Domain-Dependent Policy Learning using Neural Networks in Classical Planning

Bachelor Thesis

Lukas Schäfer

Reviewers:
Prof. Dr. Jörg Hoffmann,
Dr. Mario Fritz

Supervisors:
Prof. Dr. Jörg Hoffmann,
Patrick Ferber,
Maximilian Fickert

Foundations of Artificial Intelligence (FAI) Group
Faculty of Mathematics and Computer Science
Department of Computer Science
Saarland University

Submitted: July 30, 2018

Acknowledgements

First, I would like to express my gratitude to Prof. Dr. Jörg Hoffmann for offering me the opportunity to write this thesis and giving useful feedback. Thanks also to the entire staff of the FAI Group for the support as well as the software and hardware provided for the evaluation.
Special thanks goes to Christian Bohnenberger for his counsel, valuable feedback for these past months and proofreading.
Furthermore a big thank you to my advisors Patrick Ferber and Maximilian Fickert for their invaluable guidance throughout the whole work. They were always keen to assist and their help was essential for the success of this project.
Finally, I want to thank my family and friends for the motivation and constant support during the entire development process of this thesis.

Erklärung

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

Statement in Lieu of an Oath

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

Einverständniserklärung

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

Declaration of Consent

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

Saarbrücken, ______________________ ______________________________________

(Datum/Date) (Unterschrift/Signature)

Erklärung

Ich erkläre hiermit, dass die vorliegende Arbeit mit der elektronischen Version übereinstimmt.

Statement

I hereby confirm the congruence of the contents of the printed data and the electronic version of the thesis.

Saarbrücken, ______________________ ____________________________________

(Datum/Date) (Unterschrift/Signature)

Abstract

This thesis transfers and evaluates the work of Action Schema Networks (ASNets) for domain-dependent policy learning for classical automated planning. First, we will introduce the foundational background of automated planning and deep learning in the form of neural networks. Subsequently, the structure and learning of ASNets will be explained as well as their partially already evaluated performance. Afterwards, the definition of the network for application in the Fast-Downward planning system and necessary extensions to this framework will be explained. This also includes an adapted training and sampling strategy for efficient learning of ASNets. Lastly, an extensive empirical evaluation is conducted to compare the network performance in classical planning to state-of-the-art planners and assess whether these neural networks are suited for this planning field. While it could be seen that ASNets are capable of learning effective policies for application in search, they still have major limitations regarding their ability to generalise and scale. In the end, we propose extensions and modifications based on our evaluation results to improve their performance for further research.

Contents

Chapter 1

Introduction

The wide field of artificial intelligence (AI) has many different branches. One of which is automated planning with the big goal of creating an intelligent agent which is able to efficiently solve (almost) arbitrary problems. While this sounds like a vision far in the future, modern planning systems are already capable of solving a wide variety of tasks, e.g. complex scheduling tasks. The field has seen great progress with the upcoming of heuristic search in 2000 and upon this concept many sophisticated heuristics were developed and thoroughly researched. However, it might be surprising that planning has seen little interaction with the field of machine learning despite its rise in popularity. Only in recent past, these two fields were combined with mixed success.

Machine learning itself is a branch of AI covering algorithms which allow a system to learn from data. This concept has received a tremendous increase in attention from public and research over the last decade. The ability to learn and rationally apply gained knowledge was and still is the main reason why humans are superior to computers in solving many problems despite their gradually increasing computational power. This makes machine learning so exciting, because its approach aims to change the major remaining limitation of computers. While technology is still far from rational thinking robots, as presented in science fiction, the field has seen astonishing progress in the last years. One popular success story of machine learning would be Alpha Go [49] and Alpha Go Zero [50]. These algorithms attracted a wide media attention in 2016, when they were able to beat a human professional player in the Chinese board game Go. A success story of AI given that this achievement was predicted to still be a decade from reality, due to Go's computational complexity. In the core of these programs were neural networks, which are often titled with the topic of deep learning, combined with algorithms from or at least related to automated planning.

The success of neural networks, even besides Alpha Go in domains like image recognition [33] and machine translation [4] motivated research from all branches of AI to look for further applications of these networks and fitting architectures for the upcoming challenges. Toyer et al. from the Australian National University recently proposed a new neural network structure designed for application in probabilistic and classical automated planning, called Action Schema Networks [53, 54]. These are able to learn domain-specific knowledge in planning and apply it to unseen problems of the same domain. The promising structure was primarily introduced and evaluated considering probabilistic planning.

In this thesis, we aim to evaluate the possible performance of neural networks in form of Action Schema Networks in classical planning. Therefore we begin by explaining the relevant background of automated planning as well as deep learning and outline already existing approaches for learning in planning. Secondly, a detailed description of Action Schema Networks will be provided explaining their architecture and general capabilities.

The main contribution of this thesis will be the implementation of this novel neural network structure in the Fast-Downward planning system [25] for application in deterministic, classical planning. We will explain extensions of the Fast-Downward system necessary for Action Schema Networks and lastly conduct an empirical evaluation comparing the neural network to prominent, modern planning systems on multiple tasks of varying complexity. This evaluation will consider different configurations to assess whether Action Schema Networks are a suitable method for classical planning and if so under which conditions. In the end, possible further modifications and extensions for neural networks and Action Schema Networks in particular for the classical planning field will be proposed, based on our analyses of the evaluation, aiming towards the goal of learning complex relations occurring in planning tasks.

Chapter 2

Background

In this chapter, we aim to explain the core concepts of automated planning as well as neural networks. Afterwards we will briefly look into the already existing research done about learning in automated planning. This serves as a foundation to explain Action Schema Networks.

2.1 Automated Planning

For this thesis, we will focus on classical planning, which is the most basic form of *automated planning* (*AP*). Hence, the work focuses on finite, deterministic, fully-observable problems solved by a single agent. The predominant formalisation for planning tasks is *STRIPS* [19] representing such a task as $\Pi = (\mathcal{P}, \mathcal{A}, c, I, G)$:

- $\mathcal{P}$ is a set of *propositions* (or *facts*)
- $\mathcal{A}$ is a set of *actions* where each action $a \in \mathcal{A}$ is a triple (pre_a, add_a, del_a) with $pre_a, add_a, del_a \subseteq \mathcal{P}$ including a's *preconditions*, *add list* and *delete list* with $add_a \cap del_a = \emptyset$
  - preconditions are facts, which have to be true for a to be applicable
  - add list contains all propositions becoming true after applying a
  - delete list contains all propositions becoming false after applying a
- $c : \mathcal{A} \rightarrow \mathbb{R}_0^+$ is the *cost function* assigning all actions to their cost
- $I \subseteq \mathcal{P}$ is the *initial state* containing all propositions, which are true at the start of the task
- $G \subseteq \mathcal{P}$ is the *goal* with all facts which have to become true to solve the task

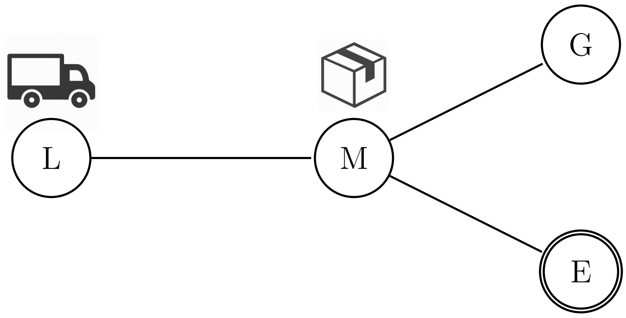


Figure 2.1: Transport planning-task. The truck has to drive from London (L) to Manchester (M), pick up the package, drive to Edinburgh (E) and unload the package there

For example, one could describe a transportation task in which a truck has to deliver a package from some location to its destination by driving along streets and load or unload packages, illustrated in Figure 2.1. There would be propositions $\mathcal{P} = \{at(o, x) \mid o \in \{t, p\}, x \in \{L, M, G, E\}\}$ and actions $\mathcal{A} = \{drive(x, y, z) \mid x \in \{truck\},\ y,\ z \in \{L, M, G, E\},\ y$ and z are connected$\} \cup \{load(x, y, z), unload(x, y, z) \mid x \in \{truck\},\ y \in \{package\},\ z \in \{L, M, G, E\}\}$. The goal could be formalised as $\{at(package, E)\}$ and the initial state describes the starting position of the truck and package as $\{at(truck, L), at(package, M)\}$.

To solve any task Π, the agent has to observe the current state and choose actions, one at a time, in order to reach a goal state s^* with $G \subseteq s^*$. The sequence of actions, applied to get to such a state, is called *plan* for Π. A plan is considered *optimal* if it has the least cost out of all plans reaching a goal. E.g., the optimal plan for our transport task would be $\langle drive(truck, L, M),\ load(truck, package, M),\ drive(truck, M, E), unload(truck, package, E)\rangle$.

Modelling planning-tasks consist of two components: the *domain* and the *problem*. This separation has its origin in the main modelling language for planning *PDDL* (Planning Domain Definition Language) introduced by McDermott et al. [39]. A domain describes a family of various problems sharing the core idea. It contains predicates defined on abstract objects, which are organised in a type hierarchy, as well as action schemas. Problem instances are always assigned a domain which predefines mentioned elements. In the problem file concrete objects are defined, which instantiate the predicates and action schemas of the domain to propositions and actions respectively. Furthermore the initial and goal states are specified.

Looking at the transport task, the domain would define general concepts like locations, trucks and packages with the predicates $at(?o - locatable, ?l - location)$ and $connected(?l1 - location, ?l2 - location)$ annotating the position of locatable objects like trucks and packages as well as connections between locations. Additionally, it would include the abstract schema for all actions. E.g. the drive schema could look like the following $drive(?v - vehicle, ?from - location, ?to - location)$ with its preconditions, add and delete lists. The problem would introduce the exact locations, here L, M, G and E, as well as the *truck* and *package* and state the initial positions of both these objects together with the connections between locations and the goal.

Most of these planning problems seem conceptually easy for rational-thinking humans, but this impression can be misleading. In fact, planning is computationally extremely difficult. Merely deciding whether a task is solvable is already PSPACE-complete [10].

In order to overcome this problem and build a planner, which is able to solve arbitrary planning-tasks, AI-research came up with many different approaches over the past 50 years. Since IPC[1] 2000, the most promising solution seems to be heuristic search. This approach traverses the problem's state-space with a search algorithm, e.g. A*, led by a heuristic function which estimates a state's distance to the closest goal.

