v^t) = W * h_v^t \tag{2.2}$$

$$h_u^{t+1} = \phi\left(h_u^t, \bigoplus_{v \in N_u} W * h_v^t\right) \tag{2.3}$$

The following figure shows an example graph:

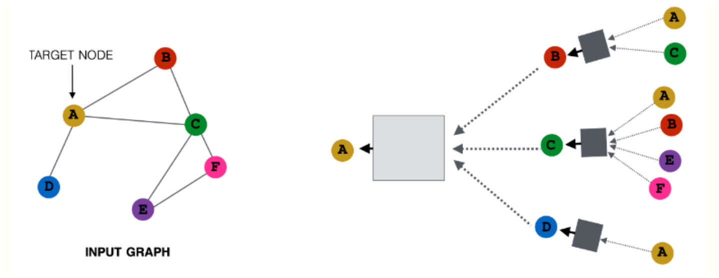


Figure 2.2: Example graph and convolutions. Source - [40]

The left-hand side of the figure shows the input graph. To create the embedding vector for node A, we consider its neighbours *B, C, D* without any specific order in the first layer. Next, we iterate over the neighbours of the neighbours. The depth of the network is clearly visible in the visualization, as it contains three identifiable layers.

2.3.3 Operator Classification

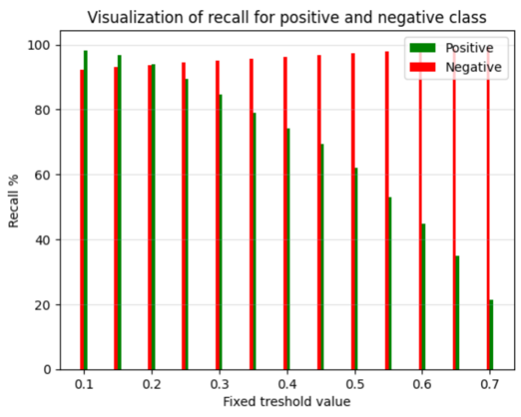
The aim of utilizing the Graph Neural Network is to reduce the number of grounded actions and facilitate planning algorithms in discovering the solution. To accomplish this goal, node classification is employed as a technique. This method involves classifying whether an operator node belongs to good or a bad class. PDG is a heterogeneous graph, so only a subset of nodes, namely the operators, are targeted for classification. Variables and Values are not involved in the classification process. The PDG is input into the GNN to construct a hidden representation of the nodes. Next, the operator nodes undergo a Sigmoid function to determine the probability of the operator (action) being either good or bad.

The prototype solution had a straightforward concept, only data regarding the initial and target states were embedded as features. These features are represented by binary values, denoted as $x \in 0, 1$. Since the *Problem Description Graph* is heterogeneous, various features can be embedded in different types of nodes. Therefore, only value-type nodes that are necessary for describing a state will contain this information.

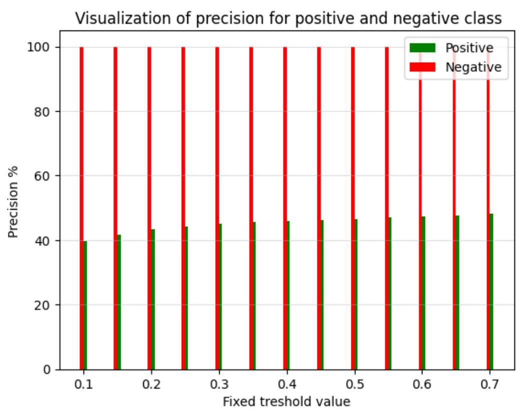
The GNN classification output provides the probability of an operator being beneficial for the planner. To classify a specific grounded action as positive, an additional factor is considered: the threshold. This is a predefined value that sets a boundary between positive and negative probabilities. The default value in the prototype solution is 0.5, indicating that any operator with a predicted probability greater than this threshold will be classified as positive.

Results

The prototype solution's experiments were carried out on three distinct planning domains. To evaluate the model's performance, two metrics, recall, and precision, were combined. The recall metric demonstrated that, in all three cases, the models accurately classified nearly 100% of good actions, while precision ranged from 40% to 50%. Furthermore, the model identified over 90% of bad actions as negative, indicating that many of them would be eliminated from the state search space. An example results can be seen on the figure below:



(a) Recall



(b) Precision

Figure 2.3: Satellite domain results

Opportunities for enhancements

While developing the prototype, various opportunities were identified that could significantly enhance its overall performance. The first improvement that could potentially enrich the prototype is incorporating additional features. The initial version was quite limited in terms of information passed to the model.

Planning domains exhibit varying characteristics, and the relations between their operators and values hold different significance, leading to different interpretations by the GNN. As a result, the models produced using only one type of architecture and message-passing function display different performances. The crucial aspect of optimization for any domain is utilizing hyperparameter tuning techniques. Testing and pre-selecting the optimal network components for a particular domain could considerably enhance the performance across different domains.

Another avenue for enhancing the solution lies in the integration of features. In the previous research, the prototype solution was trained solely based on the graph structure, which limited the exploitation of the GNN's full potential. By incorporating features into the training process, the overall performance of the tool can be greatly improved.

The final opportunity for enhancing the prototype involves post-processing the GNN output graphs. Following the classification, the operators are removed, but no further analysis is performed. This results in value and variable nodes left in the graphs with no connections or meaning. Furthermore, apart from basic graph modifications, more in-depth analysis could be conducted. In the planning setting, problems and their solutions adhere to strict rules. For instance, in the logistics domain, if the goal is to deliver only one package, only one truck is necessary to solve the instance. This leads to the constraint that operators concerning this specific truck are vital, while others can be removed. By identifying such constraints, it may be possible to significantly reduce the search state space for the planner.

The tool can achieve completeness by enhancing the prototype in the areas mentioned above. This innovative approach to automated planning optimization offers a promising alternative to existing solutions. Furthermore, this solution could be a significant contribution to the research field, inspiring future researchers to build upon the GNN approach or augment it with supplementary components.

2.4 PyTorch geometric

PyTorch Geometric (PyG) [42] is the chosen library for this project, as it is built on top of PyTorch [31], and is designed to facilitate the development and training of Graph Neural Networks (GNNs) for various applications and structured data environments. PyG provides a range of features for deep learning on graph-structured data, including a mini-batch loader that allows operations on many small and single large graphs, as well as GPU support. The batching functionality is particularly important for this project, as the learning process will involve many small problem instances, each producing its own small graph. This allows for optimization of the training process and significant improvements in performance.

Chapter 3

Features

3.1 Motivation

Simple landmarks and Relaxed Plan are the two methods presented below that introduce way of gathering information about a problem prior to execution of the plan search. They are commonly used in classical planning to guide the planner during its search in an efficient way. However, as it's possible to derive them prior to the execution of planning it allows for using them feature to machine learning models. There are successful applications in the area of planning that use them as inputs to machine learning models used to improve planning performance.

For instance, the Action Schema Network(ASN) [45] maps the current state of the planning problem onto a neural network and uses it to read out the "best" action to take in a given state. One particular challenge of this approach is the limited horizon of the network constrained by the number of hidden layers. This caused the under-performance of solutions when dealing with an arbitrarily long chain of actions and states. To address this issue, authors supplied the network with additional, domain-independent features derived from Disjunctive Action Landmark [19]. This particular landmark represents a set of actions from which at least one has to be used in any plan. Adding this type of knowledge to the features of machine learning model turned out to be enough to overcome the depth-fixed horizon limitations of the network.

3.2 Feature selection

The process of feature selection and feature engineering is a crucial part of any machine learning work. Careful selection of the information that is being fed into the model is the key to its success as it can vastly improve its predictive power. This chapter outlines the steps taken to enrich the information used by the tool as well as further ideas one could explore.

As previously stated, the former iteration of this project utilized GNNs to evaluate the usefulness of certain actions. The GNNs were successful to some extent - as evidenced by the improved search time of the planner through the built prototype as well as the classification metrics [13]. Nonetheless, they were limited by the use of a small number of features and heavy reliance on the network's topology. For these reasons, this iteration of the project puts more emphasis on enriching the model with valuable information. There are two types of information one can obtain from the Fast Downward tool by applying straightforward modifications which were determined as low-hanging fruit - *relaxed plan* and *landmarks*.

3.3 Relaxed plan

The term "relaxed" in the context of planning refers to a modification of a planning problem in which the delete list of, actions is removed. Given an instance, the corresponding relaxed version is created by copying the actions set and then removing all negative effects. Therefore, a relaxed planning problem is a version of the original in which it is not necessary to worry about deleted effects of the actions. Going further, the relaxed plan is the plan that solves the relaxed planning problem.

Formally, using the example of the PDDL task Π represented by a previously:

$$(P, A, \sum, I, G) \tag{3.1}$$

The corresponding relaxed planning problem is $(P, A^+, \sum, I, G)$, where the new actions set A^+. We can define positive $eff^+(a)$ and negative $eff^-(a)$ effects of action a for all actions $a \in A$, then a relaxed plan is a solution to a problem where:

$$A^+ = a \in A | eff^-(a) = \emptyset\} \tag{3.2}$$

Finding a solution to this relaxed problem assumes that executing an action on a binary state can only change its value from *False* to *True* and never the other way around.

The concept of relaxed planning problems presents several noteworthy characteristics that are applicable in the context of this project. Specifically, plans generated using a relaxed representation offer a reliable estimate of the original planning cost. Moreover, the length of these relaxed plans is commonly utilized as a heuristic function in planners. It is important to note, however, that the deletion of negative action effects in relaxed plans does not guarantee a solution to the original planning problem. Consequently, the removal of these deleted lists merely serves to simplify the problem, resulting in a significant drop in computational complexity. Therefore, the key takeaway from the characteristics of relaxed planning problems is that they can be leveraged to facilitate efficient planning in this project [51].

Consequently to summarize the advantages of the relaxed plan computation are:

- The relaxed plan simplifies the planning problems since ignoring the negative effects makes it easier to reason about the problem.
- It allows the planner to learn valuable information about the key-point propositions that need to be achieved in order to solve the problem.
- It is often a good approximation of the actual plan, serving as a reference point.

3.4 Landmarks

As mentioned previously in 2.1.2, a state in a planning problem is represented by a set of facts. Several facts which reason about the same object can be grouped into a variable, within the variable only one of the facts can be true at any given time. A landmark represents which of the values under a variable must be true at some point in the search in order to solve a planning problem.

The process of finding landmarks is based on analyzing the planning problem and identifying key propositions needed to achieve the goal. Note that one might also reason about actions needed to achieve these propositions and then create so-called Action Landmarks, however, let us further reference landmarks as the propositional version. Since finding actual landmarks is very complex there are several types of algorithms that compute them. The algorithms range in complexity and the information the derived landmarks carry. *Simple landmark* represents value assignment such as *at(truck1, location1)* to one variable and is most often represented as an index of a variable *var_xyz*. *Disjunctive action landmark*, the one that helped *ASN* overcome the limited horizon obstacles, is a type of landmark that is represented by a set of simple landmarks such that any plan has to use at least one action from that set. Finally, the *Conjunctive landmark* indicates that a set of simple landmarks have to be true at the same time at some point during the search in order to solve the plan. There are several algorithms which are used for Landmarks computation such as RHW [32] or Zhu/Givan [53] Landmarks. In this research for the landmarks computation, the Zhu/Givan was used.

Landmarks are typically used to help the planner break down the complex task of planning a problem into smaller sub-parts which then in turn allows the planner to focus on solving each of them individually. This leads to an efficient search by focusing on key goals that are more likely to be included in the actual solution.

3.5 Incorporation of Relaxed Plan

A relaxed plan has the same structure as an actual plan - both use the same object types, predicate definitions and similar action definitions (relaxed actions don't have negative effects). There are however slight differences between the two, typically the relaxed plan is smaller than the actual plan. Unlike a regular plan, actions in the relaxed plan are never repeated, since once a certain fact is set to *true* it will never change its value. A plan - a list of grounded actions - can be used to add a new feature to the **operator nodes**. The feature set of the **operator nodes** that previously was empty, now has been enriched by the binary flag indicating whether this operator is in the relaxed plan.

Fast Downward supports finding the relaxed plan out of the box and with a slight modification to the pipeline from the previous iteration of this project, the tool was able to output the list of the operators in the relaxed plan into a file. This file is in turn passed to the preprocessing pipeline that generates the graph constructs from the grounded representation of the problem. An additional feature has been placed on the **operator nodes** that is set to **False(0)** if the current operator is not presented in the relaxed plan, or to **True(1)** if it is. As finding the relaxed plan is very fast, this step did not increase the complexity and run time of the tool vastly and provided it with valuable information 6.6. Further deliberations of the pipeline will be described in chapter 5.

3.6 Incorporation of Landmarks

Similarly to the relaxed plan, landmarks also enrich the feature set of the Fast Downward tool out of the box. They can be mapped onto the **value nodes** of the PDG, and since one value is associated only with one variable it was decided to not include that landmark information into the variable node.*Simple landmarks* are then used to add a Boolean flag indicating whether a certain value is a simple landmark. Since the selected algorithm also supports Conjunctive Landmarks and Disjunctive landmarks, for the time being, the tool reads unique facts and creates a set of simple landmarks out of them disregarding the additional information about relationships between them.

Finally a set of *simple landmarks* is obtained. The additional feature is added to the **node values**. They are assigned with the additional feature equal to *True(1)* if the value is in the *simple landmarks set* otherwise *False(1)*.

Chapter 4

Hyperparameters optimization

The current parameters of the Graph Neural Network model were determined through manual testing, resulting in the selection of the best setting. However, this method does not fully capture the diversity and complexity of real-world planning tasks. As previously mentioned, planning domains often vary in complexity, as well as in the significance and knowledge captured by relationships between values. Therefore, a more sophisticated approach is necessary to tune the model parameters for different planning domains accurately.

The next step in developing a complete tool is optimizing the hyper-parameters of the model. This process involves identifying a set of distinct or continuous parameters that alter the behaviour of the model, along with their corresponding possible values. The goal is to find the best configuration of hyper-parameters with respect to some target function that represents the performance of the model.

Previous hyper-parameter tuning was carried out solely based on classification metrics, neglecting the performance of the tool as a planner. Although this approach was a reasonable starting point, optimizing hyper-parameters based on plan-related metrics is likely to lead to better results for the actual planner. For example, if a classifier identifies good operators only for one optimal plan but classifies operators from other optimal plans as bad, then the classification score of the model would be lower compared to the model that found good operators from all optimal plans however the actual end to end performance of the planner might not be worse.

More formally, let A represent all grounded actions for some problem instance and $P_1 \in A$, $P_2 \in A$ be two non-overlapping optimal plans, that is: $\wedge\{a \in P_1 | a \notin P_2\}$. Let $clf_1(a)$ and $clf_2(a)$ represent two classifiers as functions that given an action return a binary flag indicating whether an action is considered good or bad.

Assume that clf_1 only recognizes actions in P_1 as good, while clf_2 recognizes actions in both P_1 and P_2 as good. Thus, the classification score of clf_2 would be higher compared to clf_1 based on classification metrics. However, the actual planner might perform better when combined with clf_1 since it explores fewer states by only considering actions from P_1, which is a smaller yet sufficient subset of A. This shows the importance of hyper-parameter tuning with respect to planning metrics rather than solely relying on classification metrics for machine learning classifiers used in classical planning problems. Another upside of tuning the model this way is that it prevents over-fitting, with a simple trick of producing a very low score when the plan is not found at all.

4.1 Optimization parameters

4.1.1 Message passing functions

The message-passing functions in Graph Neural Networks (GNNs) are indeed critical for their performance. The selection of the most suitable algorithm depends on the network's specifications, such as the number of node features and graph properties, but also on features as the presence of communities. For example, the choice of algorithm can be heavily influenced by whether communities can be identified in the graphs. An excellent example of utilizing graph communities can be seen in the research paper authored by Mingxia Zhao and Adele Lu Jia, where they developed a Dual-Attention Heterogeneous Graph Neural Network [52].

Their research focused on Online Agricultural Question and Answering (Q&A) Communities that aim to help farmers obtain useful suggestions in the form of interactive dialogues. With a constantly growing number of users, traditional methods failed to provide satisfactory results. Therefore, by utilizing the Dual-Attention Heterogeneous Graph Neural Network model, which considers the influence of both the node type and different neighbours of the same type, they obtained better performance than state-of-the-art methods.

In the hyperparameter tuning process, different message-passing functions will be considered for the model, drawing inspiration from other researchers' work.

Sage Convolution - SageConv

A noteworthy message-passing function is discussed in the research paper that introduces GraphSage [14]. The name *Sage* is derived from the words *sample* and *aggregate*. Essentially the Sage Convlution was created for feature-rich graphs. However, it is proven by the creators that it is also suitable for structures with only structural features:

"... our approach can also make use of structural features that are present in all graphs (e.g., node degrees). Thus, our algorithm can also be applied to graphs without node features." [14]

Considering these factors, SageConv appears to be an ideal choice for this project. The plan is to incrementally add features, and the SageConv message-passing function is well-suited for this approach. Its ability to provide good performance throughout the project steps and produce comparable experiment results makes it a promising option.

In contrast to a conventional graph convolution where the aggregation function is fixed, the aggregators in GraphSage are trainable parameters. As a result, each convolution has its own aggregator, and a Graph Neural Network (GNN) of depth N consists of N trainable aggregators. The trainable aggregator weights are then utilized to produce the hidden representation of "neighbourhoods". To obtain the node's feature representation, the neighbourhood representation is merged with that node's features. This is a crucial concept as it enables GraphSage to be used in an inductive setting. This technique presumes that immediate neighbours of a given node should have comparable embeddings (community). The knowledge gained from the aggregators is carried forward to subsequent iterations, and as they interact with the same type of graph, they gain more experience, resulting in a learning model. This approach was a good match for the prototype as it had a tendency to perform well on unknown graphs on which the model was applied on.

Graph Attention Network - GATConv

When constructing a heterogeneous Problem Description Graph in the planning environment, numerous relationships can be revealed. To leverage these relationships, various message-passing functions can be employed. One example of such a relationship is the proximity of critical operators and facts on the graph. Though the edge count between them may not be high, this reveals detectable communities within the graph.

In Graph Neural Networks, a graph community refers to a group of nodes that are densely connected within themselves and relatively sparsely connected with the rest of the graph. Communities can be identified by detecting patterns in the graph's topology, such as the presence of subgraphs with high clustering coefficients and small characteristic path lengths. Graph communities are often used as a basis for designing message-passing functions in GNNs, as they can help capture higher-level graph structures and relationships between nodes. By leveraging graph communities, GNNs can achieve better performance in tasks such as node classification, link prediction, and graph clustering.

The attention-based architecture in Graph Neural Networks is effectively harnessed through the Graph Attention Network (GAT) [46]. GAT computes the hidden representation by attending over neighbours, thereby capitalizing on their relationships. This architecture is not only highly efficient, thanks to its parallelization across node-neighbour pairs but also adaptable to nodes with varying degrees, as it allows the specification of arbitrary weights to neighbours. Furthermore, the GAT model can be applied directly in the inductive setting, which is particularly valuable in planning, where the task involves classifying operators from various problem instances.

The attention algorithm in Graph Neural Networks accepts a set of node features as input, which is denoted as $h = \vec{h_1}, \vec{h_2}, \ldots, \vec{h_N}$, where $\vec{h_i} \in \mathbb{R}^F$. Here, N represents the total number of nodes in the graph, while F denotes the number of features associated with each node. With one layer, the algorithm generates a new set of node features, denoted as $h' = \vec{h_1}', \vec{h_2}', \ldots, \vec{h_N}'$, where $\vec{h_i}' \in \mathbb{R}^{F'}$.

To enable node-level feature transformations in the Graph Attention Network, a learnable linear transformation is applied to each node. This transformation is parameterized by a weight matrix $W \in \mathbb{R}^{F' \times F}$. Following this, the self-attention mechanism is performed on the transformed features, resulting in the computation of coefficients:

$$e_{ij} = a(W\vec{h_i}, W\vec{h_j}) \tag{4.1}$$

The coefficients computed in the Graph Attention Network represent the relative importance of the features of node j with respect to node i. This is the fundamental computation carried out by the algorithm. However, by incorporating the graph structure information in the mechanism, a more advanced approach called *masked-attention* can be employed. This method only computes the coefficient e_{ij} for nodes j that belong to the *neighbourhood* of node i, denoted as N_i.

In the initial prototype, solution discussed earlier, only the graph structure was utilized without any incorporation of features. Hence, any attention network would not have been effective. However, as discussed in Chapter 3, relevant features have since been added to the graph, representing important states and actions for the plan computation. These features correspond to significant nodes in the graph, such as *landmarks* and *relaxed plans*. Therefore, it is anticipated that the

GATConv will successfully leverage this information.

4.1.2 Training set selection

To effectively train a Graph Neural Network, it's necessary to have a set of labelled grounded operators that serve as the target feature for the network's predictions. In order to obtain this set, a planner such as *Fast-downward* [16] must be employed to solve a planning task and generate a plan. The operators that are included in the resulting plan are then naturally labelled as "good operators".

However, the number of operators to explore while computing the plan, far exceeds the operators contained within a single plan, leading to a significant imbalance between the number of positive samples (i.e., good operators in the plan) and negative samples (i.e., all other operators). This imbalance can pose a challenge for the GNN's training process, potentially resulting in lengthy training times or even hindering the network's ability to learn valuable weights.

To enhance the training set for the GNN, one solution is to adjust the parameters of the *Fast-Downward* planner which changes the computation method. By running the planner with these additional parameters, it performs more computations and explores all possible routes to solve the instance, rather than searching for a single plan. This yields a larger set of good operators consisting of all operators that are part of all computed plans. By considering all possible plans, the number of positive samples increases, thereby reducing the imbalance between positive and negative samples. However, even with the larger number of positive samples, the imbalance remains a significant challenge. This issue has been addressed in the previous project through measures such as implementing sample weights [13].

Obtaining a set of good operators for a given problem can be challenging due to the computational complexity involved. Exploring all possible routes to solve a task is much more computationally expensive than following a heuristic function to quickly reach the goal. Consequently, it may not always be feasible to perform such computations within reasonable time frames, which can limit the number of instances that can be included in the training set for the GNN. This issue is particularly pronounced when dealing with larger problems. Assumption relies on the fact that data obtained by running the planner (in perhaps a sub-optimal way) is better than a lack of data on this problem.

To enhance the training set for the GNN, a combination of the approaches mentioned above has been proposed. This results in a dataset with mixed origins of target features, with the underlying idea that the more instances of training data the GNN is exposed to, the better the results will be. Technically, this is achieved by first obtaining plans for all available instances, followed by a selection process. The candidates for obtaining good operators are selected based on computation times, with instances that can be solved in under 30 seconds being eligible for inclusion. This training set approach ensures a broader range of problem instances than using the good operators approach alone, thereby enriching the set of data that can be used for GNN training.

Despite the advantage of having a larger training set, this solution introduces an underlying issue. Feeding two different types of training instances to the GNN may hinder the network's ability to learn effectively. Although both approaches leverage the same type of relations and result in similar graph representations, the relationship between good operators and the overall number of operators differs between the two types of instances. This could introduce unnecessary noise to the GNN. To ensure the best performance of the output model, three different types of training will be performed. Two of these will use the initial approaches described above, one with computed plans and one with good operators. Additionally, a model with mixed data will be created. This approach ensures the best performance for different types of domains, as it enables the selection of the best candidate from a range of options.

4.1.3 Model parameters

The structure of the Graph Neural Network influences the performance significantly. The too-simple network will not be able to capture the relations as the expressive power will not be enough. On the other hand, too complex a structure will significantly extend the training process and can even lead to a situation when the network will learn the noise of the data instead of the desired properties. Moreover, in the Graph Neural Network, another parameter that can influence the performance is the aggregation function. In the general equation 2.3 it is denoted as permutation invariant $\bigoplus$. The simple aggregation can be denoted as:

$$a_u^{(t)} = AGGREGATE^{(t)}(\{h_v^{(t-1)} : v \in N(u)\}) \tag{4.2}$$

The neighbour aggregation of the node u in t-th GNN layer is expressed by aggregating the neighbour V hidden representation h_v in the previous layer $t-1$. $N(u)$ is a set of nodes adjacent to u [49]. The choice of the aggregation function is crucial. In the case of classification the node representation in the final layer $h_u^{(t)}$ is used for prediction. Therefore the last aggregation is called *READOUT* and outputs a probability of an operator belonging to the positive class. READOUT can be a simple permutation invariant function or a more sophisticated pooling function [50]. In this case, as the binary classification is performed the READOUT function is not parameterized and is set to *sigmoid*.

$$h_u^t = READOUT(h_u^{(t-1)}) \tag{4.3}$$

In the message-passing functions, the aggregation functions play an important role. Many researchers demonstrate that the specific choice of aggregation function contributes significantly to the representational power and performance of the model [49] [14] [41] [5].

To account for all the considerations mentioned above, a hyperparameter optimization approach was implemented. The optimization tool, which will be described later in this chapter, takes a set of pre-selected parameter options as input. The network size is determined by specifying the *hidden size* and *number of layers* parameters, which dynamically create the network structure. These parameter values were chosen empirically based on manual experiments conducted during the prototype development phase. The selected layers' numbers are {4,5,6,8} and the hidden sizes are {4,8,16,32}. Although initial runs had the option to use the sizes 64, 128, and 256, it turned out to be very slow to train and inefficient. Besides the network size specification, the parameters such as batch size and initial learning rate were passed to the tool.

On the other hand, the choice of aggregation functions included in the optimization process is based on information from the *torch geometric* documentation. For instance, the *mean* aggregation function is suitable for capturing the distribution of properties or proportions of elements, while the *max* function is advantageous in identifying representative elements. Alternatively, the *sum* function enables the learning of structural graph properties [49] [43]. Since all of these functions are considered valuable in the planning setting, they will be included in the set of parameters for the optimizer.

4.1.4 Training parameters

When training causal neural networks and graph neural networks, the choice of training parameters can significantly impact the training performance. Similarly, in this project, while optimizing models for a given planning domain, the optimal values of training parameters may differ. To achieve the best possible optimization results, the SMAC tool is used to set training parameters, such as the learning rate and optimization parameters. This subsection focuses on optimizers, as they have the most significant impact on learning speed. Choosing the right optimizer can greatly improve training results, especially when a constant number of epochs is used in training. In this project, two optimizers were considered: Adam [24] and

RMSProp [18].

Stochastic gradient-based optimization holds significant practical significance across various domains in science and engineering. Numerous problems in these fields can be framed as optimizing a scalar objective function, where the goal is to maximize or minimize it by adjusting its parameters. Gradient descent, an optimization algorithm, leverages the gradient of the objective function to explore the search space effectively. Therefore, optimisation algorithms such as the two mentioned above play a great role in training performance.

RMSprop, a gradient-based optimization technique introduced by Geoffrey Hinton [18], addresses a common problem encountered in training neural networks. When complex functions, such as neural networks, process data, they tend to exhibit either vanishing or exploding gradients. This issue becomes prominent in networks with numerous hidden layers that employ activation functions like sigmoid, as the derivatives are multiplied together during backpropagation. Consequently, the gradient diminishes exponentially as it propagates through the layers, resulting in ineffective updates to the weights and biases of the initial layers. Since these initial layers play a crucial role in recognizing the fundamental elements of the input data, this problem can lead to overall inaccuracies in the model [47].

To overcome this challenge, RMSprop employs a mini-batch learning approach. It calculates a moving average of squared gradients to normalize the gradient, ensuring a balanced step size or momentum. By decreasing the step for large gradient updates and increasing it for smaller steps, RMSprop prevents explosions and mitigates vanishing gradients. Additionally, RMSprop incorporates an adaptive learning rate, dynamically adjusting it over time to optimize the learning process effectively [36].

The Adam optimization algorithm has emerged as a leading choice for deep learning tasks, offering improved performance over traditional stochastic gradient descent. It combines the strengths of two algorithms, AdaGrad [6] and RMSprop described above, making it an adaptive gradient descent method. Adam, as described by its creators, is a highly efficient stochastic optimization technique that utilizes first-order gradients while minimizing memory requirements.

AdaGrad, one of the precursor algorithms, adjusts the learning rate on a per-parameter basis, making it particularly effective for sparse gradients commonly encountered in tasks like natural language processing and image recognition. On the other hand, RMSprop optimizes the learning rate by utilizing a moving average of squared gradients. By combining the favourable aspects of both AdaGrad and

RMSprop, Adam achieves enhanced computational efficiency, handles sparse gradients in noisy data effectively, and demonstrates excellent performance on large datasets.