Due to the computational complexity of planning, these heuristics must significantly relax the task to provide feasible guidance. One popular approach is the delete-relaxed plan heuristic h^{FF}, introduced by Hoffmann and Nebel in 2001 [27], which relaxes the problem by removing all deletes from actions. Therefore all propositions remain true within the computation of the heuristic value for a given state after they have been achieved once.

[1] The International Planning Competition (IPC) is the major driving force in AP research which started in 1998.

This form of abstraction seems highly confusing to humans, but can be computed in polynomial time, significantly reducing the computational effort, without loosing too much information. There are many more approaches to heuristic functions besides delete-relaxation.

2.2 Learning in Automated Planning

While heuristic functions significantly improved the performance of planning systems, most planners use domain-independent heuristics due to their flexibility. These can be applied to arbitrary domains but compared to domain-specific heuristics, which can improve scalability of planners [2, 41] in a subset of domains, limit performance. However, domain-specific knowledge is required to define these tailored heuristics. Manually defining this knowledge is difficult and needs expertise in the domain and planning. Therefore one application of learning in AP is to learn such domain-specific knowledge to create specialised heuristics.

Generally there are various different approaches on how to apply (machine) learning in AP. This topic, despite having a long history, only received more attention in recent past. Jiménez et al. provide an overview of multiple of these ideas proposed [29] and differentiate between two categories of learning in planning. The first one is learning action models, so learning the fundamental model of the task. Secondly they look at learning control knowledge which can be exploited during the search. For each approach, one has to decide on how to represent, obtain and exploit learned knowledge. One popular approach of learning control knowledge, used in search, is to learn generalised policies proposing actions depending on the context, which usually consists of the current state and the goal. E.g. de la Rosa et al. used generalised policies in their ROLLER planner [15].

Another application that received tremendous attention was DeepMind's AlphaGo [49], later AlphaGo Zero [50] which has convincingly beaten human professionals in the Chinese board game Go. They used Monte Carlo Tree Search (MCTS) [31, 14] with two policies, which were both learned by neural networks using (supervised learning and) reinforcement learning. The success of these algorithms were especially impressive given the computational complexity of the game Go and their usage of neural networks. Before, these architectures were rarely seen in the context of AP.

2.3 Deep Learning

The idea of *neural networks* (*NNs*) has a long history reaching back to the 1940s [38] inspired by the human brain whose immensely impressive capabilities are partly due to the dense connectivity of neurons. With the introduction of the perceptron, which was capable of learning, by F. Rosenblatt in 1958 [45] and backpropagation by Rumelhart et al. in 1986 [46] the foundation for modern NNs was built.

2.3.1 Multi-Layer Perceptron

The simplest, modern NN architecture is the *fully-connected feedforward network* or *multi-layer perceptron* (*MLP*) as visualized in Figure 2.2. A MLP always consists of at least two layers: one *input layer* receiving the input x, visualized in yellow colour, and the *output layer*, green in the figure, whose nodes transmit the output vector $\hat{y}$. In between there can be an arbitrary amount of *hidden layers* and every node in a layer l has a weighted forward connection to every node in successive layer $l+1$. The weights are represented in the matrices W^l and are the primary parameters which are learned by such a network.

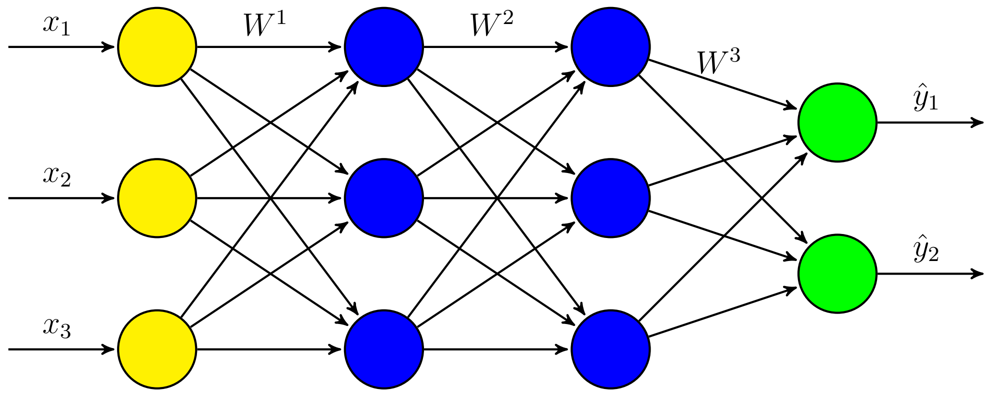


Figure 2.2: Multi-Layer Perceptron architecture with 4 layers: one input layer in yellow, two hidden layers in blue and one output layer in green

During evaluation the values are computed layer-wise leading to a representational vector h^l for layer l, which is the result of the corresponding weight matrix W^l, a bias vector b^l, the hidden representation of the previous layer h^{l-1} and an elementwise, non-linear *activation function* f:

$$h^l = f(W^l \cdot h^{l-1} + b^l) \tag{2.1}$$

The nonlinearity of f allows the network to represent more complex relations with popular activation functions being RELU $f(x) = max(0, x)$ and $f(x) = tanh(x)$.

Training of MLPs is often done by *supervised learning* where a dataset of labelled data, consisting of input vector x and corresponding outputs y, is provided. The goal is to find the parameters θ of W^1, ..., W^L and b^1, ..., b^L minimizing the *loss* $L(\hat{y}, y)$ computed by the prediction $\hat{y}$ and the correct labels y. One popular loss function is the *mean squared error* (*MSE*):

$$L(\hat{y}, y) = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2 \tag{2.2}$$

Loss functions can be seen as an estimated distance from the network's prediction to the correct output labels. This can be used as a metric for the network's quality but also to improve the network's parameters. Given the loss, θ can be updated in the direction of the steepest descent of L with respect to θ using *gradient descent*:

$$\theta' = \theta - \alpha \nabla_\theta L(\hat{y}, y) \tag{2.3}$$

α represents the *learning rate* which determines the magnitude of the descent. These updates can be seen as gradual steps downhill the loss surface towards parameters deemed to be of higher quality, i.e. leading to predictions closer to the provided training data. Therefore during training, the neural network tries to imitate the underlying function implicitly represented by the used data set. To compute $\nabla_\theta L(\hat{y}, y)$ *backpropagation*, i.e. the chain rule for derivation, is used.

Continuously using gradient descent guarantees the loss to reach a local minimum (or saddle point) where $\nabla_\theta L(\hat{y}, y) = 0$.

There are two main challenges in using NNs. The first one is introduced by hyperparameters of the network like the learning rate α, the number of layers L and units in each layer etc. They have a tremendous influence on the learning and capacity of the network. Therefore choosing appropriate parameters is one of the main problems to solve whenever using NNs.

Another issue, networks frequently suffer from, is *overfitting*. This happens whenever the generalisation error is high, so the network is able to minimize the loss on training data efficiently, but is incapable of performing well on data outside of the specific training data. One could say, the network is "too specialised" on the limited data set used for training. There are various different causes for overfitting and measures to prevent it. One of the most popular approach to avoid overfitting is *regularization* in which a term is added to the loss function punishing e.g. large weights. This is often already sufficient to prevent generalisation issues but the difficulty whenever dealing with neural networks remains. Another popular strategy to diminish the risk of overfitting is dropout [51] in which nodes in the network with their corresponding connections are deactivated with a given probability p during training, i.e. their output values and connections are dropped in the currently processed training computation.

There are multiple challenges, like overfitting and choosing hyperparameters, for which frequently used methods exist which often resolve potential issues. However, these should not be regarded as generally applicable solutions to those problems but rather as potentially useful techniques when facing these situations.

2.3.2 Convolutional Neural Networks

Convolutional neural networks (*CNNs*) [34] are a class of NNs which are specifically designed for multi-dimensional inputs. They are most prominently applied to image data, reaching state-of-the-art performance in image classification [42] among other domains of visual computing. But further application has been found. E.g., a convolutional layer has successfully been used to extract contextual features on hashed word-representations for query-processing in information retrieval [48].

The main characteristic of CNNs is the application of the mathematical *convolution operation*. This linear operation replaces the typical matrix multiplication known from MLPs where every unit in each layer has a weighted connection to every node in the successive layer. In convolution, smaller weight matrices, called *filters* or *kernels*, are applied by "sliding" the filters over the units in one layer with each applying its operation to a set of neighboured inputs. This form of processing brings multiple advantages.

Due to the usually smaller size of filters, CNNs have *sparse connectivity*, only combining neighboured units in one operation. This allows to make use of local properties to extract input features like edges in visual domains, which is especially meaningful in deep CNNs. While shallow layers could detect edges or shapes of an input image, filters in deeper layers could work upon these features and potentially detect increasingly abstract objects like cars and humans.

On the other hand, sparse connectivity also means that units in the network are not necessarily connected to every node in the consecutive layer. While in CNNs, one unit might only affect a few units in the following layer, the same holds true for these nodes.

Therefore the number of units (indirectly) affected by a node increases the deeper we go in the network. This indirect interaction proved to be highly efficient and sufficient to learn complex dependencies.

Additionally, filters are reused repeatedly during the "sliding", applying their operation to (partially) different input units. This form of *weight sharing* allows to significantly reduce the amount of parameters needed. Hence, the memory requirements of the networks are lowered, making them more efficient than fully-connected NNs because less parameters have to be learned, so needed training time and data can be reduced by the approach.

After applying the convolution operation, a nonlinear activation function is applied to the output. Similarly as for MLPs, these functions are essential to learn connections more sophisticated than linear relations. In addition, it is important for CNNs to be *invariant* to small input transformations to be able to reliably detect local properties as intended. The convolution operation in itself is already *equivariant*, i.e. applying a function to its output leads to the same data as applying the convolution to the translated input, making it invariant to some translations, like minor shifts of input images. In many cases this property is not sufficient, because the CNN would still be affected by e.g. rotations. To be able to deal with these, *pooling functions* are used after applying nonlinearity. A pooling function provides a statistical recollection of a group of input units by e.g. averaging same-sized groups of neighboured values of the previous layer or taking the maximum of each of these groups. This provides further invariance to input transformations and also reduces the input size without major loss of information, which can be essential to deal with varying input sizes.