According to Andrew Ng [29], Adam can be thought of as a fusion of RMSprop and Momentum techniques. This amalgamation of strategies contributes to Adam's remarkable versatility and efficacy in various deep-learning applications. Additionally, Adam's ability to handle sparse gradients and its computational efficiency makes it a preferred choice among practitioners in the field.

4.2 SMAC optimization

SMAC, known as sequential model-based algorithm configuration [25], is a versatile tool that effectively optimizes algorithm parameters. It can be applied to various scenarios, including automating processes, evaluating functions such as simulations, and adjusting parameters accordingly. To better comprehend the distinctions between Sequential SMAC and standard model configuration techniques, it is necessary to introduce two well-known alternatives for tuning the model:

- Grid Search, which computes the product of all possible combinations and while it guarantees optimal solution it is very complex and unfeasible in real-life scenarios.
- Random Search solves the complex issue of the Grid search however it does not guarantee an optimal configuration and blindly looks for the best result.

Both of them could be categorized as "model-less" optimization solutions that perform their search solely based on predefined *exploration*. There exists no feedback loop that could enhance the search: results computed for each run of the configuration are used only to compare it current best model. This is highly inefficient as the resources spent on obtaining the performance of certain configurations are not fully utilized.

Sequential Model-based Optimization (SMBO) which is the premise of SMAC is an approach that uses a probabilistic search to efficiently search for good solutions in a computationally expensive function space. Its power comes from an intelligent configuration search that neither checks all possible configurations nor explores them only at random. Instead, SMBO uses Bayesian search to model the probabilities of choosing certain configurations and updates these probabilities given the evidence. The evidence, a numerical value representing the performance of the model, is analyzed separately for each of the parameters. SMBO obtains the evidence by choosing a configuration, obtaining its result to them choose the best values for certain parameters.

More formally, let a configuration Y represent a set of parameters $w \in W$, with domain of the parameter w equal to $D(w) = \{v_1, v_2, ..., v_n\}$, well performing value v_{good} and under performing value v_{bad} for a certain parameter w, then probability of selecting these values have a following relation $P(v_{good}) > P(v_{bad})$

In order to prevent staying in local optima SMBO implements an exploitation/exploration balance that allows the model to stay on track of good values but at the same time try unexplored configurations.

- exploration - trying out configurations at random, with a goal to possibly find a totally new, best-performing configuration and leave the local optimum.
- exploitation - trying out configurations that yield good results, given the evidence on what is believed to be good

The first significant application of SMBO was developed in 1998 [23] and has been used as a premise for several optimization approaches: SPO[2], SPO+[21], SPO-TB [22] as well as SMAC [20]. SMBO has been applied to a variety of optimization problems, including hyper-parameter tuning, experimental design, and reinforcement learning. One of the most popular SMBO algorithms is the Sequential Model-based Algorithm Configuration (SMAC) algorithm, which is widely used for optimizing the hyper-parameters of machine learning algorithms. SMAC works similarly but has some additional features that allow it to exploit promising configurations faster

4.3 Training

In order to use SMAC one needs to provide it with configuration parameters, their possible values, and a way to evaluate a configuration. As mentioned previously, the optimization focus of this project included the evaluation of the actual end-to-end planner - using the GNN-enhanced Fast Downward tool. In order to achieve that a set of planner-related metrics has been set.

Since the primary goal of the tool is to solve planning instances, in the case of failure to do so, a disproportional high punishment was introduced. This was supposed to prevent overfitting and make sure that SMAC prioritizes configurations that solve the planning problems over the ones that correctly classify the good operators. In the case of the planner successfully obtaining the plan, the main metric includes the number of operators that were provided to the planner. The objective of the tool is to minimize this value. This was supposed to enforce the GNN to disregard as many operators as possible while keeping the plan solvable (previous guard)

A set of problems has been selected for training and evaluation. Unlike many optimization techniques, SMAC does not use the entire evaluation set to evaluate a single configuration. This would be very inefficient since it would require running the entire planner for all of the evaluation instances - presumably giving the optimizer less time to explore different configurations. Instead, SMAC chooses 1 problem at a time and keeps track of a database of performances for each of the problems. If comparing two configurations becomes problematic, more instances are used to find the best one.

Using SMAC allowed to significantly automate and control the end-to-end learning process of the GNN while keeping in mind the performance of the planner.

4.4 Iterative training process - practical details

Solving hard planning problems might sometimes overwhelm the planner. It can end up in a rabbit hole - get stuck in a part of the search space where The solution is hard to find. Fortunately, SMAC supports evaluation time limits. Utilization of this feature played a significant role since it allowed the training to predefine a maximum time that can be spent on evaluating a single instance before killing the planner and returning the disproportional punishment. Since the tool was submitted for the international planning conference competition [38], where it could run for 72h hours before being used for planning it was important to make sure that the knowledge of SMAC is periodically saved in case of any disruptions. In practice, every 5 hours the execution of SMAC is temporarily paused and the best configuration model is saved into a persistence state.

4.5 Configuration

In conclusion, optimizing hyperparameters, and selecting suitable message-passing functions, training sets, and GNN structures are essential for achieving accurate and effective performance in planning domains. By considering these factors and leveraging planning metrics, the GNN model can be tuned to enhance the planner's performance and deliver improved results in real-world planning tasks. Consequently, all the aforementioned parameters will be incorporated into the SMAC optimization tool for monitoring and analysis within the model's specific planning domain. The insights gained from these reflections may contribute to further optimization improvements and a better understanding of best practices.

Chapter 5

Pipeline implementation

The objective of the tool under development is to effectively categorize operators into good and bad target groups. To achieve this, an essential initial step involves integrating the tool with the planner. This chapter will elaborate on the pipeline and delve into the design choices made during the development process, informed by valuable insights gained from extensive testing and observation.

5.1 Preprocessor implementation

The Graph Neural Network (GNN) tool serves to optimize planning problems, making them more amenable to efficient solving by the planner. Its primary function is to perform pre-processing of the tasks and must be integrated into the original planning pipeline of Fast Downward(FD). To facilitate smooth integration, we leverage the Scorpion planner [37], a classical planning system built upon FD, which offers a range of plugins to enhance its capabilities.

One noteworthy plugin is the ability to incorporate a preprocessor into the planning pipeline. When invoking the Scorpion planner with appropriate options, it initiates the problem translator, responsible for grounding, followed by the execution of the designated preprocessor. The preprocessor takes as input a STRIPS representation of the planning task, identical to that provided to the planner itself. By default, the Scorpion planner includes the h^2-preprocessor [1], which performs comprehensive problem analysis and optimization. It excels at eliminating redundant operators and identifying unreachable states. However, the framework allows for the implementation of custom preprocessors to suit specific needs.

Consequently, our objective entails the development of a script that orchestrates the execution of tailored components for problem optimization. The execution sequence of the planner can be outlined as follows:

Algorithm 1 Planner execution

```
Require: $problem.pddl, domain.pddl$
1: $output \leftarrow translator(domain, problem)$          ▷ Compute STRIPS representation
2: $output \leftarrow preprocessor(output)$ ▷ Preprocess the output from translate module
3: $plan \leftarrow search(output)$            ▷ Compute plan from the preprocessed output
```

The intuition is to combine the advantages of both preprocessors the h^2-preprocessor and GNN. The functionality of the h^2-preprocessor when combined with GNN brings additional value to the solutions as the trained model is only capable of removing the operators. However, no further analysis of values or variables is performed. Consequently, by combining the two components as the result they create a complete optimization tool.

The initial implementation of the preprocessor follows an intuitive three-step process. Firstly, the h^2-preprocessor is executed to analyze the planning task and perform initial optimization. Subsequently, the output of the first step is passed to the GNN, which classifies the operators. Finally, once the identified operators are removed, the h^2-preprocessor is rerun on the GNN's output to eliminate redundant and unused values and variables.

By following this approach, we aim to achieve optimal results. The h^2-preprocessor contributes to initial optimization, while the GNN's operator classification enhances the selection of operators for removal. The subsequent execution of the h^2-preprocessor on the GNN's output ensures further refinement by eliminating redundant values and variables. This solution, based on intuition, is expected to yield the best possible outcomes.

Algorithm 2 Initial implementation of the preprocessor

```
Require: $output$                          ▷ STRIPS representation from translator
1: $output \leftarrow h^2 - preprocessor(output)$
2: $output \leftarrow GNN(output)$                                    ▷ Classification
3: $output \leftarrow h^2 - preprocessor(output)$
```

As the algorithm shows the components in the preprocessor script update the same *output* file which is afterwards passed to the planner.

5.1.1 Initial insights

The primary objective of the developed optimization tool was twofold: to accelerate plan computation and to optimize problems that were previously unsolvable due to their size, making them feasible for planners to handle. However, the initial setup proposed for the tool, based on preliminary testing, proved to be inadequate in achieving these goals.

During the preprocessing stage, striking a balance between the time allocated for preprocessing and the subsequent planning phase is crucial. It is essential to consider that exhaustive preprocessing, despite potentially yielding better overall optimization, can often result in worse outcomes in terms of the total time spent on solving the problem. Therefore, although the initial setup was based on a theoretically sound idea, it could not meet the time constraints in practice.

As a result, further adjustments were necessary to find the appropriate balance between preprocessing and planning. These adjustments aimed to optimize the overall efficiency of the process, taking into account the time limitations. The objective was to strike a balance where the preprocessing stage provided sufficient optimization while ensuring that the total time spent on the problem remained within acceptable limits.

The initial experiments revealed that the first application of the h^2-preprocessor did not yield significant optimization improvements for the initial problem versions. Moreover, as the problem size increased, the computational time consumed by the preprocessing step became disproportionately large, overshadowing other crucial stages of the process. Consequently, it was decided to eliminate the h^2-preprocessor from the initial step of the optimization pipeline.

Another significant factor that had a substantial impact on the performance was the classification threshold, which was previously discussed. The threshold acts as a boundary for the sample probabilities. Any probability above the predefined threshold is classified as positive, while lower probabilities are classified as negative. In order to minimize the number of errors, it is ideal to set a higher threshold value. However, it is important to note that the optimal threshold may vary depending on the domain or even within different problems in the same domain.

Initial experiments revealed that a default threshold value of 0.5 yielded good results for small problem instances (below 500,000 operators). However, when dealing with significantly larger instances (exceeding 5 million operators), it became necessary to fine-tune the predefined threshold. To provide a clearer understanding of the issue, instead of directly referring to threshold values, we will employ the percentage of operators classified as positive. In smaller problems, the tool typically selects between 5% to 20% of the actions for further processing.

However, in the aforementioned larger planning instances, this percentage drops to nearly 0% or 1%. Consequently, there is a significantly higher likelihood of missing an important operator during the classification process, as observed in several cases. To tackle this problem, the concept of retries was introduced to address the issue at hand.

The retries algorithm plays a crucial role in determining the completion of future steps and ultimately determining whether a plan has been found or not. When a solution cannot be reached with the current set of operators, the algorithm performs a rollback and passes a larger set of operators to the planner. This ensures that even if the initial threshold value was insufficient for a particular problem, it is adjusted to a lower value so that more operators are classified as positive, eventually leading to a solution.

To optimize performance and minimize unnecessary planner runs, the threshold is initially adjusted to ensure that the proportion of operators classified as positive is not lower than 10%. Subsequently, based on a parameter passed, a certain number of additional retries are computed. By default, three retries are performed, and the results are stored in separate files, ready to be used and passed to the planner. The number of operators included in each consecutive retry increases by 10%. Therefore, if the initial run with 10% of the operators fails, the algorithm rolls back and utilizes 20% and 30% respectively in subsequent retries. The preprocessor algorithm, incorporating these implemented changes, is presented as follows.

Algorithm 3 Preprocessor retries computation

```
Require: $output, threshold, retriesNum$          ▷ output - STRIPS representation
 1: $defaultPredictions \leftarrow GNN(output)$
 2: for $i$ in $defaultPredictions$ do
 3:     if $defaultPrediction[i] \geq threshold$ then
 4:         $predictions.add(defaultPrediction[i])$
 5:     end if
 6: end for
 7: $operators\% \leftarrow predictions/totalNum$
 8: if $operators\% \leq 10\%$ then
 9:     $predictions \leftarrow$ top 10% of defaultPredicitons
10: end if
11: $r \leftarrow 0$
12: for $RetriesNum$ do
13:     $gnn_retries\{r\} \leftarrow$ top $10\% * r$ of defaultPredicitons
14:     $r++$
15: end for
```

This algorithm precomputes the initial operators' predictions and additionally produces multiple files storing predictions with higher thresholds for future eventual failures.

Algorithm 4 Planner execution with retries

```
Require: $problem.pddl, domain.pddl, retriesNum$
 1: $output \leftarrow translator(domain, problem)$
 2: $initial\_output \leftarrow output$
 3: $output \leftarrow preprocessor(output)$
 4: $plan \leftarrow search(output)$
 5: $retry \leftarrow 0$
 6: while not $plan$ or not $retry = retriesNum$ do
 7:     $plan \leftarrow search(output, retry)$
 8:     $retry + +$
 9: end while
10: if $plan$ then
11:     return plan
12: end if
13: $plan \leftarrow search(initial\_output)$        ▷ search with original STRIPS without
    preprocessing
```

The algorithm presented above demonstrates the process of computing a plan using retries. The preprocessor analyzes the information provided, selects relevant predictions, and passes them to the h^2-preprocessor for further optimization. The planner then attempts to find a solution based on this refined input. If the number of retries exceeds a certain threshold, the planner resorts to using the original STRIPS from the translator component to continue the search for a solution.

Upon testing the new implementation, the tool encountered an additional issue related to the h^2-preprocessor, which is executed after the GNN classification. In this preprocessing step, the objective is to remove unused values and variables from the graphs. By postprocessing the classification, which considers only the operators with the h^2-preprocessor, the planner would be presented with an even simpler task. However, due to the time spent analyzing certain problems, the processing sometimes exceeded the pre-set time limits.

To address this challenge and for experimental purposes, two potential solutions have been proposed, both of which will be tested. The first implementation will completely exclude the use of the h^2-preprocessor and directly pass the classification results to the planner. The second algorithm will incorporate time limits on the h^2-preprocessor since it is an iterative process. This approach allows for stopping the preprocessing at an intermediate state, even if it is not fully optimized. This approach could still yield valuable results and potentially overcome the time constraint issue.

5.1.2 Final implementations

After a thorough investigation of the issues encountered during the preliminary testing phase, several adjustments were made to the aforementioned tool. The final versions selected for the experimental phase have demonstrated the highest performance and have shown the most promising potential based on empirical evidence. As a result, the following configurations will be further tested:

1. Scorpion planner with the h^2-preprocessor as a base for comparison
2. Scorpion with preprocessor which uses only GNN classification and retries logic
3. Scorpion with preprocessor which uses GNN classification followed by h^2-preprocessor with set time limit and retries logic

Through the comparison of these configurations, it will be possible to not only identify the best-performing model but also understand how different components influence the computation of plans. Furthermore, the diversity of problem complexity during the experimental phase will provide insights into how these configurations scale with larger instances. By analyzing the results, it will be possible to determine which of the configurations are the most effective within the given context. This evaluation will contribute to a better understanding of the tool's capabilities and guide further improvements.

Chapter 6

Experimental Settings

To conduct meaningful experiments, the first step was to establish baseline results using the state-of-the-art planner, Fast Downward. Thanks to the custom implementation of Scorpion planner the pipeline including $h^2 - preprocessor$ and the Fast Downward was created. The planner's settings were kept consistent in both the baseline and subsequent experiments to ensure clear and accurate outcomes.

During the experimentation phase, the GNN is run as a preprocessor and optimises the planning task. The planner is then executed using the same settings as in the baseline development, and the results are compared using various metrics such as the number of operators in the task and the computation time for the plan. Further details on these metrics will be provided in the upcoming section of this chapter.

The experiments are run using a tool that allows running and evaluation planners on multiple problem instances automatically. The *Lab* [39] is a Python package suited for planners benchmarking. Additionally, the tool includes code for parsing the results of the runs and creating reports. Thanks to that setup all experiments are run using the same setup and environment ensuring valid experimentation.

6.1 Data

To conduct the experiments four different planning domains were selected. They differ in complexity and will provide the best overview of the solution's potential. The domains are satellites, depots, barman and rovers.

The satellite domain deals with the modelling of a satellite observation scheduling problem. This problem involves using one or more satellites to observe objects in space. Each satellite is equipped with different instruments that have distinct characteristics and require appropriate calibration targets. The satellites can be directed towards various targets for observation. In the original version of the domain, there are considerations for downlinking the observation data to the station, but this is only possible within certain time windows. Moreover, the energy consumption factors were taken into account. The satellite relies on its battery for power and can be recharged using solar panels and the energy provided by the sun. In this version of the problem, the satellites consume energy while performing actions, and the instruments need to be warmed up before use. However, for simplicity, we will focus on the basic version of the domain, excluding energy and data downlinking constraints. The tasks are based on finding the most efficient covering of the observations given as a goal. [28]

The depots domain is another domain utilized for experimentation. It involves actions related to loading and unloading trucks using available hoists located at fixed positions. The cargo being handled consists of crates that can be stacked and unstacked onto a predetermined set of pallets at these locations. Unlike specific order requirements in the Blocks domain, the trucks in this domain do not hold crates in any particular order. This flexibility allows the trucks to function similarly to a table in the Blocks domain, enabling crates to be rearranged. The depots domain is considered more complex as it combines elements from two simpler domains, namely Logistics and Blocks. By merging logistics-related tasks with the stacking of blocks, this domain presents additional challenges and sub-problems. While the primary objective remains to achieve efficient delivery, the domain also incorporates smaller-scale problems. This complex domain serves as a valuable test set to evaluate the capabilities of the tool and assess its performance in handling more intricate problem scenarios. [26]

Another domain to consider is the realm of the Barman. Within this domain, an automated bartender takes charge, deftly handling drink dispensers, glasses, and a shaker. The objective is to devise a comprehensive plan for the robot, dictating its actions in order to fulfil a specified set of drink orders. In this domain, the removal of certain actions carries crucial information, reflecting the robot's inherent limitation of grasping only one object at a time. Moreover, it acknowledges the prerequisite that glasses must be cleaned of any liquid and contaminants before they can be filled. This is an example of a hard domain. the difficulty comes from the complex plans instead of the number of operators. [4]

One of the domains that will be employed is the rovers domain, which is a simplified rendition of the challenges faced by NASA during the Mars Exploration Rover missions launched in 2003. In this project, we will utilize a version of the problem that focuses on planning for multiple rovers equipped with different sets of equipment, which may have some overlapping capabilities. These rovers are tasked with traversing the surface of a planet, moving between designated way-points, gathering data, and returning to the lander. This is the simplest domain that will be used for experiments. [27]

The problems chosen for experimentation are obtained from a repository which is specifically created for storing viable benchmarks for automated planning [44]. This is a collection of benchmark sets generated with the Autoscale tool. The tool automatically computes a good scaling in task difficulty so that it is easy to compare the performance of different planners. Each domain mentioned earlier will be subjected to testing using 30 diverse problem instances. These instances vary in terms of the difficulty of finding a solution (measured by the number of goals to satisfy) as well as the size of the problem (measured by the number of objects involved). By incorporating this comprehensive approach, we can thoroughly assess the performance of the solution across all complexity levels. This will allow us to identify scenarios where the solution outperforms the base approach and those in which it may lag behind.

6.2 Training

The training as mentioned before is performed with the support of the hyperparameters tuning tool SMAC. However, to ensure comparable results the parameters such as time constraints and number of training epochs are kept constant. This approach enables to investigate of different domains in terms of the tool performance. Consequently, valuable knowledge will be obtained which in the future will allow optimization of the solution so that it shows similar performance across all domains. The number of epochs is set to 500. This is determined empirically based on the experiences from the previous project.

The chosen loss function for the training process is the *Binary Cross Entropy* (BCE) [12]. Based on the knowledge gained from the previous project [13], this loss function was found to be suitable for the current project. The primary objective of minimizing the loss calculated using the BCE is to reduce the amount of information required to represent the operators' labels and increase the predicted probability for the labels. By averaging the loss over the entire data set, the resulting model exhibits good average performance across all data points. Furthermore, the BCE can be a weighted loss function which enables this technique for optimizing unbalanced data sets. This is necessary in this case to preserve the model from predicting all samples as negative which typically in the unweighted environment would produce the best results for the model perspective. The introduced weights are calculated based on the relations of positive and negative samples.

$$W_0 = \frac{N}{N_0} \tag{6.1}$$

$$W_1 = \frac{N}{N_1} \tag{6.2}$$

In the above equations, the w_0 is the weight of negative samples and the w_1 is positive. The weights are computed dynamically for every dataset which ensures consistency of the sample's importance across all datasets.

To conduct the SMAC hyperparameter optimization, a separate dataset must be carefully chosen. The selection process is based on the time required by the planner to solve each planning instance. Therefore, data splitting occurs after all available tasks have been run through the planner. Subsequently, instances are filtered based on the time it took for the planner to solve them. The first filter removes instances solved in more than 120 seconds, and the remaining instances are then sorted by planner time. Preferably, instances with longer planner times are chosen. Additionally, the tool ensures that the number of selected instances is no more than 10% of the total number of instances. This filtering approach helps to select the most balanced (solvable but not trivially easy) instances for the optimization process, thereby improving the quality of the final model.

6.3 Experimental Metrics

During the previous iteration of this project, the prototype metrics were strictly focused on operator classification performance (BCE). The recall and precision were taken into account as only a limited scope of the tool was tested [13]. However, in this project, a full spectrum of the pipeline is taken into consideration. Therefore, to visualize the advantage of the preprocessor the two metrics that are the primary goal to optimize are taken into consideration. These are the number of operators passed to the planner, plan search time and coverage.

The experiments were conducted in three separate series, excluding the baseline run. As mentioned in previous chapters, the previously developed prototype solution was enhanced with a hyperparameter optimization tool and additional features. To track the improvements accurately, the components were added one by one. The first series involved only hyperparameter tuning. In the second series, the relaxed plan feature was introduced, followed by the addition of landmarks in the last series. This sequential approach enables precise monitoring of the progress of the experiment and enables the determination of the effectiveness of the enhancements.

6.4 Results of hyperparameters optimization

For all of the domains, a tendency for small and medium-sized networks could be seen. Big networks were inefficient and seemed to have way worse performance as well as the execution of learning was way slower. With smaller networks, it was possible to faster get the models and evaluate them with often better performance

The batch size of 16 seemed to have caused overfitting of the model in many cases since the classification loss was small however the search was reliant on many retries - initial predictions were not sufficient to solve the problem. With batch size 4 or 8, the classification loss was smaller but the models were more likely to solve the planning

During the hyperparameter optimisation process, the SAGEConv message-passing algorithm was chosen in almost all of the instances. On the contrary, the GATConv which is an attention-based message-passing algorithm was performing significantly worse. From such an outcome it can be concluded that an attention-based network was not a suitable choice in this scenario. The reason behind such poor performance could be the limited number of features in the network.

6.4.1 Problem with models selected by SMAC

One problem that could have been seen across all tested domains was the fact that the learning happens on relatively simple problems compared to the 30 selected benchmark instances. This has often caused SMAC to (correctly) choose models that perform worse on the benchmark set and better on the training/validation set. This was discovered by manually testing different models (that SMAC has not selected) and getting better (sometimes even much better) results.

6.5 Results of the experiments

To analyse the results in the clearest and most structured manner they will be described for all the domains separately. As the domains are different in solving complexity and in the size of the environment considering the number of objects in the problem instances, a one-by-one analysis will provide the best insights.

In order to enhance clarity regarding the results, it is essential to establish precise terminology for identifying specific configurations. The three primary configurations mentioned earlier will now be outlined using the following designations:

1. Scorpion planner with the h^2-preprocessor - **base**

2. Scorpion with preprocessor which uses only GNN classification and retries logic - **GNN**
3. Scorpion with preprocessor which uses GNN classification followed by h^2-preprocessor with set time limit and retries logic - **GNN H2**

Additionally, to separate the configurations using different features the prefixes will be added to the names:

- Model trained using Hyperparameeter Tunning without any features - **Raw**
- Model trained using Hyperparameeter Tunning with the relaxed plan as features - **Relaxed**
- Model trained using Hyperparameeter Tunning with both relaxed plan and landmarks as features - **Relaxed Landmarks**

The performance of only landmarks models was usually very poor and it was decided not to include it in the competing configurations.

6.5.1 Satellites

The satellite domain as described above is a simple domain to tackle for planners. The complexity grows based on the number of objects in the problem which is followed by a growing number of operators grounded.

Coverage

Coverage is a metric that indicates the number of planning problems successfully solved by the tool. The primary objective of the tool is to attain the highest possible coverage, as its purpose is to provide a solution that enables the planner to address previously unsolvable problems through preprocessing. To present an overview of the performance of all tested configurations, the coverage results will be showcased in a table.

Domain	Base	Raw	Relaxed GNN	Relaxed Landmarks GNN	Relaxed Landmarks GNN H2
Satellite	30/30	30/30	30/30	30/30	30/30

Table 6.1: Coverage Satellites

As the satellite domain is simple to solve, all the configurations achieved 100% coverage on the problems from the experimentation set.

Operators

The complexity of the satellite domain is primarily determined by the number of objects that define the number of all operators. Nevertheless, a significant portion of these operators are not utilized in the actual plan and can be considered redundant. The following figure illustrates the count of operators that are supplied to the planner from the base configuration. This configuration is then compared with the new configurations described above.

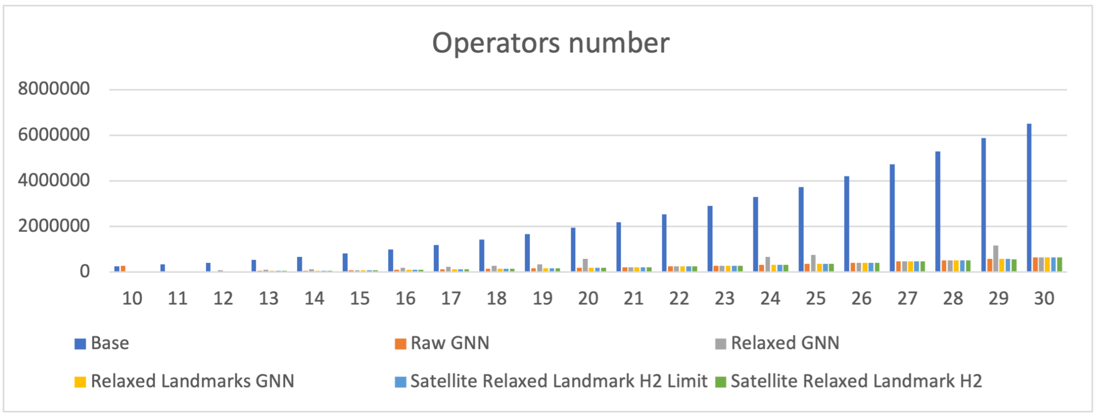


Figure 6.1: Number of operators for every configuration

The depicted figure showcases a subset of the problems, specifically those targeted for optimization. Hence, it only includes problems ranging from 10 to 30. Notably, the graph demonstrates that the number of grounded operators in the base configuration exhibits an exponential growth pattern, quickly surpassing millions shortly after problem 17. Conversely, with the aid of preprocessing, the operator count is substantially reduced, even for the largest problems, ensuring it does not exceed one million. Furthermore, it is evident that the variance in the operator count among different preprocessor configurations is relatively minor.