2.4 Action Schema Networks

In this section, we aim to explain the *Action Schema Networks* (*ASNets*), a novel NN family suited for AP, proposed by Sam Toyer et al [53, 54]. The network is capable of learning domain-specific knowledge, in form of policies, to exploit on arbitrary problems of a given (P)PDDL domain. First, we will cover the architecture and design of the network. Further, the training and exploitation of learned knowledge will be explained. Lastly, we will briefly cover Sam Toyer's empirical evaluation of ASNets' performance.

2.4.1 Architecture

ASNets are composed of alternating action and proposition layers, containing *action modules* and *proposition modules* for each ground action or proposition respectively. This alternating approach of action and proposition layers was influenced by the Graphplan Planner [5]. The input features are always fed into an action layer. Overall the network computes a *policy* π^θ, outputting a probability $\pi^\theta(a \mid s)$ to choose action a in a given state s for every action $a \in \mathcal{A}$. The output is done by an action layer and is depending on the network's parameters θ. One very simple approach to exploit this policy during search on planning tasks would be to always choose the action with the highest probability according to π^θ.

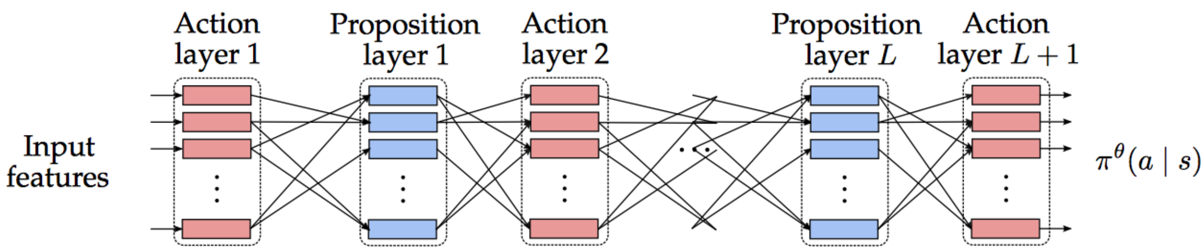


Figure 2.3: Illustration out of Toyer's thesis [53]; Depicting a L-layer ASNet with L proposition layers (blue) and $L+1$ action layers (red)

Action modules

The action modules forming each action layer represent ground actions of the planning task. An action module for $a \in \mathcal{A}$ in the l-th layer produces a hidden representation

$$\phi_a^l = f(W_a^l \cdot u_a^l + b_a^l) \tag{2.4}$$

where $u_a^l \in \mathbb{R}^{d_a^l}$ is an input feature vector, $W_a^l \in \mathbb{R}^{d_h \times d_a^l}$ is a learned weight matrix for this action module and $b_a^l \in \mathbb{R}^{d_h}$ is the corresponding bias. f is a nonlinearity, e.g. RELU or tanh as specified in Section 2.3.1. d_h is a chosen intermediate representation size. The input vector is a concatenation of hidden representations of proposition modules ψ_1^{l-1}, ..., ψ_M^{l-1}, corresponding to propositions p_1, ..., p_M that are *related* to action a, in the preceding proposition layer.

$$u_a^l = \begin{bmatrix} \psi_1^{l-1} \\ \vdots \\ \psi_M^{l-1} \end{bmatrix} \tag{2.5}$$

Proposition $p \in \mathcal{P}$ is said to be related to action $a \in \mathcal{A}$, also written $R(a,p)$, iff p appears either in pre_a, add_a or del_a. Each hidden proposition representation $\psi_i^{l-1} \in \mathbb{R}^{d_h}$ has chosen representation size d_h. Therefore the input size $d_a^l = d_h \cdot M$ is fixed. The sparse connectivity of ASNets based on the definition of relatedness is inspired by the idea of filters in CNNs, which connect data points based on spatial neighbourhood.

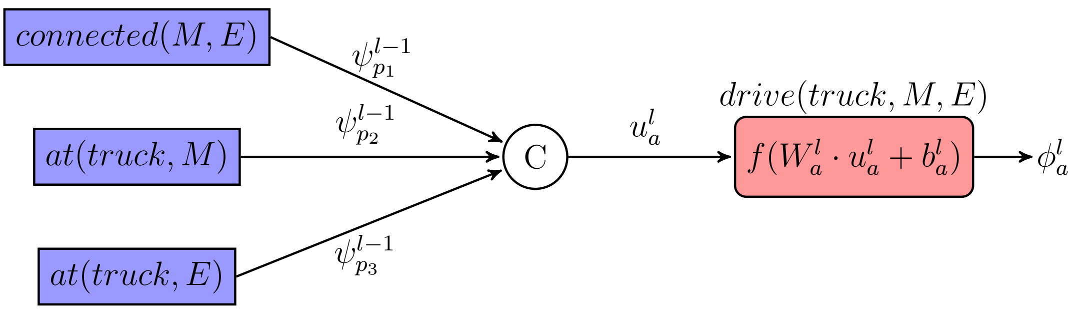


Figure 2.4: Illustration of the action module for $drive(truck, M, E)$ action, with $truck$ driving from Manchester (M) to Edinburgh (E). The hidden representations of the related proposition modules are received as an input, concatenated to u_a^l and the hidden representation of the action module ϕ_a^l is computed as described.

Due to relatedness, given two actions a_1 and a_2, that are concrete instances of the same action schema in the domain, the number of related propositions N will be the same. If p_1, ..., p_N and q_1, ..., q_N are related to a_1 and a_2 respectively, then there exists an order so that p_i and q_i are concrete propositions constructed by the same predicate.

E.g. $a_1 = drive(truck, M, E)$ (as seen in figure 2.4) and $a_2 = drive(truck, L, M)$ are both actions instantiated by the action schema $drive(?v{-}vehicle,\ ?from{-}location,\ ?to{-}location)$. This schema defines the preconditions as $\{connected(?from, ?to),\ at(?v, ?from)\}$, the add list $\{at(?v, ?to)\}$ and delete list $\{at(?v, ?from)\}$. Therefore, we can extract that all actions of this schema are related to the concrete propositions instantiated by the predicates $\{connected(?from, ?to),\ at(?v, ?from),\ at(?v, ?to)\}$.

It becomes apparent, that all actions of a shared schema have inherently similar related propositions based on the definition of action schemas and relatedness. This property is essential for the ASNets' approach to share weights, because in each layer l the actions c and d from the same action schema will be able to share the same weight matrix $W_c^l = W_d^l$ and bias vector $b_c^l = b_d^l$. This leads to ASNets' generalisation ability sharing weights among arbitrary problems of a domain.

Input layer Modules in the first input layer receive an input vector u_a^1 instead of the hidden representations of a proposition layer. This vector includes truth values for related propositions in state s, values indicating which propositions are relevant for the problem's goal and a value indicating whether an action is applicable in s.

Additionally Sam Toyer et al. experimented with different heuristic features regarding disjunctive action landmarks as further inputs, computed by the LM-cut heuristic [26]. They proved to be especially important because without these inputs the receptive field of an ASNet would be limited in the number of layers L, i.e. the ASNet would only be able to reason about chains of related actions and propositions with length at most L. This limitation is of special significance whenever the task includes chains of actions that can be arbitrarily long or are at least longer than L without repetitions of propositional values or actions.

Output layer In the end, the network has to output a probability $\pi^\theta(a \mid s)$ for every action a. Therefore the last layer of an ASNet has to compute a probability distribution over all applicable actions, ensuring that $\pi^\theta(a \mid s) = 0$ iff $pre_a \nsubseteq s$ and $\sum_{a \in \mathcal{A}} \pi^\theta(a \mid s) = 1$. This is achieved by a *masked softmax* activation function in the last output layer where the binary mask vector m indicates which actions are applicable with $m_i = 1$ iff $pre_{a_i} \subseteq s$ and $m_i = 0$ otherwise. Additionally the hidden representations $\phi_{a_i}^{L+1}$ need to be scalar values instead of d_h-dimensional vectors for the softmax function. Therefore we replace the weight matrix with a vector $W_a^{L+1} \in \mathbb{R}^{d_a^{L+1}}$ and the bias vector with a scalar value $b_a^{L+1} \in \mathbb{R}$.

The softmax activation function computes the probability π_i to choose action a_i like the following for all actions $\mathcal{A} = \{a_1, ..., a_N\}$:

$$\pi_i = \frac{m_i \cdot exp(\phi_{a_i}^{L+1})}{\sum_{j=1}^{N} m_j \cdot exp(\phi_{a_j}^{L+1})} \tag{2.6}$$

Due to the mask, a is guaranteed not be chosen if it is not applicable in s and the probability distribution function among the enabled actions is ensured by the softmax function.

Proposition modules

Proposition modules are similar to action modules but only occur in intermediate proposition layers. Therefore a hidden representation produced by the module for proposition $p \in \mathcal{P}$ in the l-th layer looks like the following

$$\psi_p^l = f(W_p^l \cdot v_p^l + b_p^l) \tag{2.7}$$

where $v_p^l \in \mathbb{R}^{d_p^l}$ is an input feature vector, $W_p^l \in \mathbb{R}^{d_h \times d_p^l}$ is a weight matrix for this proposition module and $b_p^l \in \mathbb{R}^{d_h}$ is the corresponding learned bias. f is the same nonlinearity used in the action modules.

The main difference between proposition and action modules, which also makes the input features v_p^l in proposition layers slightly more complicated, is that the number of actions related to one proposition can vary.

This can for example be seen in our transport problem. The proposition $at(truck, M)$ is related to the drive-actions $\{drive(truck, M, x),\ drive(truck, x, M) \mid x \in \{L, G, E\}\}$, while $at(truck, L)$ has only two such related actions. This is caused by the different connectivity of locations. Furthermore $at(truck, M)$ is related to the actions $load(truck, package, M)$, $unload(truck, package, M)$ and $at(truck, L)$ to $(un)load(truck, package, L)$ respectively.

To deal with this variation and be able to share weights among proposition modules, similarly to the approach for action layers, the input feature vector's dimensionality d_p^l has to be equal for all propositions with the same underlying predicate. Therefore the action schemas A_1, ..., $A_S \in \mathbb{A}$ referencing $pred(p)$, the predicate of proposition $p \in \mathcal{P}$, in their preconditions, add or delete list are collected. When building the hidden representation v_p^l of proposition p, all related actions from the listed action schemas are considered with action module representations of the same action schema being combined to a single d_h-dimensional vector. This is achieved by using a pooling function.

$$v_p^l = \begin{bmatrix} \text{pool}(\{\phi_a^l \mid \text{op}(a) = A_1 \wedge R(a,p)\}) \\ \vdots \\ \text{pool}(\{\phi_a^l \mid \text{op}(a) = A_L \wedge R(a,p)\}) \end{bmatrix} \tag{2.8}$$

This way, the dimensionality of the hidden representation $d_p^l = d_h \cdot L$ is fixed and the weight matrix W_p^l and bias b_p^l can be shared among all propositions instantiated by the same predicate, similarly to the weight sharing of actions belonging to the same action schema. $op(a)$ in equation 2.8 stands for the corresponding action schema A of action a.

2.4.2 Supervised Training

During training, the ASNet is executed on small problems from a domain to learn weights, which still lead to an efficient policy on larger problems of the same domain. In this chapter, we will cover the supervised training algorithm, using a *teacher policy*, proposed by Sam Toyer et al.

At the beginning of the training, the weights θ are initialized by employing the *Glorot initialisation*, or *Xavier initialisation* [21] using a zero-centred Gaussian distribution.

After initializing the weights, the *training epochs* are performed. During these, the state space of each training problem $\zeta \in P_{train}$ is explored. Starting from its initial state $s_0(\zeta)$, the exploration follows the network policy π^θ and either stops when $L =$

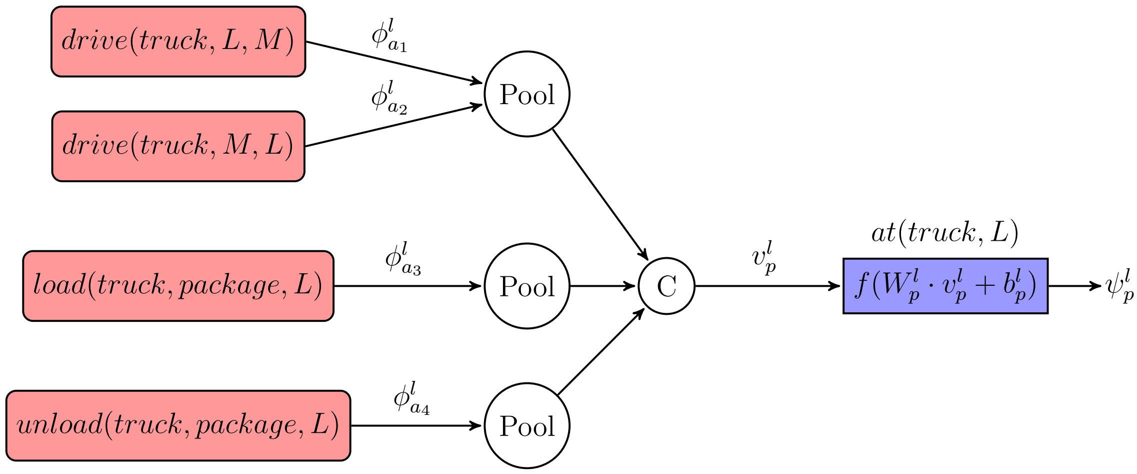


Figure 2.5: Illustration of the proposition module for $at(truck, L)$, describing $truck$ to be in London (L). The hidden representations of the related action modules are received as an input, combined with pooling and then concatenated to v^l_p and the hidden representation of the module ψ^l_p is computed as described.

$T_{trajectory-limit}$ states have been visited or a goal or dead-end has been reached. A delete-relaxed heuristic is used for efficient dead-end detection. This can significantly reduce training time. Let the set of explored states in epoch e be S^e_{exp}.

Additionally, for every $s \in S^e_{exp}$ a teacher policy (usually an optimal policy) π^* is used to extract all states which are reachable from s with nonzero probability. All these states are included in the set S^e_{opt}. Afterwards, the set of training states $\mathcal{M}$, which initially is $\emptyset$, is updated as $\mathcal{M} = \mathcal{M} \cup S^e_{exp} \cup S^e_{opt}$. The states from S^e_{opt} ensure that the network is always trained with "good" states, while the states from S^e_{exp} are important so that the network is able to improve upon its performance in already visited states.

After each exploration phase the ASNet's weights θ are updated using the loss function

$$\mathcal{L}_\theta(\mathcal{M}) = \frac{1}{|\mathcal{M}|} \sum_{s \in \mathcal{M}} \sum_{a \in \mathcal{A}} \pi^\theta(a \mid s) \cdot Q^*(s, a) \tag{2.9}$$

where $Q^*(s, a)$ is the expected cost of reaching a goal from s by following the policy π^* after taking action a. The update is performed by using *minibatch Stochastic Gradient Descent* (*SGD*) where the loss is computed with respect to smaller, randomly selected minibatches $\mathcal{B} \subseteq \mathcal{M}$ of fixed size, instead of using the whole dataset $\mathcal{M}$. This strategy can save the significant expense of computing gradients on large datasets and can generally converge faster [35]. Additionally the Adam optimization algorithm, proposed by Kingma and Ba [30], is used to update the weights θ in a direction minimizing $\mathcal{L}_\theta(\mathcal{M})$.

The exploration and learning stops when $T_{max-epochs}$ epochs are exceeded or an early stopping condition is fulfilled. The condition consists of two parts. First, the network policy π^θ must reach a goal state in at least 99.9% of the states in $\mathcal{M}$ in the most recent epoch. Secondly, the success rate of π^θ did not increase by more than 0.01% over the previous best rate for at least five epochs. This early stopping condition is rather strict and Sam Toyer et al. already mention that loosening it might improve training.

They also mention that, given the cost of computing an optimal teacher policy, they tested training ASNets using unguided policy gradient reinforcement learning like the FPG planner by Buffet and Aberdeen [9]. But in their tests, supervised learning was found to be more efficient and reliable than reinforcement learning.

2.4.3 Empirical performance evaluation

Experiment

To be able to evaluate the performance of ASNets, Sam Toyer et al. constructed an experiment comparing it to the state-of-the-art probabilistic planners LRTDP [8], ILAO* [22] and SSiPP [55]. All planners were run with the admissible LM-cut and the inadmissible h^{add} heuristic. They are limited at 9000s time and 10Gb memory. LRTDP and ILAO* are executed until they converge and SSiPP was trained for the entire time but the last 60s, which were used for evaluation.

The ASNet was trained for each domain using small problem instances and evaluated 30 times on each problem. The networks were always constructed with three action and two proposition layers, $d_h = 16$ for each module, ELU [12] used as the nonlinear activation function. A learning rate of 0.0005 and a batch size of 128 was utilized for the Adam optimization. Additionally, L_2 regularization with $\lambda = 0.001$ on the weights and dropout [51] with $p = 0.25$ on the outputs of all intermediate layers was used to prevent overfitting. The whole training was limited at two hours. As the teacher policy in the ASNets LRTDP with h^{LM-cut} and LRTDP using h^{add} was employed. While the ASNets run with the optimal LM-cut policy were always provided with additional heuristic inputs from the LM-cut heuristic, ASNets guided by the suboptimal h^{add} policy were once executed with and once without these input features.

The experiment used three different probabilistic planning domains: CosaNostra Pizza [52], Probabilistic Blocks World [56] and Triangle Tire World [36].

In his thesis, Sam Toyer also analysed ASNets' performance on deterministic classical planning. As baselines, multiple heuristic search planners were implemented on the Fast Downward framework [25]. Greedy best-first search (GBFS) and A* were both used with h^{LM-cut} and h^{add}. Furthermore LAMA-2011 [44] and LAMA-first, which won the IPC 2011, were used as baselines.

For deterministic planning, Sam Toyer used the Gripper domain [37] for the experiment.

Results

Unsurprisingly, ASNets performed better on large problems compared to smaller ones. The necessary training of the network is too expensive for most smaller problems to compete with the state-of-the-art baselines. On the other side, ASNets heavily outperformed all baseline planners in the CosaNostra Pizza and Triangle Tire World domain on large problem instances. The networks were even able to learn an optimal or near optimal policy for many problems of both these domains.

Additionally, it is worth noting that the ASNet using a more efficient, but suboptimal h^{add} policy was still able to perform well and even exceeded the performance of ASNets with an optimal LM-cut policy on the Probabilistic Blocks World domain. The reason is probably the high expense of problems in this domain and ASNets with the LM-cut policy were unable to scale sufficiently well. But all ASNets needed the heuristic input features in the Blocks World domain to be able to consider long proposition and action chains. This was necessary to perform well in this complex domain.

In deterministic planning, ASNets took significantly more time to train and evaluate compared to most baseline planners. Only the A* planners did not converge for problems of a size larger than 15. While the solutions of both ASNets using heuristic input features were found to be optimal on all problems, the ASNet only using the h^{add} policy without additonal heuristic input was unable to solve even problems of medium size. The training of this network was faster, but its confidence in the correct actions was too low to generalise well. The LAMA planners outperformed all ASNets in the problems, considered by Sam Toyer, finding optimal solutions significantly faster.

However, it should be noted that the experiment primarily focused on probabilistic planning and the classical planning part was merely to show the ability of ASNets to be executed on these tasks. To measure and evaluate the performance of ASNets in classical deterministic planning, a comprehensive experiment would still be needed.

Chapter 3

Related Work

ASNets are certainly an exiting and promising application of learning in form of neural networks to automated planning. However, there is more than just this one approach.

Shahab Jabbari Arfaee, Sandra Zilles and Robert C. Holte proposed a learning strategy in 2011 which combines multiple heuristic functions and a neural network to evaluate for large state spaces [28]. They started with a heuristic function simply representing the maximum of a set of input features. These inputs were specifically chosen for the domains, but are all fairly simple measures, e.g. they used the manhatten distance and five pattern database heuristic functions [17, 24] among others for the sliding-tile-puzzle domain. The initial heuristic was incapable of solving any problems in reasonable time. Using this heuristic a search on problem instances was conducted to extract states along plans leading to a goal. Arfaee et al. then used the original input features on the sampled training states together with their goal distance as the input for a neural network. The network was therefore trained to compute a heuristic itself which was combined with the initial heuristic function by maximizing over their values for each state. With this algorithm, it was possible to iteratively improve the heuristic function which led to impressive results for the evaluated domains sliding-tile-puzzle, pancake-puzzle, Rubik's Cube and blocksworld.

To collect valuable training states during the heuristic search, even if the used heuristic function is not performing well yet, they also used random walks from the goal backwards. This method is of significant importance to make progress, especially at the very beginning of the training process. Despite the performance, it should be noted that preselected input features had to be chosen for each domain. Even if these inputs are kept simple, it represents a major limitation to extend this approach to further domains. Additionally, this approach combines multiple, already existing heuristic functions rather than extracting the final heuristic directly from the network.