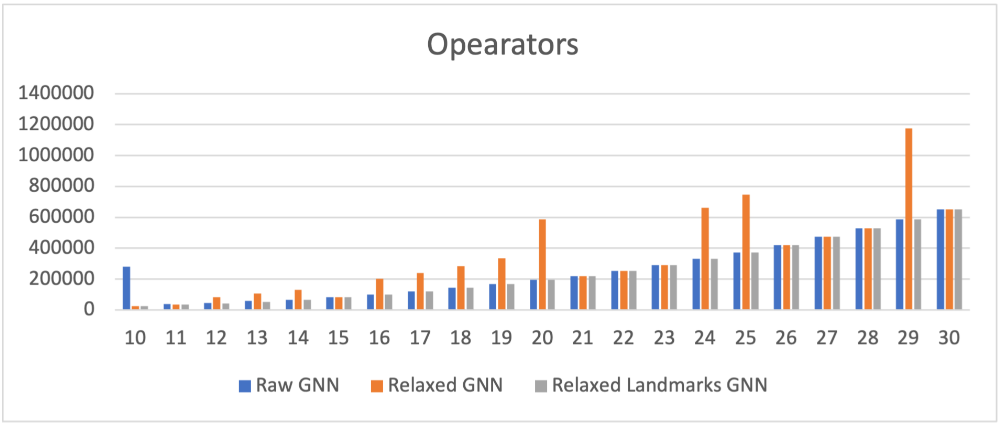


Figure 6.2: Number of operators for GNN configurations

The provided figure illustrates the comparative analysis of the classification performance among all GNN configurations. The experimental results demonstrate that the configurations exhibit nearly identical levels of accuracy. However, it is worth noting that one of the configurations, which incorporates a relaxed plan as a feature, underperforms in certain cases. Upon analyzing the model training and the outcomes, it can be deduced that incorporating a relaxed plan as an independent feature introduces unnecessary noise during training, leading to a decline in overall performance. Conversely, during the experimentation phase and through analysis of the training process, it was observed that the combination of relaxed plan and landmarks features yielded the best results.

As previously mentioned, the combination of GNN and h^2-preprocessor is expected to yield superior results. However, it is crucial to consider the time required for the h^2-preprocessor to optimize the output of GNN. Thus, a detailed examination of the optimization benefits it offers in relation to the time required for optimization becomes necessary. This investigation will provide a comprehensive understanding of the trade-off between the optimization advantages and the associated time constraints.

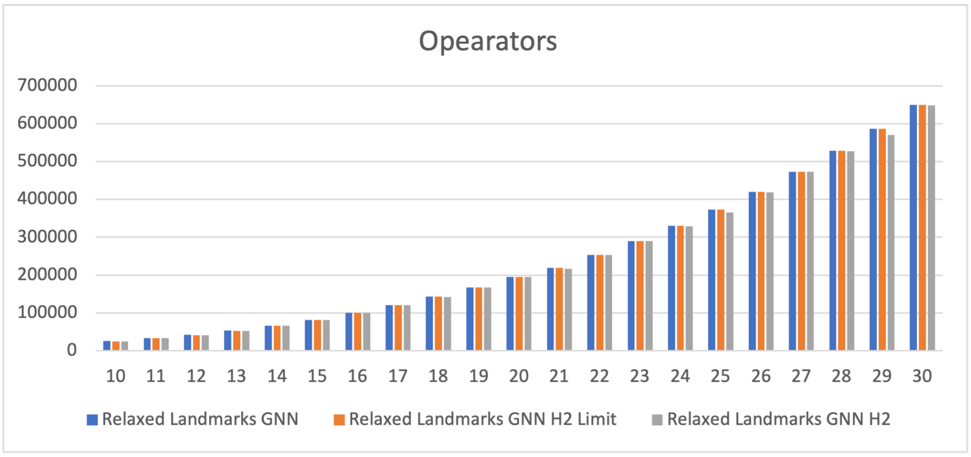


Figure 6.3: Number of operators GNN vs GNN H2 with time limits and without

The depicted figure demonstrates that the execution of the h^2-preprocessor on the output of GNN does not yield significantly improved results. However, it is important to note that the computational time required for this process significantly hampers the performance of the tool, which will be elaborated on in the subsequent subsection. The results from the figure indicate that employing the h^2-preprocessor with a predetermined time limit does not alter the number of operators in any of the cases. This is primarily due to the problem instances being too large to analyze even after classification performed by GNN. The same scenario is observed when employing the configuration without a time limit on the h^2-preprocessor. Slight improvements in optimization can be observed only in problems 22, 24, and 29; however, these differences are not significant. Based solely on the classification of operators, it can be concluded that running the h^2-preprocessor may not be worthwhile. Nonetheless, it is important to consider that the primary objective of the h^2-preprocessor is to eliminate unused values and variables. To assess its effectiveness in this regard, the planner's execution times must be taken into consideration, as outlined in the subsequent subsections.

Search time

Another metric taken into consideration for comparing the results is the search time. This metric specifically measures the time spent on planning, focusing solely on the search component. Translation and preprocessing times are not included in this calculation. Analyzing these results allows for an assessment of how the optimized problem instances influenced the performance of the planner.

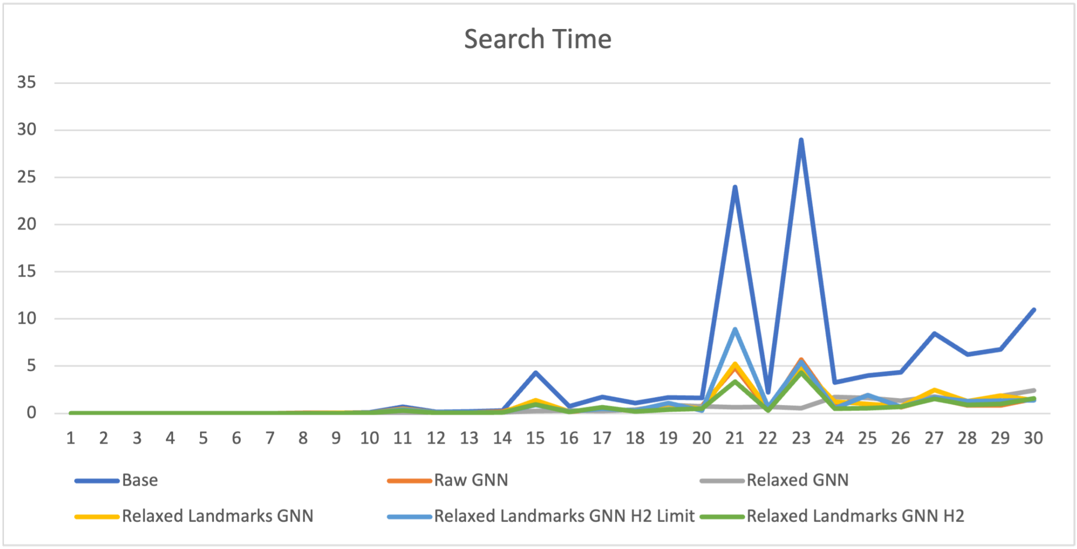


Figure 6.4: Search time of all configurations

The provided figure depicts the search times of all tested configurations. The axis Y shows the time required for the search to complete in seconds and the X is the problem. It is evident that the base configuration performs the poorest. It can be assumed that the search time using the base configuration will increase exponentially, particularly in the last six problems where the computation time tends to grow much faster than in the earlier part. During the experimentation phase, it was also discovered that problems 21 and 23 are considerably more computationally challenging than the others. Although it is difficult to determine the exact reason behind this complexity, as the number of objects in the instances and the goal conditions to satisfy are similar to other problems, it can be inferred that the routes to achieving the goals are more intricate. However, when examining the performance of the planner after preprocessing by the developed solution, significant improvements can be observed. Although there is still a noticeable spike in search times for problems 21 and 23, the overall search time scaling becomes linear. By excluding these two computationally challenging problems from the figure, the advantages of the preprocessor become more apparent.

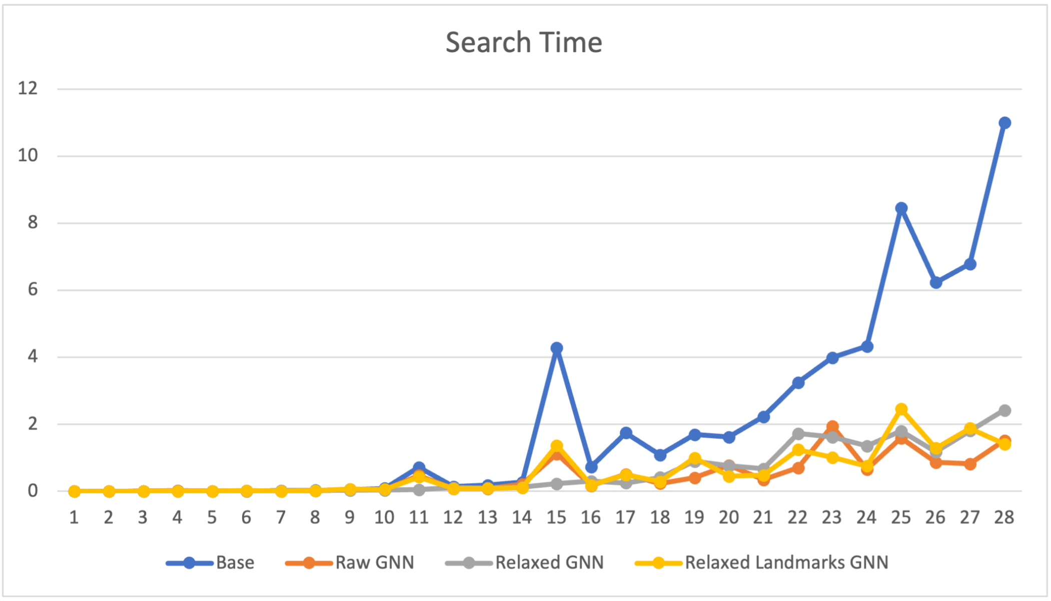


Figure 6.5: Search time excluding problems 21 and 23

Planner time

The final metric to consider is the planner time, which encompasses the total time required for solution computation. This metric incorporates the time taken by the translator, preprocessor, and search component. By comparing these times, we can evaluate how the new solution has influenced the overall planning process.

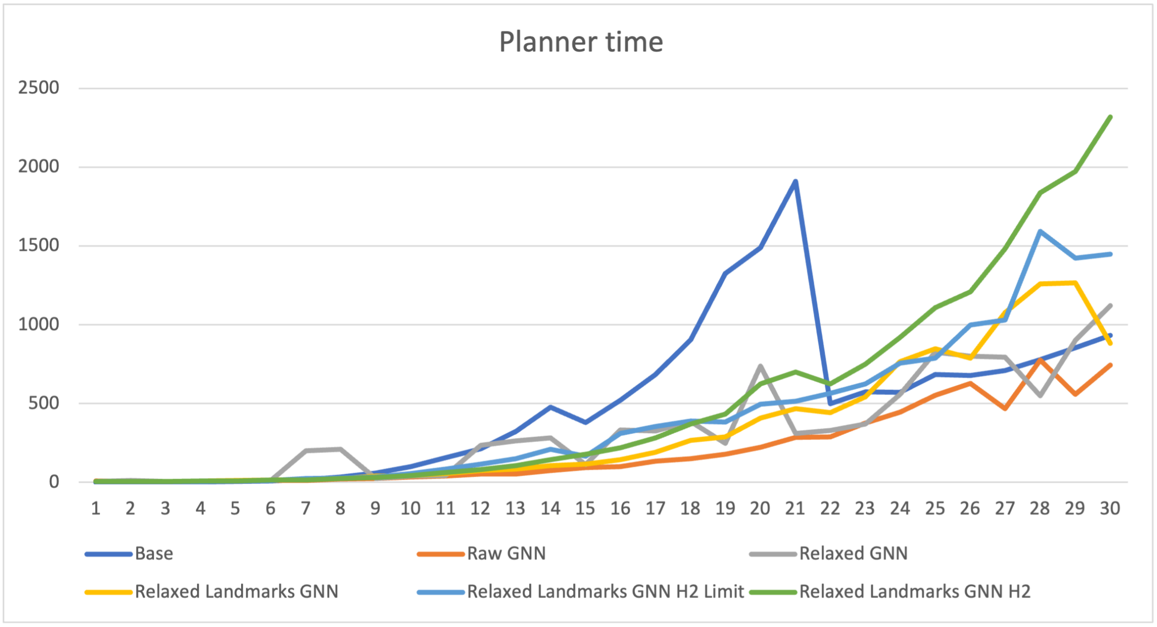


Figure 6.6: Planner time of all configurations

The presented figure compares the planner times achieved using all configurations. Similarly to the previous subsection, problems 21 and 23 stand out as outliers with significantly longer computation times compared to other cases. However, it is evident that the planners utilizing the developed solution perform better overall. Consistent with earlier findings, configurations employing the h^2-preprocessor after GNN classification lag behind in larger problem instances. To gain a clearer understanding of the advantages and for further investigation, let's divide the experiment into specific cases. Firstly, let's consider the comparison between the base configuration and the preprocessor without the use of the h^2-preprocessor.

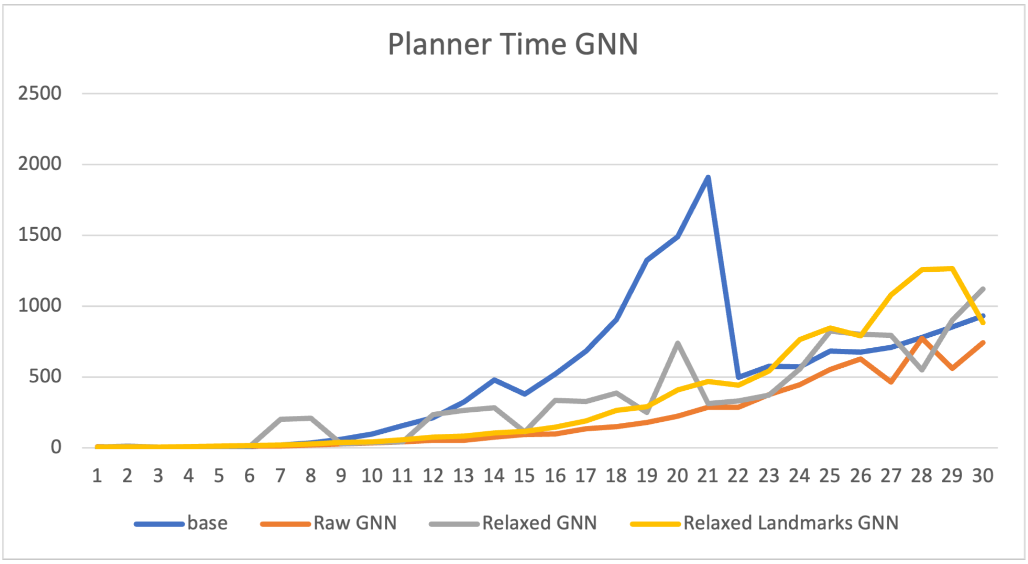


Figure 6.7: Planner time of base and GNN configurations

The presented figure showcases the performance of the base planner in conjunction with all the best-performing tool configurations. It is evident that the solution outperforms the base planner significantly in more challenging problem instances. This highlights the advantage of the solution when tackling more difficult scenarios. However, as the computational complexity decreases, the results become more similar. This can be attributed to the tool's implementation, as all tasks in the pipeline are executed using the Python programming language. Although Python is not the most efficient language for processing large files, which are commonly encountered in planning tasks, it explains why the planner times appear similar. Nonetheless, it is worth noting that refactoring and outsourcing the file processing to C++ code would greatly accelerate the entire process. Furthermore, it is noticeable that the configurations utilizing features exhibit slightly lower performance. This can be attributed to the extra computation required for relaxed plans and landmarks before preprocessing. While this computation does not have a significant impact on the overall process performance, it is still reflected in the results.