In contrast, Christian Bohnenberger trained neural networks to directly compute a heuristic function for given states of a planning task in his bachelor thesis [6]. The network input was, similar to ASNets, a binary vector, indicating the truth values of STRIPS propositions. However, the approach did not include the goal nor applicable values for actions in the input. The performance of regression and classification networks were compared using a variety of configurations including early stopping, dropout, different batch sizes and activation functions. For learning, a supervised training algorithm was used where the input states were labelled with their h^* values. The evaluation showed that the neural networks were able to learn almost optimal heuristics using this approach but it should be noted, that the entire investigation was only conducted for a single problem of the transport domain. Therefore, the generalisation capability of this approach is still uncertain.

Patrick Ferber followed a similar approach in his master thesis regarding the use of neural networks as heuristic guidance in search for planning [18]. In this work, multiple neural network architectures were trained to compute a heuristic function which provides a good distance measure to the goal. This approach led to almost optimal heuristics while being significantly more efficient than current optimal heuristics like e.g. LM-cut. Besides these networks, the thesis also includes extensive studies on sampling strategies used to extract states with corresponding goal-distances as labelled training data. Approaches like forward, backward and random walks all have different strengths and were evaluated regarding the size of the obtained training set and its quality which is often neglected. A large data set is not necessarily useful when its states are not representative for the entire state space of the task which should be the main goal of sampling. Lastly, the work addresses the training process for neural network heuristics comparing a variety of training as well as network configurations. Overall, it was found that the neural network heuristics were able to perform well on multiple domains while the optimal configuration regarding sampling and training of the networks varied among the benchmarks. However, it is important to note, that this approach is separately trained and exploited on problem instances and unlike ASNets is incapable of generalising among problems of a common domain.

Chapter 4

Network Definition

The main contribution of this thesis will be the implementation and evaluation of Action Schema Networks for classical planning based on the Fast-Downward planning system. Prior to the integration of the network in this system, the training and evaluation, we have to first define the network. In Section 2.4.1, we already explained the architecture proposed by Sam Toyer, showing that the model structure is directly dependent on a given PDDL domain and problem. In this chapter, we will explain the necessary Fast-Downward foundation for this process. Following, an efficient strategy to compute the relations between actions and propositions of the task will be explained and lastly the concrete definition of the network will be outlined.

4.1 PDDL in Fast-Downward

As mentioned in Section 2.1, planning tasks are usually represented in PDDL, separated in the domain and problem file. The Fast-Downward planning system uses these in its translation process to build an internal representation of a PDDL task containing its abstract action schemas and predicates, instantiation capabilities to extract groundings as well as the initial and goal states. Afterwards, the task is simplified by normalization techniques removing any universal and existential quantifiers potentially included in conditions of the tasks.

4.1.1 Instantiation

To instantiate action schemas and predicates obtaining grounded actions and propositions, a PROLOG [13, 32] model is used with the major advantage that instantiations, which are theoretically possible with the objects given in the PDDL problem but can never be reached, are not constructed. The impact of this step should not be underestimated, especially in the context of ASNets. E.g. in the transport domain, the *drive(?v, ?from, ?to)* action schema would usually be instantiated with every vehicle, and pair of locations without considering their connections. Similarly the *connected(?l1, ?l2)* predicate would be instantiated with every pair of locations leading to $|L|^2$ such propositions with $|L|$ being the number of locations in the task. The truly existing connections in a problem would all be given in the initial state and there is no action adding any such *connected(x, y)* facts, i.e. all these propositions which are not included in the initial state can never become true. As ASNets contain an action and proposition module for each grounded action and proposition respectively, pruning such unreachable propositions can sometimes avoid immense blow-up of the network size.

Imagine a transport task with two trucks and eight locations connected in a circle, so that each location is connected to its two neighbours.. The task contains 8 streets in total which can be driven by each truck in both directions, leading to $8 * 2 * 2 = 32$ *drive* actions. Naively instantiating all possible groundings would lead to $2 * 8 * 8 = 128$ such actions. Similarly, the number of *connected* propositions would be reduced from 64 to just 16. This has significant impact on the network size of ASNets and therefore improves their scalability.

4.1.2 PDDL - SAS compatibility

For the planning process, Fast-Downward internally uses a task representation based on the SAS$^+$ formalism [3] including variables with domains instead of propositions. However, variable-value pairs represent facts which directly correspond to propositions in the PDDL task. During the Fast-Downward translation from PDDL to SAS$^+$ further propositions, which remain constant throughout the whole task but were not previously removed during the simplification steps, are pruned. Therefore, we identify and remove the corresponding propositions in the PDDL task representation, so that the number of facts in the SAS$^+$ representation used during search matches the number of propositions in the corresponding PDDL task.

This is essential for the network definition using PDDL, because we later use the ASNet policy during a Fast-Downward search. Hence, the number of expected input and output values, connected to the number of propositions and actions, during the network model creation has to be equal to the number of input values fed into the network and its output size during search.

However, we do still want to consider the constant propositions which is why we identify these in the first action input layer of the model and extract their truth values from the initial state of the task.

4.1.3 Relations

The PDDL task itself is important for the network structure because the relations between groundings and prior abstract action schemas and predicates need to be determined. These abstracts [1] do not exist in the SAS$^+$ formalism which is why the PDDL representation is necessary. Recall that a proposition $p \in \mathcal{P}$ is said to be related to action $a \in \mathcal{A}$ iff p appears either in pre_a or eff_a [2].

The definition of relatedness is used for the sparse connectivity of ASNets, one of the core components of its architecture. The relations between concrete propositions and actions can efficiently be derived from relations of the underlying abstract action schemas and predicates. Therefore, we first compute the relations between abstracts by going through the preconditions and effects of the action schemas. Each predicate contained in these, is by definition related to the action schema. Due to the commutativity of the

[1] In this section, we refer to action schemas and predicates as abstracts in contrast to groundings standing for instantiated actions and propositions.

[2] Note that a PDDL domain separates the action properties in preconditions and an effect list which is why the relatedness is here based on those. This is identical to the definition found in Section 2.4.1 which was based on the STRIPS planning description distinguishing between positive (in the add list) and negated effects (in the delete list).

relatedness property, we can directly deduce the relations in both directions. To obtain relations between grounded propositions and actions, we extract the objects from the PDDL problem used to instantiate the underlying action schema' arguments to receive the actions. These mappings from abstract arguments to objects can be represented as functions which can be used to also instantiate the related predicates of the action schema obtaining the propositions related to the instantiated action.

An example for this process in the transport domain for the schema $drive(?v - vehicle, ?from - location, ?to - location)$ is visualized in Table 4.1.

$$A = \{?v \to truck, ?from \to L, ?to \to E\}$$

	abstracts		groundings
action (schema)	$drive(?v, ?from, ?to)$	$\xrightarrow{A}$	$drive(truck, L, E)$
related	$connected(?from, ?to)$	$\xrightarrow{A}$	$connected(L, E)$
predicates/	$at(?v, ?from)$	$\xrightarrow{A}$	$at(truck, L)$
propositions	$at(?v, ?to)$	$\xrightarrow{A}$	$at(truck, E)$

Table 4.1: Illustration of $drive(?v - vehicle, ?from - location, ?to - location)$ instantiated to the action $drive(truck, L, E)$ using the mapping A from arguments of the action schema to concrete objects in the task. Below the corresponding instantiation of related predicates to propositions is shown.

For propositions the related actions are grouped by underlying action schemas for the pooling operation applied in proposition modules as indicated in Section 2.4.1.

4.2 Keras

Based on the modified PDDL task and the deduced relations, the ASNet model can be defined. We used *Keras* [11] with the *Tensorflow* [1] backend to define and train the networks. *Keras* is a python library serving as an API to multiple machine learning libraries, in our case *Tensorflow*, offering a modular approach with high-level abstraction. This makes *Keras* model definitions comparably simple to read and write as well as easily extendible. During our experiments, we used *Keras* version 2.1.6 with *Tensorflow* 1.8.0.

Generally, the ASNet structure can be separated in action and proposition layers. While there are action and proposition modules in the respective layers for each grounded action and proposition, the weights are shared among all modules corresponding to an instantiation of the same abstract in one layer. Therefore we distinguished between input layers for each module computing its input tensor and the main module corresponding to an abstract action schema or predicate holding the respective weights which is reused among all instantiations of the respective abstract in the given layer. All the following modules were implemented as custom *Keras* layers.

4.2.1 Action modules

The input computation for action modules is separated for the first and intermediate layers. While for action modules in the first layer input values are obtained from the network's input, for intermediate layers these are extracted for related propositions out of the concatenated output of the last proposition layer.

The following main action module, containing the weights and bias, is shared among all action modules in one layer which have the same underlying action schema. These components compute the module output as already outlined in Equation 2.4. However, it should be noted that action modules in the last action layer prior to the softmax function output scalar values rather than tensors of the fixed hidden representation size d_h and also avoid applying any nonlinear activation function.

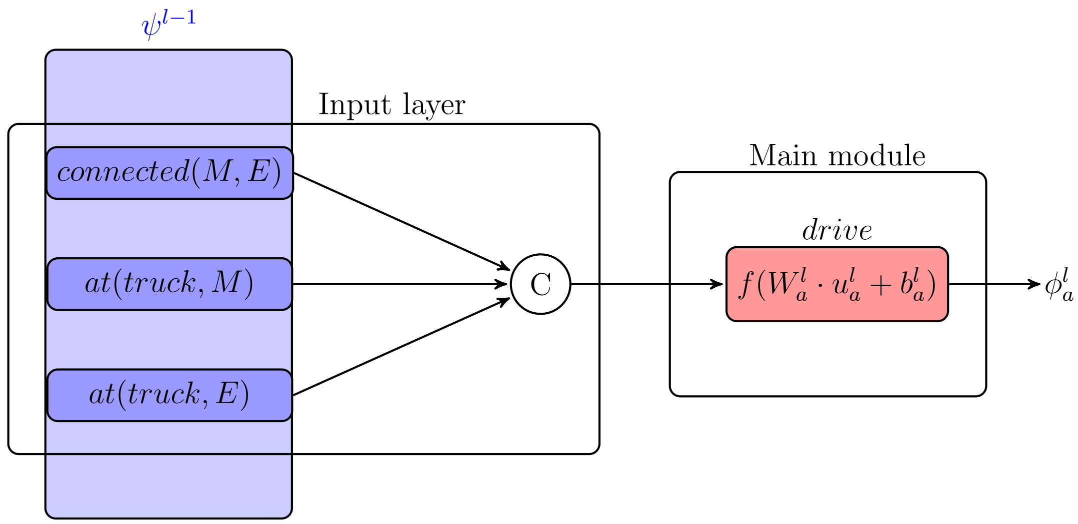


Figure 4.1: Illustration of intermediate action module for $drive(truck, M, E)$ showing the distinction of input and main module. Note that the input layer is specific for the grounded action and extracts the necessary output values out of the entire previous proposition layer output ψ^{l-1}. The main module is shared among all modules of actions instantiated from the *drive* action schema.

4.2.2 Proposition modules

The proposition modules are split into input and main layers in the same way as action modules. As there are only intermediate proposition layers, there is no further distinction needed.

While the general structure of input and main modules for propositions is almost identical to action modules, pooling operations are applied on all input tensors of related actions sharing the same underlying action schema. It is possible, that the abstract predicate of proposition p is related to an action schema for which no instantiated action is related to p. In this case the pooling operation would be computed among the empty set of tensors leading to a tensor with only zero entries and the necessary hidden representation size d_h.

In the main proposition module the computation using the weights and bias, shared among all proposition modules of the same underlying predicate in one layer, is applied.

4.2.3 Softmax output layer

ASNets compute a policy and therefore the final network output needs to be a probability distribution over the set of actions indicating how likely the network would choose each action in the state represented by the network input. This is achieved by applying a masked softmax function to the last action layer output with scalar values for each action.

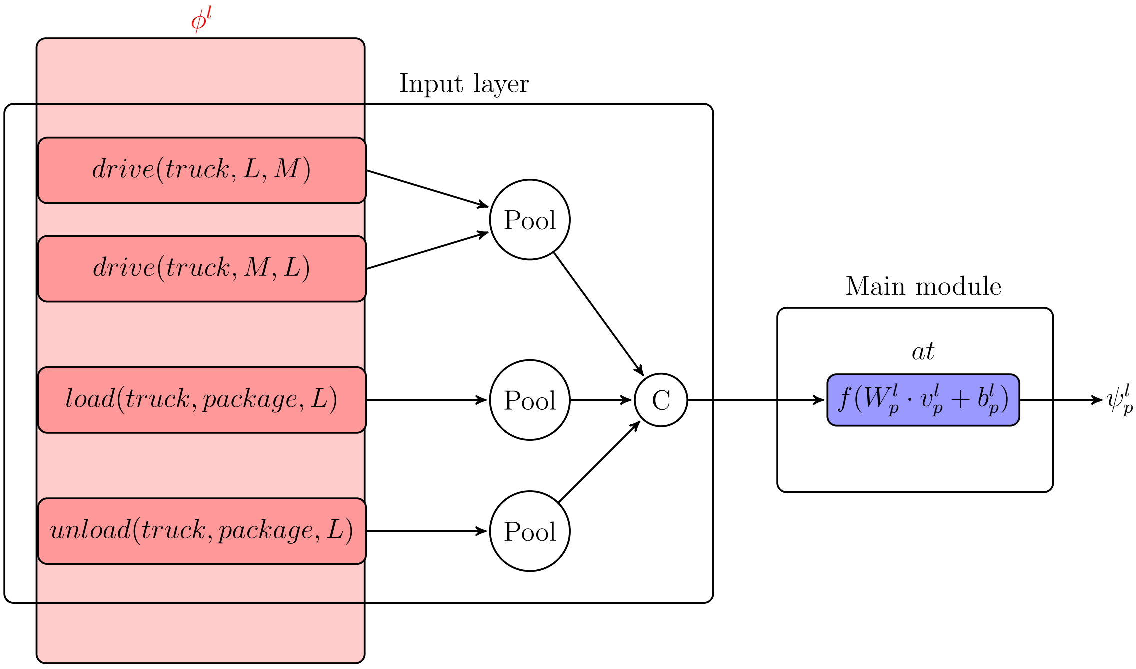


Figure 4.2: Illustration of proposition module for $at(truck, L)$. The input layer is entirely individual for the proposition and extracts the output values of related actions out of the entire previous action layer output ϕ^l. The main module is reused among all modules of propositions instantiated from the at predicate.

The function additionally uses the input values showing which actions are applicable as an input mask. The exact formula can be found in Equation 2.6. Its most important property is that it guarantees zero probabilities for all inapplicable actions. Therefore, a distribution among all applicable actions only is computed.

Chapter 5

Training

In chapter 4, we explained how to obtain the Action Schema Networks capable of learning policies for planning tasks. To be able to exploit the networks during search solving problems of a given domain, we have to acquire knowledge first. This knowledge acquisition is achieved by training the networks, so we continuously update the parameters θ of ASNets including the weight matrices and bias vectors with the goal of improving the network policy guidance. ASNets for a given domain can share the weights because they involve the same action schemas and predicates defined. Therefore, it is possible to train the networks on small problem instances from a domain and afterwards exploit its learned policy on all problem instances based on the same domain. This is essential for ASNets generalisation capability.

5.1 Training cycle

Our training algorithm is mostly based on the proposed supervised training of Sam Toyer et al. explained in detail in his thesis [53]. However, we made minor modifications for the usage in classical planning rather than probabilistic planning which was the focus of the previous work. The algorithm iterates over $T_{max-epochs}$ epochs and trains the network for each given training problem in P_{train}. Therefore, the network is first build for the current problem instance and then $T_{prob-epochs}$ problem epochs are executed. These involve sampling of states using the network search $\mathcal{S}^{\theta}$ and a given teacher search $\mathcal{S}^{*}$. This process will be explained in detail in Section 6.1. After the sampling $T_{train-epochs}$ training epochs over the set of sampled states $\mathcal{M}$ are executed to optimise the weights θ based on $\mathcal{M}$.

5.1.1 Initialisation

As indicated in Algorithm 1, we save and load the already trained weights to reuse them for all problem networks used during training. However, in the first iteration there are no weights available yet, i.e. we have to initialize our weight matrices and bias vectors. Parameter initialization in machine learning applications is a science on its own. We followed the suggestion of Sam Toyer, using the *Glorot* or *Xavier initialisation* [21] to set our weight matrices values. This initializer has proven to be highly valuable for deep neural networks, especially when using the RELU activation function (or some modification of it). It initializes the weights with small values from a zero-centered Gaussian distribution using an adaptive variance depending on the number of input and output nodes for our parameters.

The bias vectors are simply initialised with zero values.

Algorithm 1 Training Cycle on set of training problems P_{train}

procedure TRAIN(P_{train})
  $\mathcal{M} \leftarrow \emptyset$ ▷ Sampled states
  $n_{epoch} \leftarrow 0$ ▷ Epoch counter
  while $n_{epoch} < T_{max-epochs}$ and not early stopping **do**
    for all $p \in P_{train}$ **do**
      $asnet_p \leftarrow$ BUILD-MODEL(p, weights) ▷ ASNet model
      for $n_{p-epoch} = 1, ..., T_{prob-epochs}$ **do** ▷ Run problem epochs
        $\mathcal{M} \leftarrow$ SAMPLE(p) ▷ Sample on p
        TRAIN($asnet_p, \mathcal{M}, T_{train-epochs}$) ▷ Train network
      weights $\leftarrow asnet_p.save\_weights()$
      $\mathcal{M} \leftarrow \emptyset$
    $n_{epoch} \leftarrow n_{epoch} + 1$
function BUILD-MODEL(p, weights) ▷ Build ASNet model and load weights
  task_meta $\leftarrow$ COMPUTE_META(p) ▷ Compute task meta information
  $asnet_p \leftarrow$ CREATE_MODEL(task_meta, p)
  if weights exist **then**
    $asnet_p.load\_weights$(weights) ▷ Load weights if existing
  return $asnet_p$

5.1.2 Epochs

During the training cycle we make use of three levels of iterations. First, we execute $T_{max-epochs}$ epochs in which we train the ASNets for every given problem in our training set P_{train}. This alternating training on problems allows the networks to continuously improve the weights without becoming too specialised and therefore also limited to any of the training problems which is essential for the networks generalisation ability beyond the training problems in P_{train}.

For each problem p in a epoch, we run $T_{prob-epochs}$ problem epochs after building the network for p in which we train the network. The reason for these problem epochs is very practical. Building the network can take considerable time for problem instances involving large amounts of grounded actions or propositions. Therefore, it is more efficient to run multiple sampling and training sessions before going to the next problem for which we would need to build a network again. Without these problem epochs, we might spend a large portion of the "training time" with building ASNet models instead of sampling and improving our weights based on the obtained samples for some domains.

Lastly, during training we go over the set of sampled training states for $T_{train-epochs}$ epochs. This is a core technique applied in most training algorithms for neural networks making the most out of the sampled data. Usually, the sampling process takes significantly more time than the training steps themselves, so it is important to make noticeable progress during one training run.

5.1.3 Early stopping

For some domains, good policies can be learned fairly easy and also quickly. It is unnecessary to finish all $T_{max-epochs}$ epochs when the network is already performing very well, so to save computational power and time we use the early stopping criteria suggested by Sam Toyer. It includes two conditions that have to be met for the training cycle to terminate early. First, a certain percentage of network searches $\mathcal{S}^\theta$ during the sampling process have to be successful, i.e. reaching a goal. Secondly, this success rate should have not improved by more than 0.01% over the best, previously reached success rate for a certain amount of epochs. However, it was already mentioned in the thesis of Sam Toyer that the thresholds might be too restrictive. Therefore, we terminate early when at least 95% of the network searches reach a goal and the success rate did not improve by more than 0.01% for 3 epochs or more. For comparisons, in his thesis Sam Toyer used thresholds of 99.9% and almost no improvements for at least 5 epochs.

5.2 Loss Functions

During the training steps, we try to improve the current weights θ to ensure that "good" actions are chosen in the sampled states from $\mathcal{M}$. Mostly, the measure of quality for action choices is whether the teacher search $\mathcal{S}^*$ used during the sampling process chose the actions. This is achieved by attempting to minimise a given loss function.