Another comparison that confirms the earlier assumption involves contrasting the pure GNN approach with the GNN combined with the h^2-preprocessor.

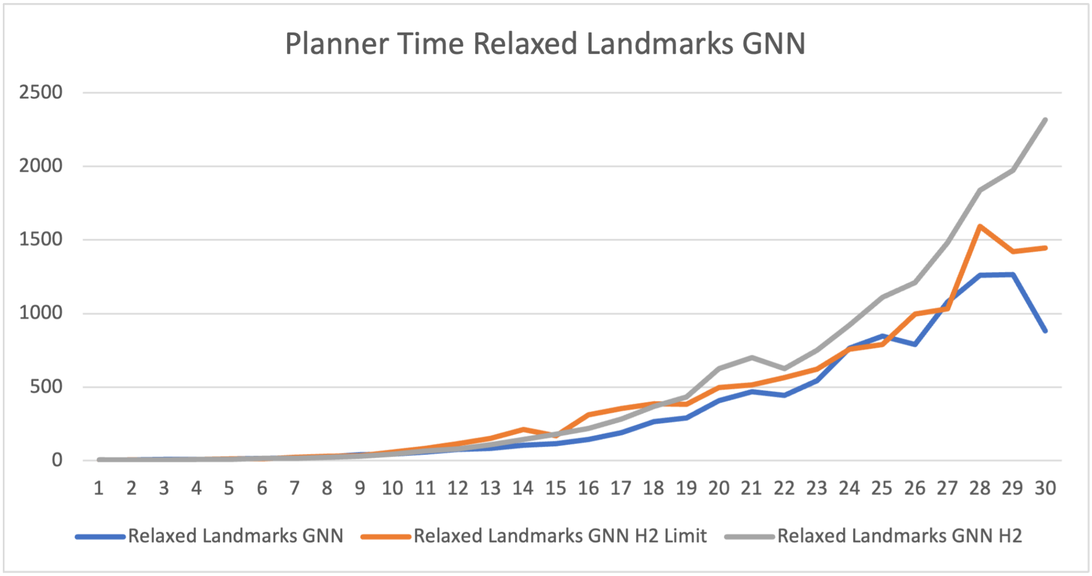


Figure 6.8: Planner time of GNN and GNN H2 configurations

As anticipated, the inclusion of the h^2-preprocessor results in increased planner times. Considering the previous results, which highlighted the number of operators passed to the planner, it can be concluded that the additional processing by the h^2-preprocessor is not optimal in terms of overall performance for this domain.

Bigger problem instances

In previous subsections, it was demonstrated that the benchmark problem set for the satellite domain achieved 100% coverage in every configuration. However, to delve deeper into the performance analysis, an additional problem set was chosen. These instances pose greater challenges for the planner and offer valuable insights for comparison, highlighting the advantages of the proposed solution. The selected difficult problems comprise between 1 million and 4 million operators, accompanied by a substantial increase in the number of goal conditions that need to be fulfilled. This configuration presents a significant challenge for the planner, as exploring such a vast number of possible states and discovering a route that satisfies all goal conditions is nearly impossible without preprocessing of the problem instance.

The table below shows the coverage of the base and relaxed landmarks GNN configurations.

Domain	Base	Relaxed Landmarks GNN
Satellite - big	4/30	11/30

Table 6.2: Coverage Satellite big

The results clearly demonstrate the substantial impact of the preprocessor in enhancing problem instance resolution. In comparison to the base solution, which only managed to solve 4 problems, the preprocessor generated more intelligible inputs for the planner, enabling it to solve 11 of the problems. Moreover, when examining the problem instances that both configurations successfully solved, the tool with the preprocessor exhibited significantly improved overall solving times. For instance, problems 12 and 13 were initially solved by the base configuration in 1442 and 1381 seconds respectively. However, after applying the preprocessing step, these same problems were solved in only 875 and 718 seconds respectively, showcasing the substantial time savings achieved.

6.5.2 Depots

The results obtained while performing experiments on the depots' domain show a definite advantage of the preprocessing tool in all areas. The problem instances from the domain are much harder to solve by the planners as compared to the satellites. Therefore, it is a perfect scenario for the experiments as the coverage is the metric that the solution aims to improve. However, this time the model with $h^2 - preprocessor$ was needed in order to tackle the balance between solving small problems efficiently. The time limit of the $h^2 - preprocessor$ allowed it to help the GNN model to solve small problems that turned out to be problematic. For the larger problems, the time limit was always exceeded, which was beneficial since GNN can tackle them without any help.

Domain	Base	Relaxed Landmarks GNN
Depots	22/30	30/30

Table 6.3: Coverage Depots

The results reveal that the base configuration struggled to solve only 22 out of 30 problems, indicating that the unsolved problems were the most challenging ones in the problem set. In contrast, the developed solution successfully solved all of the problems. The optimization process significantly eased the burden on the planner, enabling it to handle the problems more effectively. This improvement is evident not only in the successful solving of all problems but also in the reduced planner times and search times, highlighting the enhanced efficiency of the solution.

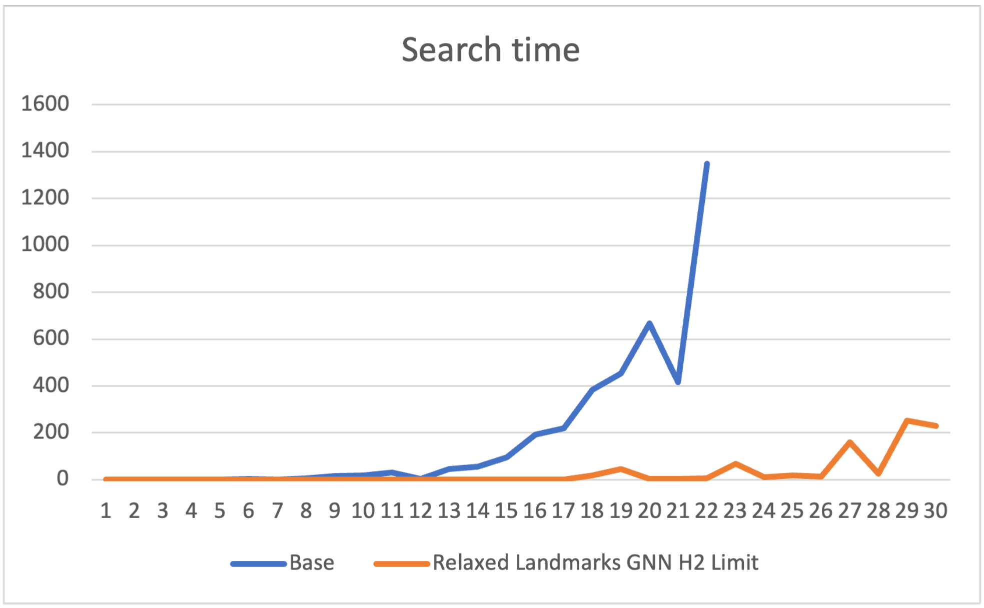


Figure 6.9: Search time base and Relaxed Landmarks GNN

The figure clearly illustrates the exponential growth of the base configuration's search time, which rapidly escalates to unreasonable values. However, with the application of preprocessing, the planner exhibits significantly faster problem instance resolution. The complexity of the problem instances included in the benchmarks does not reach a threshold where the solution would start to underperform. From the obtained results, it can be inferred that while the search time eventually experiences a noticeable increase, it does so much later than in the base configuration. This trend becomes evident when considering the planner times, emphasizing the improved efficiency and scalability achieved through preprocessing.

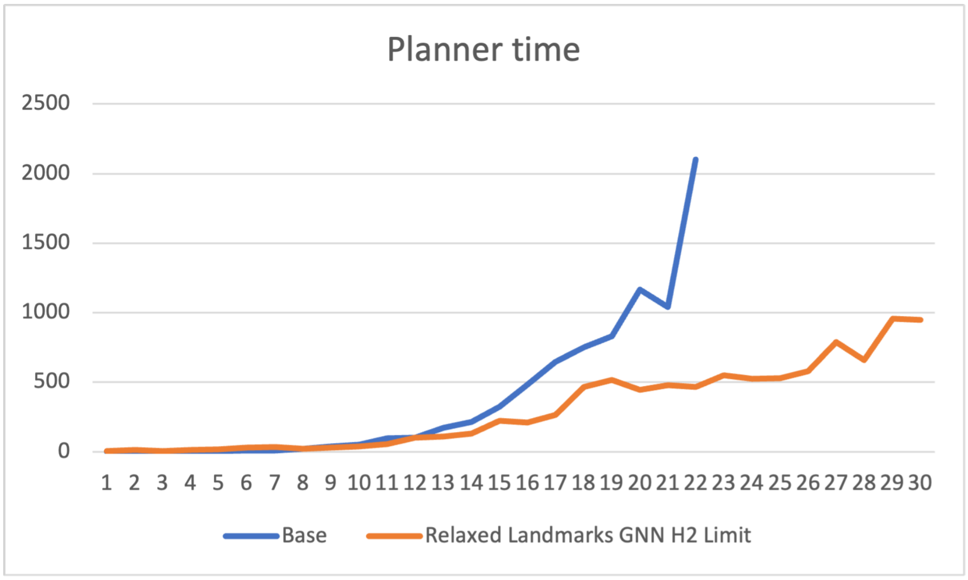


Figure 6.10: Planner time base and Relaxed Landmarks GNN

The figure provides clear evidence of the significant advantage offered by the developed tool compared to the base configuration. The total time required for plan computation exponentially increases in the base solution, while it follows a more linear growth pattern in the case of the preprocessing tool. This observation highlights the superior efficiency and scalability of the developed tool, which effectively mitigates the exponential growth of computational time experienced by the base configuration.

6.5.3 Barman

In the Barman domain, the baseline failed to solve problems 23, 25, 26, and 29, highlighting the domain's complexity. However, the similarly to the depots domain, Relaxed Landmarks GNN H2 successfully solved all of the problems, underscoring the significant improvement in coverage achieved by using the GNN model. In this domain analysis, particular attention will be given to the intriguing disparities between the performance of the GNN model and the $h^2 - preprocessor$.

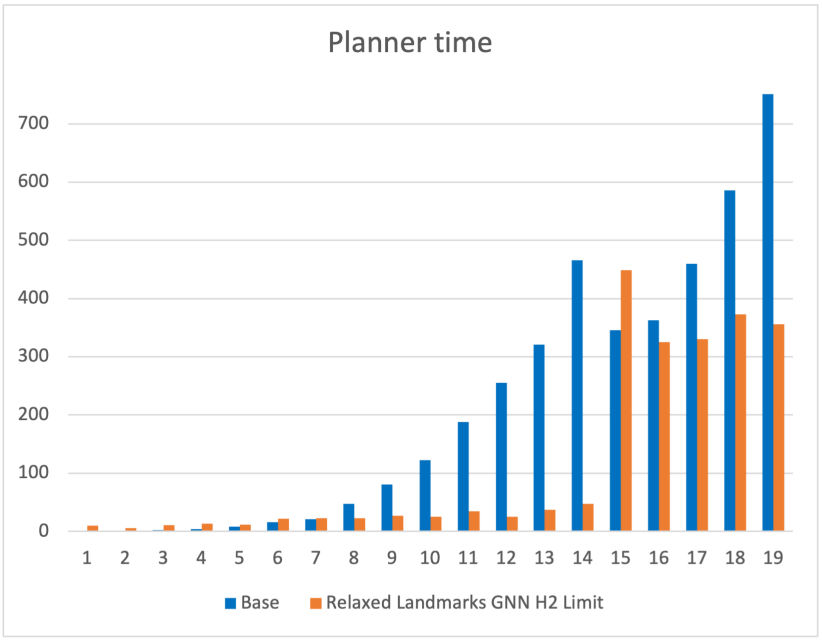


Figure 6.11: Planner time base and Relaxed Landmarks GNN problems 1-19

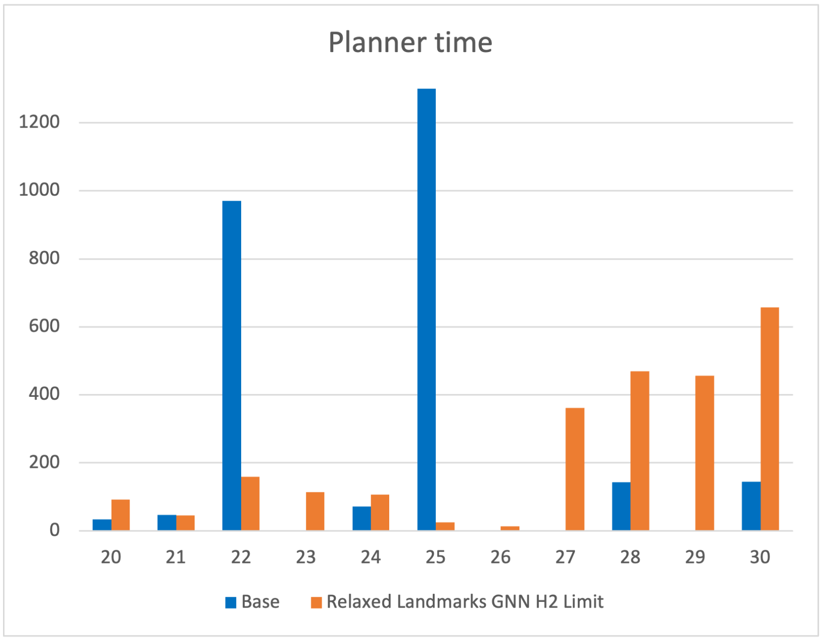


Figure 6.12: Planner time base and Relaxed Landmarks GNN problems 20-30

The analysis of planner time is segmented into two sections: problems 1 to 19 and problems 20 to 30. As depicted in Figure 6.12, problems 20 to 30 demonstrate that the GNN model outperforms the base configuration in terms of coverage, with each model facing challenges in different areas. Notably, the base configuration fails to solve certain problems, as indicated by the absence of a computation time bar (blue bar). Moreover, significant disparities in computation times can be observed. For instance, while the base configuration fails to solve problem 26, the GNN model swiftly resolves it.