We mainly considered the typical binary crossentropy loss function, which is the negation of the loss function already proposed by Sam Toyer et al. in their paper about ASNets [54]:

$$\mathcal{L}_\theta(\mathcal{M}) = \sum_{s \in \mathcal{M}} \sum_{a \in \mathcal{A}} -(1 - y_{s,a}) \cdot log(1 - \pi^\theta(a \mid s)) - y_{s,a} \cdot log\ \pi^\theta(a \mid s) \tag{5.1}$$

In the function we iterate over all sampled states $\mathcal{M}$ and all actions $\mathcal{A}$ with $\pi^\theta(a \mid s)$ being the probability with which the network policy would choose action a in state s. The binary value $y_{s,a}$ is 1 if action a starts an optimal plan from state s onwards according to the teacher search $\mathcal{S}^*$. The exact acquisition of these values will be explained in Section 6.1.

The loss function should be reaching its global minimum when the network search $\mathcal{S}^\theta$ matches the teacher search $\mathcal{S}^*$ which is our point of reference. Any deviation should be "punished" with a higher loss $\mathcal{L}_\theta$. When evaluating the loss function, we can analyse three separate cases:

1. $\pi(\mathbf{a} \mid \mathbf{s}) = \mathbf{0}$ **and** $\mathbf{y_{s,a}} = \mathbf{0}$:
   In this case our network would never choose the action which is also not considered optimal by the teacher search. This would lead to the following term in the loss function:

$$\begin{aligned} -(1-0) \cdot log(1-0) - 0 \cdot log(0) &= -1 \cdot log(1) - 0 \cdot log(0) \\ &= 0 \end{aligned}$$

   Note, that we consider $log(0)$ not to be $-\infty$ but rather a large negative number $\ll 0$. In the implementation, this is achieved by clipping the network probabilities, so that they always stay in the interval $(0, 1)$ without ever reaching 0 nor 1. Such a technique is often necessary for numeric stability avoiding the risk of overflowing or underflowing numbers. Usually this would also mean, that $log(1)$ is not 0 but a very small negative number. We only want to give an intuition why the loss function is effective for our purpose. Therefore we keep it simple and consider $log(1)$ to be 0.

This evaluation by the loss function seems fitting given that we try to learn optimal policies according to the teacher search to reach its performance, so we do not want to choose suboptimal actions.

2. $\pi(\mathbf{a} \mid \mathbf{s}) > \mathbf{0}$ **and** $\mathbf{y_{s,a}} = \mathbf{0}$:
Here, the network policy would choose the action a for some non-zero probability despite a not being labelled optimal by the teacher search. This case should be punished as we aim to learn only the optimal paths chosen by the teacher search. The higher the probability to choose the not optimal action a, the higher the loss should be which is fulfilled by the loss function:

$$-(1-0) \cdot log(1-\pi(a \mid s)) - 0 \cdot log\ \pi(a \mid s) = -log(1-\pi(a \mid s))$$

$(1-\pi(a \mid s)) \in [0;1]$, so $-log(1-\pi(a \mid s)) \in [0,\infty]$. As we can see, the higher the network probability for a, the closer does $1-\pi(a \mid s)$ get to 0 leading to a loss of ∞ or a very large loss considering clipping which is the desired effect given that we do not want to choose suboptimal actions.

3. $\pi(\mathbf{a} \mid \mathbf{s}) \geq \mathbf{0}$ **and** $\mathbf{y_{s,a}} = \mathbf{1}$:
In this third and last case the network probability is larger or equal to 0 for an optimal action according to the teacher search. Obviously, the higher the network probability for an optimal action the better. This is represented in the loss function:

$$-(1-1) \cdot log(1-\pi(a \mid s)) - 1 \cdot log\ \pi(a \mid s) = -log(\pi(a \mid s))$$

Similarly to the second case, it can be stated that $\pi(a \mid s) \in [0;1]$, so $-log(\pi(a \mid s)) \in [0,\infty]$ with a zero loss being reached in the case of $\pi(a \mid s) = 1$ which is the desired outcome.

We implemented a second loss function, which was also included in the implementation of Sam Toyer [1], closely related to the first binary crossentropy loss:

$$\mathcal{L}_\theta(\mathcal{M}) = \sum_{s \in \mathcal{M}} \sum_{a \in \mathcal{A}} -y_{s,a} \cdot log\ \pi^\theta(a \mid s) \tag{5.2}$$

This loss only includes the second part of the first loss and therefore exclusively considers actions with $y_{s,a} = 1$, i.e. actions which start an optimal plan according to the teacher search, without directly punishing any probabilities for suboptimal actions.

[1] The source code of Sam Toyer for the ASNet paper of AAAI'18 is available at https://github.com/qxcv/asnets

Chapter 6

Fast-Downward Extensions

Our goal is to exploit the acquired knowledge, gained during training, in search based on the network policy. So far, Fast-Downward did not include policies in its framework which is why we extended the system with this functionality. Additionally, the sampling process during the training executes multiple searches to obtain the set of sampled states. These search executions make use of the Fast-Downward system, which is why the sampling process itself is implemented in this planning framework.

6.1 Sampling

To be able to train a network based on supervised learning, labelled data, which includes not only the network input values but usually also the expected outcome, is needed. The data is necessary to compute loss functions which include the network output, often called prediction, and the expected output. Obtaining such data is the main challenge whenever using supervised learning and is achieved by sampling. Therefore generally, a sampling algorithm aims to collect labelled data used during the following supervised learning.

6.1.1 Sample representation

For ASNets, the collected data must not only include information regarding the state used as the network input but also all further teacher search information necessary to compute the loss. With respect to our loss functions mentioned in Section 5.2 we have to include the $y_{s,a}$ values for the sampled states which indicate whether action a starts an optimal plan from s according to the teacher search. Therefore, a sample for state s can be represented by a quadruple (g, t_s, a_s, y_s) where all entries form lists of binary values. g contains values for each fact indicating whether it is part of the planning task goal. The values in t_s show which facts are true in s, the values in a_s indicate which actions are applicable in s and y_s contains the $y_{s,a}$ value for each action. Note, that g can be shared among all sampled states for one planning problem.

g, t_s and a_s are used as the input values for the network to receive the probability distribution from the policy for state s, $\pi^\theta(s)$, which together with y_s is used to compute the loss during training.

It is important for the network in- and output that the ordering of the binary values for facts and actions is consistent among all samples, e.g. the ith binary value of g and t_s have to correspond to the same fact while likewise the jth value of a_s, y_s and $\pi^\theta(s)$ must refer to the same action. This is achieved by sorting the facts and actions lexicographically according to their names.

6.1.2 Sampling search algorithm

The sampling search is one of the core components used during the training, outlined in Section 5.1. The entire sampling process for a given problem p, illustrated in Algorithm 2, can be divided into two phases. First, we explore the problem's state space by applying the network search $\mathcal{S}^\theta$, which is based on the previously built ASNet and naively follows its policy. We start at the initial state s_0 and follow the most probable action in each state according to the network's policy π^θ until we either reach a dead-end, an already encountered state or a goal. Note that expanding a search node of any state for a second time would lead to a circle which is why we terminate the exploration in this case. Afterwards, all states $s_0^\theta, ..., s_N^\theta$ along the explored trajectory during search are collected (represented by extract_search_states in the pseudocode). For every extracted state the sample values t_s, a_s and y_s need to be acquired (with extract_sample_values) before the quadruples are added to the sample data $\mathcal{M}$. The corresponding goal values g for the given problem are computed beforehand as they are shared among all samples.

The states sampled during this step are essential to improve the network policy for states already encountered. However, they do not provide reliable guidance towards the goal, especially not at the beginning of the training process when weights and therefore π^θ is randomly initialized and not trained yet.

To ensure that states forming good trajectories solving the task are identified and learned, we additionally sample such states by using the teacher search $\mathcal{S}^*$. In this phase, we start $\mathcal{S}^*$ from all states collected during the previous exploration and similarly to the first phase sample the goal trajectory. If no goal was found during the search, the trajectory to the last expanded state will be used instead because it usually represents the "closest state to a goal" encountered.

Algorithm 2 Sampling search on problem p

function SAMPLE(p) ▷ Sample exploration and teacher search states
    $\mathcal{M} \leftarrow \emptyset$
    $g \leftarrow$ GET_GOAL_VALUES(p)
    $s_0^\theta, ..., s_N^\theta \leftarrow$ EXTRACT_SEARCH_STATES($s_0(p), \mathcal{S}^\theta$) ▷ Network exploration
    for $s = s_0^\theta, ..., s_N^\theta$ **do**
        $(t_s, a_s, y_s) \leftarrow$ EXTRACT_SAMPLE_VALUES(p, s)
        $\mathcal{M} \leftarrow \mathcal{M} \cup \{(g, t_s, a_s, y_s)\}$
    for $s^\theta = s_0^\theta, ..., s_N^\theta$ **do** ▷ Teacher sampling
        $s_0^*, ..., s_N^* \leftarrow$ EXTRACT_SEARCH_STATES($s^\theta, \mathcal{S}^*$)
        for $s = s_0^*, ..., s_N^*$ **do**
            $(t_s, a_s, y_s) \leftarrow$ EXTRACT_SAMPLE_VALUES(p, s)
            $\mathcal{M} \leftarrow \mathcal{M} \cup \{(g, t_s, a_s, y_s)\}$
    return $\mathcal{M}$

While the extraction of the sample values g, t_s and a_s is straight forward and only requires looking up fact values in the goal set, s or checking which actions are applicable, obtaining the $y_{s,a}$ values for s is less trivial. In order to determine which actions start an optimal plan, with respect to $\mathcal{S}^*$, we first compute a plan from s onwards with $\mathcal{S}^*$ and store its cost $cost_s$. For each applicable action a in s we then extract the state s' reached by applying a to s and execute a teacher search from s', again storing the cost $cost_{s'}$. If $cost_{s'} + cost(a) \leq cost_s$ then choosing action a in state s did not increase the plan cost according to $\mathcal{S}^*$, i.e. a starts an optimal plan with respect to $\mathcal{S}^*$ and therefore $y_{s,a} = 1$. Otherwise, a does not start an optimal plan of $\mathcal{S}^*$ or is not even applicable. In both these cases we set $y_{s,a} = 0$.

6.1.3 Teacher search

One of the main advantages of this sampling search is its flexibility because it can be used with an arbitrary search configuration implemented in the Fast-Downward planning system. Due to the popularity of Fast-Downward for heuristic search planners, many heuristic functions and search engines were already implemented and can therefore directly be used as guidance in the sampling search. In contrast, Sam Toyer used a teacher policy for the probabilistic planning application. Using a teacher policy instead of an entire search would also be possible for classical planning, but only limit the approach and therefore does not seem expedient.