Problems 1-19 represented by the first figure 6.11 can be divided into 3 batches: 1-7, 8-14 and 15-19. For the first batch, the base configuration works better despite the fact that GNN correctly reduces the number of operators 6.15. This is an expected behaviour since the GNN preprocessor adds computational overhead (graph data generation). However, the base model jumps into the $h^2 - preprocessor$ immediately. Nonetheless, with the increasing difficulty of the problems, GNN starts becoming superior which batch 15-19 visualizes. Although for batch 8-14, the GNN significantly outperforms the base configuration, its classification was not successful. Consequently, the advantage in planner time is caused by the ability to fail much faster than the $h^2 - preprocessor$ finishes processing.

The GNN performance gain is caused by a quick burnout of retries and falling back to the default actions. This happens before the $h^2 - preprocessor$ of the Base Model finishes prepossessing. The benefit of "GNN failing quickly" can be further evidenced by similar search time of the Base Model that successfully preprocessed data using h^2 and the GNN model

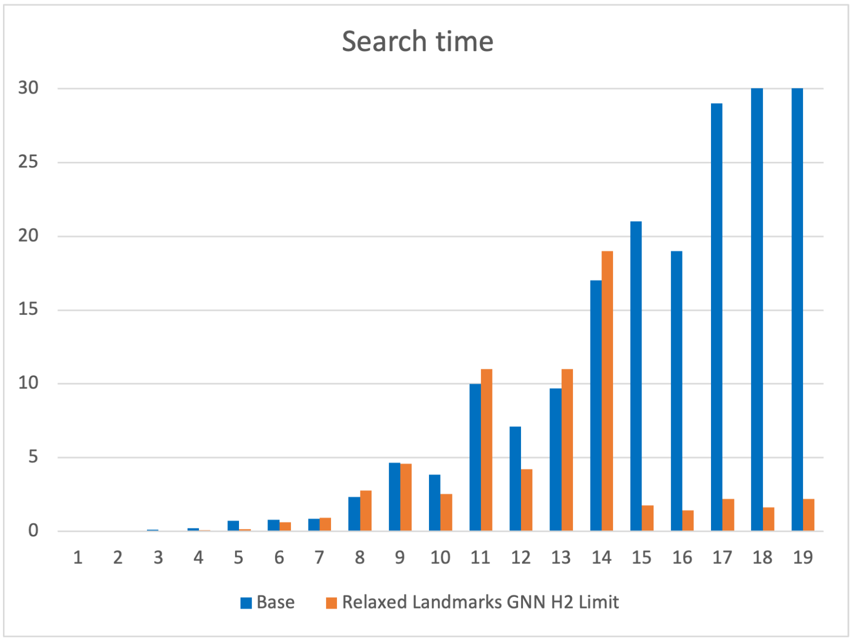


Figure 6.13: Search time base and Relaxed Landmarks GNN problems 1-19

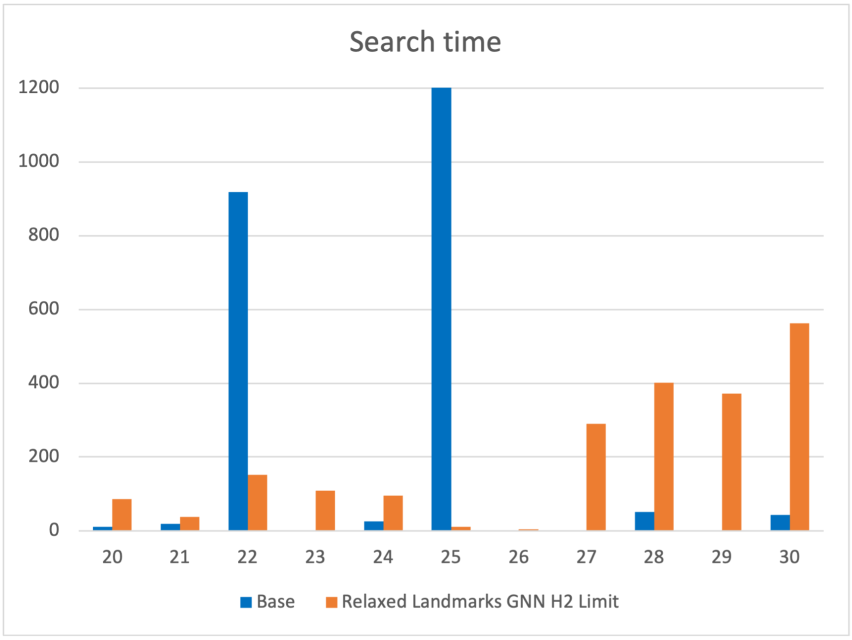


Figure 6.14: Search time base and Relaxed Landmarks GNN problems 20-30

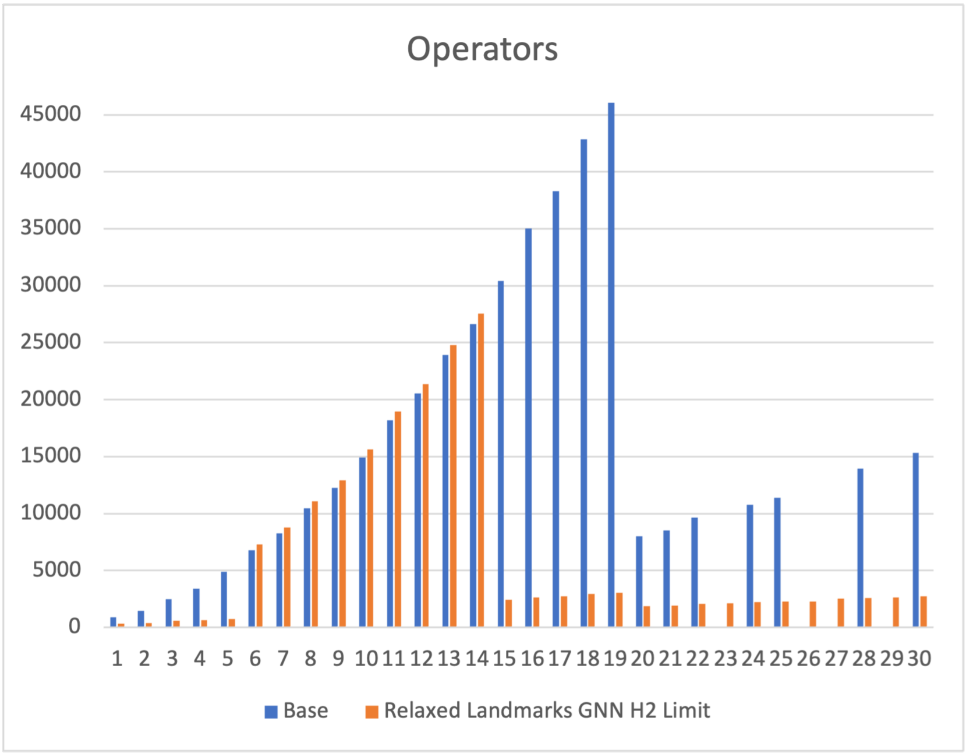


Figure 6.15: Operators number base and Relaxed Landmarks GNN

6.5.4 Rovers

The rover domain served as the simplest benchmarking domain during the experimentation phase. All of the planner configurations successfully achieved 100% coverage when tested on this domain. Moreover, the computation time for solving the base configuration did not exceed 5 seconds for any of the problem instances. These results indicate that the rover domain does not present a challenging environment.

This lack of complexity is also reflected in the classification performance. SMAC consistently selected models that predicted all operators as the positive class, achieving 100% for the Raw configuration model. Although the Relaxed Landmarks model seemed to improve the classification performance, both of the models were worse than the Base model in terms of the total planners time. 6.17

Given the specifications of the domain, the training process was considered valid since it consistently achieved 100% coverage in problem-solving. However, it is clear that the developed solution is not worth running in such simple scenarios.

The behaviour observed suggests that the solution's capabilities are suitable for handling more complex situations.

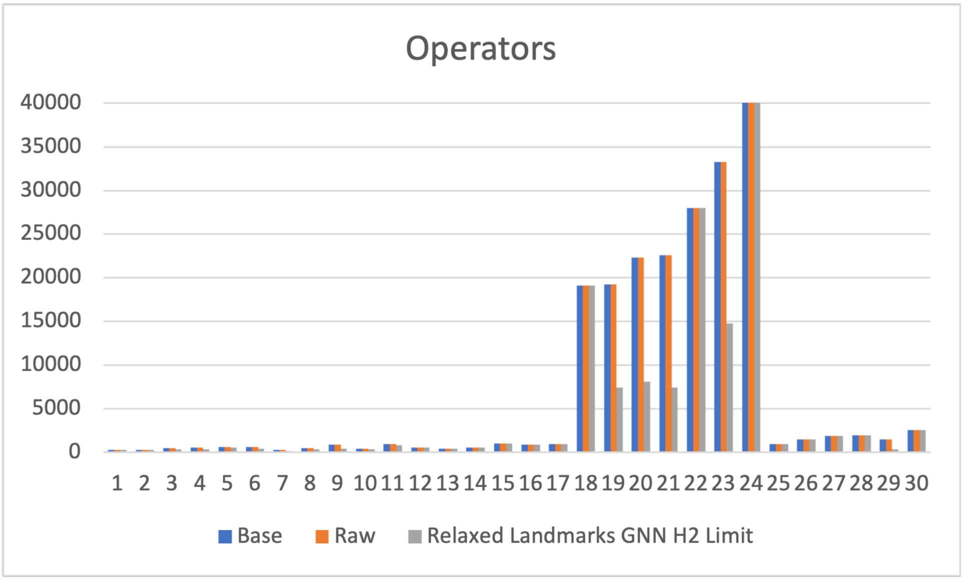


Figure 6.16: Operators number Base vs Raw vs Relaxed Landmarks

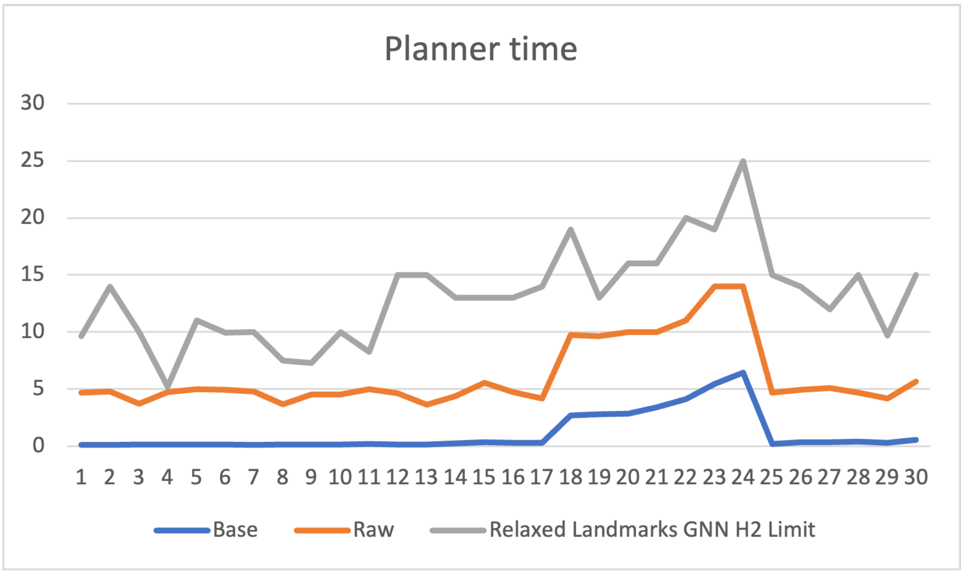


Figure 6.17: Planner time Base vs Raw vs Relaxed Landmarks

6.6 Experiments outcomes

After the experimentation phase, several conclusions can be made. The results differ depending on the complexity of the domain and its nature. The simple domains such as the described above rover domain do not benefit from additional preprocessing as the problems are easily tackled by the planner. On the other hand, solving the more complex problems belonging to more complex domains such as depots, quickly become expensive to solve or even unsolvable at all. Thanks to the diversity of the domains it is possible to asses the usability of the developed tool depending on the domain nature.

The primary purpose of the developed solution is to optimize complex problems and enable the planner to solve instances that would otherwise require preprocessing. The results indicate that this objective has been successfully accomplished. In the satellite domain, although the performance improvement was not observed in the easier problem instances, the preprocessing step significantly reduced the search time in more complex cases. Furthermore, when the tool was provided with highly complex problems, the preprocessor greatly enhanced the coverage score of the base configuration.

In the depot domain, a similar trend was observed. Due to its higher complexity, the initial coverage score achieved by the base configuration was not perfect. However, substantial improvement was achieved after applying the preprocessing step. Additionally, as the problem complexity increased, the search time and planner time grew exponentially, sometimes resulting in unmanageable processing times. The preprocessor not only enhanced the coverage score but also significantly improved both the search time and planner time.

The GNN model demonstrated an enhanced coverage score in the barman domain and improved planner times. Additionally, intriguing discoveries emerged during the examination of figures and the exploration of planning problem run logs. In some of the problems where the preprocessing failed the GNN was much faster in failing than the $h^2 - preprocessor$. This resulted in using the original set of operators, however, the planner time was improved.

This developed tool enables the planner to scale better, and efficiently solve challenging problems within shorter time frames and even tackle previously unsolved instances.

Chapter 7

Conclusion

7.1 Conclusion

The goal of this research was to develop a solution which will optimise the automated planning problems across many domains. The problem that was tackled included poor scaling of the planner which leads to the inability of solving computationally challenging problem instances. The cause of the struggles was the growing number of grounded operators which results in growing search state space and furthermore to plan computations exceeding reasonable time limits.

The approach used in the research is expanded upon the previously developed prototype whose potential was proved by the experiments in the previous work of the group [13]. In this report, the solution was extended and as a result, a complete optimization tool was created.

The enhancements include hyperparameters optimization which significantly influenced the training process and overall classification performance. The technique used was using the SMAC tool which was supplied with possible configurations and it automatically selected the best-performing options.

The prototype developed before also lacked features. Consequently, the next step for enhancing the solution was feature engineering. The proposed candidates were relaxed plan and simple landmarks which can be easily computed using a Fast-Downward planner by passing options and altering the code slightly. The features were carefully analyzed in the experimentation phase by adding them to the prototype solution incrementally.

The results of the proposed solutions were satisfactory. The experimental phase demonstrated clear advantages in solving more complex problem instances. While the solution did not significantly improve overall performance in computing plans for simple problems (which was not the primary goal), it quickly outperformed the basic approach when complex domains were employed for comparison. Noticeable improvements were observed in both search time and total planner time. Additionally, the main research goal was accomplished, as the coverage score exhibited significant improvement.

Based on the results, it can be concluded that the research was successful. The developed tool represents a comprehensive solution capable of enhancing planner performance. Moreover, it has demonstrated the viability of applying Graph Neural Networks in the field of automated planning research. Furthermore, the developed solution serves as a solid foundation for future improvements, with opportunities to consider additional features and different configurations.

7.2 Future Work

The research provided valuable insights into the strong and weak sides of the solution. Based on the experimental data the flaws of the tool can be identified and consequently, it is possible to set goals for future improvements.

One of the most significant downsides of the solution is the chosen programming language. Python is significantly slower than C++. With bigger problems where the number of operators reaches millions, this plays an important role to improve the planner's total time Consequently, one of the tasks planned for future work is outsourcing file processing to a more performant programming language such as C++.

Another improvement which could improve the performance of the Graph Neural Network classification is additional features. In the current solution, the obtained features are easy to compute and do not provide much additional knowledge. However, it could be noticed that the model which used both the relaxed plan and landmarks performed better in terms of training performance and classification. Consequently, by incorporating more features the overall performance could improve. Besides, implementing new features into the solution, the future task is to compute more meaningful landmarks features. As described in the section 3.4 there are diverse ways of obtaining landmarks which differ in computational complexity. In this solution, only simple landmarks were used, however, by incorporating more advanced information such as disjunctive or conjunctive landmarks, the tool could increase the classification accuracy.

The issues concerning the model selection process of SMAC, as pointed out in 6.4.1, deserve further attention. A more thorough investigation into improving SMAC's model selection mechanism would be beneficial. One potential approach is to intensify the complexity of the training instances—typically considered easy—by imposing stricter time constraints, implementing robust retries policies, or introducing other similar restrictions. This could potentially enable SMAC to explore superior models that have been manually identified. An unexplored strategy to alleviate this issue could be employing SMAC in two successive rounds, with the first optimizing the classification loss of the test set, followed by another aimed at the number of operators in models that resolve the plans.

Bibliography

[1] Vidal Alcázar and Álvaro Torralba. "A Reminder about the Importance of Computing and Exploiting Invariants in Planning". In: *Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling, ICAPS 2015, Jerusalem, Israel, June 7-11, 2015*. Ed. by Ronen I. Brafman et al. AAAI Press, 2015, pp. 2–6.

[2] Thomas Bartz-Beielstein, Christian Lasarczyk, and Mike Preuss. "Sequential parameter optimization". In: *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2005, 2-4 September 2005, Edinburgh, UK*. IEEE, 2005, pp. 773–780.

[3] Michael M. Bronstein et al. "Geometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges". In: *CoRR* abs/2104.13478 (2021).

[4] Sergio Jiménez Celorrio. *Barman domain description*. 2011. URL: `http://www.plg.inf.uc3m.es/ipc2011-deterministic/DomainsSequential.html?fbclid=IwAR0FHgYYHd5is-J4p72TR9jF3U2h65qIHbr-t-DjMB8e3iHuqsylvH7m14s#Barman`.

[5] Gabriele Corso et al. "Principal Neighbourhood Aggregation for Graph Nets". In: *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*. Ed. by Hugo Larochelle et al. 2020.

[6] John C. Duchi, Elad Hazan, and Yoram Singer. "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization". In: *COLT 2010 - The 23rd Conference on Learning Theory, Haifa, Israel, June 27-29, 2010*. Ed. by Adam Tauman Kalai and Mehryar Mohri. Omnipress, 2010, pp. 257–269.

[7] Richard Fikes and Nils J. Nilsson. "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving". In: *Artif. Intell.* 2.3/4 (1971), pp. 189–208.

[8] Alex Fout et al. "Protein Interface Prediction using Graph Convolutional Networks". In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc., 2017. URL: `https://proceedings.`

neurips.cc/paper/2017/file/f507783927f2ec2737ba40afbd17efb5-Paper.pdf.

[9] Alfonso Emilio Gerevini. "An Introduction to the Planning Domain Definition Language (PDDL): Book review". In: *Artif. Intell.* 280 (2020), p. 103221.

[10] Malik Ghallab, Dana S. Nau, and Paolo Traverso. *Automated planning - theory and practice*. Elsevier, 2004.

[11] Daniel Gnad et al. "Learning How to Ground a Plan - Partial Grounding in Classical Planning". In: *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*. AAAI Press, 2019, pp. 7602–7609.

[12] Daniel Godoy. *Understanding binary cross-entropy / log loss: a visual explanation*. URL: https://towardsdatascience.com/understanding-binary-cross-entropy-log-loss-a-visual-explanation-a3ac6025181a.

[13] Piotr Rafal Gzubicki and Bartosz Piotr Lachowwicz. "Exploiting graph structures of planning problems with the use of Neural Networks". In: 2022.

[14] William L. Hamilton, Rex Ying, and Jure Leskovec. "Inductive Representation Learning on Large Graphs". In: *CoRR* abs/1706.02216 (2017). URL: http://arxiv.org/abs/1706.02216.

[15] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. "A Formal Basis for the Heuristic Determination of Minimum Cost Paths". In: *IEEE Trans. Syst. Sci. Cybern.* 4.2 (1968), pp. 100–107. URL: https://doi.org/10.1109/TSSC.1968.300136.

[16] Malte Helmert. "The Fast Downward Planning System". In: *J. Artif. Intell. Res.* 26 (2006), pp. 191–246. URL: https://doi.org/10.1613/jair.1705.

[17] Malte Helmert and Carmel Domshlak. "Landmarks, Critical Paths and Abstractions: What's the Difference Anyway?" In: *Graph Search Engineering, 29.11. - 04.12.2009*. Ed. by Lubos Brim et al. Vol. 09491. Dagstuhl Seminar Proceedings. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany, 2009. URL: http://drops.dagstuhl.de/opus/volltexte/2010/2432/.

[18] Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. *Neural Networks for Machine Learning*. 2012. URL: https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf.

[19] Jörg Hoffmann and Bernhard Nebel. "The FF Planning System: Fast Plan Generation Through Heuristic Search". In: *J. Artif. Intell. Res.* 14 (2001), pp. 253–302.

[20] Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. "Sequential Model-Based Optimization for General Algorithm Configuration". In: *Learning and Intelligent Optimization - 5th International Conference, LION 5, Rome, Italy, January 17-21, 2011. Selected Papers*. Ed. by Carlos A. Coello Coello. Vol. 6683. Lecture Notes in Computer Science. Springer, 2011, pp. 507–523.

[21] Frank Hutter et al. "An experimental investigation of model-based parameter optimisation: SPO and beyond". In: *Genetic and Evolutionary Computation Conference, GECCO 2009, Proceedings, Montreal, Québec, Canada, July 8-12, 2009*. Ed. by Franz Rothlauf. ACM, 2009, pp. 271–278.

[22] Frank Hutter et al. "Time-Bounded Sequential Parameter Optimization". In: *Learning and Intelligent Optimization, 4th International Conference, LION 4, Venice, Italy, January 18-22, 2010. Selected Papers*. Ed. by Christian Blum and Roberto Battiti. Vol. 6073. Lecture Notes in Computer Science. Springer, 2010, pp. 281–298.

[23] Donald R. Jones, Matthias Schonlau, and William J. Welch. "Efficient Global Optimization of Expensive Black-Box Functions". In: *J. Glob. Optim.* 13.4 (1998), pp. 455–492.

[24] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. 2015.

[25] Marius Lindauer et al. "SMAC3: A Versatile Bayesian Optimization Package for Hyperparameter Optimization". In: *Journal of Machine Learning Research* 23.54 (2022), pp. 1–9. URL: `http://jmlr.org/papers/v23/21-0888.html`.

[26] Derek Long. *The Depots Domain*. 2003. URL: `https://www.cs.cmu.edu/afs/cs/project/jair/pub/volume20/long03a-html/node38.html`.

[27] Derek Long. *The Rovers Domain*. 2003. URL: `https://www.cs.cmu.edu/afs/cs/project/jair/pub/volume20/long03a-html/node42.html`.

[28] Derek Long. *The Satellite Domain*. 2003. URL: `https://www.cs.cmu.edu/afs/cs/project/jair/pub/volume20/long03a-html/node41.html`.

[29] Andrew Ng and DeepLearningAI. *Adam Optimization Algorithm*. 2018. URL: `https://www.youtube.com/watch?v=JXQT_vxqwIs&ab_channel=DeepLearningAI`.

[30] Nir Pochter, Aviv Zohar, and Jeffrey S. Rosenschein. "Exploiting Problem Symmetries in State-Based Planners". In: *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2011, San Francisco, California, USA, August 7-11, 2011*. Ed. by Wolfram Burgard and Dan Roth. AAAI Press, 2011. URL: `https://ojs.aaai.org/index.php/AAAI/article/view/8014`.

[31] PyTorch. *Pytorch*. URL: `https://pytorch.org/`.

[32] Silvia Richter, Malte Helmert, and Matthias Westphal. "Landmarks Revisited". In: *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008, Chicago, Illinois, USA, July 13-17, 2008*. Ed. by Dieter Fox and Carla P. Gomes. AAAI Press, 2008, pp. 975–982. URL: `https://ai.dmi.unibas.ch/_files/teaching/hs20/po/misc/richter-et-al-aaai2008.pdf`.

[33] Silvia Richter and Matthias Westphal. "The LAMA Planner: Guiding Cost-Based Anytime Planning with Landmarks". In: *J. Artif. Intell. Res.* 39 (2010), pp. 127–177. URL: `https://doi.org/10.1613/jair.2972`.

[34] Jussi Rintanen. "Planning as satisfiability: Heuristics". In: *Artif. Intell.* 193 (2012), pp. 45–86. URL: `https://doi.org/10.1016/j.artint.2012.08.001`.

[35] Alvaro Sanchez-Gonzalez et al. "Graph Networks as Learnable Physics Engines for Inference and Control". In: *Proceedings of the 35th International Conference on Machine Learning*. Vol. 80. Proceedings of Machine Learning Research. 2018, pp. 4470–4479.

[36] Sanghvirajit. *A Complete Guide to Adam and RMSprop Optimizer*. 2021. URL: `https://medium.com/analytics-vidhya/a-complete-guide-to-adam-and-rmsprop-optimizer-75f4502d83be`.

[37] Jendrik Seipp, Thomas Keller, and Malte Helmert. "Saturated Cost Partitioning for Optimal Classical Planning". In: *J. Artif. Intell. Res.* 67 (2020), pp. 129–167.

[38] Jendrik Seipp and Javier Segovia-Aguas. *International Planning Competition 2023 - Learning track*. 2023. URL: `https://ipc2023-learning.github.io/`.

[39] Jendrik Seipp et al. *Downward Lab*. `https://doi.org/10.5281/zenodo.790461`. 2017.

[40] Snap-Stanford. *Graph Neural Networks*. URL: `https://snap-stanford.github.io/cs224w-notes/machine-learning-with-networks/graph-neural-networks`.

[41] Shyam A. Tailor et al. "Do We Need Anisotropic Graph Neural Networks?" In: *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022.

[42] PyG Team. *PyG Docummentation*. URL: `https://pytorch-geometric.readthedocs.io/en/latest/`.

[43] PyG Team. *torch geometric nn*. URL: `https://pytorch-geometric.readthedocs.io/en/latest/modules/nn.html`.

[44] Álvaro Torralba, Jendrik Seipp, and Silvan Sievers. "Automatic Instance Generation for Classical Planning". In: *Proceedings of the Thirty-First International Conference on Automated Planning and Scheduling (ICAPS 2021)*. Ed. by Robert P. Goldman, Susanne Biundo, and Michael Katz. AAAI Press, 2021, pp. 376–384.

[45] Sam Toyer et al. "Action Schema Networks: Generalised Policies With Deep Learning". In: *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*. Ed. by Sheila A. McIlraith and Kilian Q. Weinberger. AAAI Press, 2018, pp. 6294–6301.

[46] Petar Velickovic et al. "Graph Attention Networks". In: *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018.

[47] Chi-Feng Wang. *The Vanishing Gradient Problem - The Problem, Its Causes, Its Significance, and Its Solutions*. 2019. URL: `https://towardsdatascience.com/the-vanishing-gradient-problem-69bf08b15484`.

[48] Yongji Wu et al. "Graph Convolutional Networks with Markov Random Field Reasoning for Social Spammer Detection". In: *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*. AAAI Press, 2020, pp. 1054–1061.

[49] Keyulu Xu et al. "How Powerful are Graph Neural Networks?" In: *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

[50] Zhitao Ying et al. "Hierarchical Graph Representation Learning with Differentiable Pooling". In: *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*. Ed. by Samy Bengio et al. 2018, pp. 4805–4815.

[51] Sung Wook Yoon, Alan Fern, and Robert Givan. "Learning Heuristic Functions from Relaxed Plans". In: *Proceedings of the Sixteenth International Conference on Automated Planning and Scheduling, ICAPS 2006, Cumbria, UK, June 6-10, 2006*. Ed. by Derek Long et al. AAAI, 2006, pp. 162–171.

[52] Mingxia Zhao and Adele Lu Jia. "A Dual-Attention Heterogeneous Graph Neural Network for Expert Recommendation in Online Agricultural Question and Answering Communities". In: *25th IEEE International Conference on Computer Supported Cooperative Work in Design, CSCWD 2022, Hangzhou, China, May 4-6, 2022*. IEEE, 2022, pp. 926–931.

[53] Lin Zhu and Robert Givan. *Landmark Extraction via Planning Graph Propagation*. In *Printed Notes of ICAPS'03 Doctoral Consortium*. Trento, Italy. 2003.