6.2 Policies and Search

However, for the integration of ASNets in Fast-Downward it is necessary to implement policies representing the network output of ASNets. Previously, the classical planning Fast-Downward system only considered heuristic functions as valid evaluators used for guidance during search. We extended the system with a general framework for policies as a second form of evaluation simplifying any further additions of policies.

6.2.1 Policies in Fast-Downward

We decided to integrate policies in Fast-Downward based on the already existing concept of preferred operators. These could previously be computed by heuristic functions and e.g. indicate which actions should be prioritized during search. We extended this concept with further preferences which can be provided but are not mandatory.

For policies, we simply provide the preferred operators in evaluation results as the actions considered by the policy and give the action probabilities as the corresponding preferences. If no such preferences are given, the actions will all be treated uniformly distributed, i.e. they are all equally probable.

Network policy

For the specific case of ASNets, a network policy was implemented which serves as an interface to network classes computing policies. For ASNets, we provide a network representation in the Fast-Downward framework which is able to feed input values into a *Protobuf* network containing an ASNet model. Similarly, the ASNet network is also capable of extracting the computed network output out of the *Protobuf* model. Whenever an ASNet policy is used, we first construct the corresponding *Protobuf* network file out of the built *Keras* model. Afterwards, the ASNet representation in Fast-Downward serves as an interface to the built *Protobuf* model and feeds input values into the network as well as extracting its output as requested by the network policy.

6.2.2 Policy search

Lastly, we needed a new search engine for policies as all currently available searches in Fast-Downward are based on heuristic functions. The policy search implemented is a naive approach and simply follows the most probable action for each state according to the given policy. While this search is very simple and potentially limits the results achieved, it is also solely reliant on the policy. Therefore, using such a naive search with the ASNet policy allows us to purely evaluate the performance and quality of the given network policy. However, it will certainly be of interest in the future to combine network policies with already established and new approaches in more sophisticated searches for classical planning.

Chapter 7

Evaluation

We defined Action Schema Networks using *Keras*, integrated them into the Fast-Downward system and proposed a slightly modified training algorithm for application in classical planning. Now, we will evaluate the performance of ASNets for this planning field.

As stated in Section 2.4.3, Sam Toyer already conducted an empirical evaluation of the ASNet performance but focused on the probabilistic planning branch. While he also considered classical planning, comparing ASNets to multiple baseline planners, this experiment was only performed for the Gripper domain which is solved fairly easily by most considered planners if not all. Therefore, we will conduct an extensive empirical evaluation for ASNets in classical planning considering multiple domains of varying complexity and comparing the performance to successful planners from the optimal and satisficing planning branch.

7.1 Evaluation objective

During the experiment and its evaluation, we will address the following questions regarding the suitability of ASNets:

1. **Are ASNets able to learn good and potentially even optimal policies with respect to the teacher search?**
   We will use optimal and satisficing search configurations as teacher searches during the sampling process. While we expect that the quality of ASNet policies will depend on the quality of the plans found by the teacher search, it is of great interest to observe to which extent the network policies can keep up with the teacher search.

2. **On which domains do ASNets perform well?**
   While this question seems trivial once the results of the experiments are presented, it is important to find common properties or concepts shared among domains on which ASNets perform especially well or disappointing. Such findings can help making further progress in improving learning techniques for automated planning as a whole.

3. **For which period of time do we need to train ASNets until they perform reasonably well?**
   To which extent does the needed time vary among domains and eventually used teacher searches? It can be assumed that the necessary training time varies significantly depending on the used configuration and planning domain. Additionally, we will observe whether longer training necessarily improves the network performance.

All these questions will be addressed during the evaluation of the experiment and should be kept in mind throughout the chapter.

7.2 Evaluation Setup

The entire evaluation for all baseline planners and trained ASNets as well as the training itself was conducted on a x86-64 server with each process using a single core of a Intel Xeon E5-2650 v4 CPU clocked at 2.2GHz and 96GiB of RAM.

7.2.1 Domains and Problems

To be able to reliably evaluate ASNets for the classical planning field, we use eight domains with different characteristics and difficulties. These are mostly from previous iterations of the International Planning Competition (IPC) and the FF-Domain collection[1] of Prof. Dr. Hoffmann. In the following paragraphs, we will provide a brief description for each domain used during the evaluation and state our expectation regarding the difficulty of policy learning.

Blocksworld The Blocksworld domain describes problems in which blocks on a table need to be stacked to match a certain formation. Blocks can be stacked on top of other blocks or lay on the table. In order to rearrange the blocks, a gripper arm is used. It can grab any blocks, which do not have another stacked on top of them, and put them down on the table or stack them on top of another block. The initial state of Blocksworld problems describes the starting arrangement while the goal specifies relations among blocks which have to be fulfilled.

We use the 35 problem instances used in track 1 of the IPC 2000, which involve problem instances with 4 up to 17 blocks, for the evaluation and train the ASNets on three instances with 4 blocks only.

Generally, there exist fairly simple strategies for solving Blocksworld problems. E.g. one can always unstack all blocks until they lie on the table. From there, it is trivial to just stack the blocks until we have built the configuration specified in the goal. This mostly does not result in an optimal plan, but can be applied for any arbitrary initial arrangement. However, none of the usual heuristic searches used as teachers or in the baseline planners make use of these strategies which is why we do not expect the network policy to learn such an approach. It most likely will try to recreate the plan construction used by the teacher search and learning this might be too challenging given that these will not necessarily include one simple approach to all situations but operate adaptively.

Elevator The Elevator domain illustrates transportation problems in which passengers have to be moved to specific floors in a building by using elevators. There are slow and fast elevators where the slower ones only move inside a block of floors and the faster ones skip floors. There is also limited capacity specified for each group of elevators and the cost involved with the elevator movements is different for fast and slow elevators.

All problem instances are taken from the sequential satisficing track from IPC 2008 and include two fast elevators. Finding optimal plans for Elevator problems is especially hard because the movement cost and schemes of slow and fast elevators varies. The acceleration

[1] The FF Domain collection is available at https://fai.cs.uni-saarland.de/hoffmann/ff-domains.html

of slow elevators is costly but therefore their movement is considerably cheaper for each floor compared to the fast elevators whose acceleration is quicker. For the evaluation, we use 30 problem instances which can be grouped into 3 levels of difficulty:

1. Problem *d-01* to *d-10*:
   These problems include 8 floors split into two blocks where one slow elevator moves in floors 0 to 4 and the second slow elevator in floors 4 to 8. The fast elevators move 2 floors at a time and therefore only serve even floor numbers. Slow elevators only fit 2 passengers while faster ones can carry up to 3 passengers at a time.

2. Problem *d-11* to *d-20*:
   These problems include 16 floors split into two blocks where one slow elevator moves in floors 0 to 8 and the second from floor 8 up to 16. The fast elevators move 4 floors at a time and therefore only serve floors 0, 4, 8, 12 and 16. Slow elevators fit 3 passengers and the faster ones can carry up to 4 people at a time.

3. Problem *d-21* to *d-30*:
   These problems include 24 floors split into three blocks with one slow elevator each moving in blocks 0 to 8, 8 to 16 and 16 to 24. The fast elevators move 4 floors at a time. Slow elevators fit 4 people while the faster ones can carry up to 6 passengers at a time.

Learning for the Elevator problems seems very hard as there is no straight-forward policy which can easily be learned or is followed by any of the planners.

Floortile In the Floortile domain, robots have to paint specific patterns in a grid. They can move in all four directions but only paint tiles up- or downwards and can not stand on painted tiles. In the used problem instances, there are two robots which have to paint an alternating chess-like pattern in black and white. Initially, no tiles are coloured yet. The grid for problem *d-xy-z* has the size $(x + 1) \times y$ and all but the bottom row has to be coloured in the pattern (z is just a counter to distinguish multiple problems of equal size).

This particular configuration is interesting because it contains many dead-ends in which uncoloured tiles can not be reached any more from the top or bottom side. The robots have to start at the top side of the grid and only paint tiles above them. Whenever a robot paints a tile below him, the task becomes unsolvable.

All problem instances used in the evaluation are from the sequential optimal planning track at IPC 2014 and include grids of size 5×3 up to 7×5. Due to the dead-ends, we assume that this domain will be hard for most planners and especially the ASNet policy search because it only naively follows the network policy. Therefore, the search fails whenever the policy leads into a dead-end and we do not expect that the network policy will be able to entirely circumvent any dead-ends. Furthermore, this might limit the training speed significantly because the network exploration during the sampling will often end up in dead-ends and as a consequence only explore comparably few states.

Hanoi The Hanoi domain encodes problems of the Towers of Hanoi task where disks of varying size have to be stacked on pegs. The difficulty comes from the fact that all discs can only be placed on top of larger disks or on top of the pegs. Initially the tower consisting of all disks is stacked on one peg and has to be transferred to another base.

All 20 problem instances were generated with the FF-Domain generator. Problem *d-x* contains x disks and all instances contain three pegs as the foundation.

While there exist strategies that can be applied to arbitrary Hanoi problems and solve them, the domain is still not trivial for modern planners. This is primarily due to the minimum number of moves required to solve an Hanoi instance. The shortest existing solution for a Tower of Hanoi puzzle with n disks has $2^n - 1$ moves. Despite this length of plans, we suppose that ASNets might be able to learn a policy for Hanoi problems because its solutions are based on repetitive patterns which are applied for all disks. This potentially simplifies learning a policy.

ParcPrinter The ParcPrinter domain models printing tasks executed on a multi-engine printer which is capable of processing multiple jobs simultaneously. The printer uses several Image Marking Engines (IME) which can be limited to black-and-white printing or able to print in colour. During the processing of one sheet, various printer components need to be handled including a feeder, transporter, inverter, finisher and the IMEs themselves. In the end, the sheets have to be stacked in the correct order. ParcPrinter tasks are very complex due to the many components, which have to be handled, each with their own constraints. Additionally IMEs are capable of varying their speed which further complicates the scheduling.

We use 10 problem files of the 2008 sequential optimal IPC track. The problem difficulty is varied with the amount of images and sheets processed for the task. The ParcPrinter instance *d-x* includes x sheets and images which need to be printed on top of the corresponding sheets. The necessary colour is varied among the tasks.

Overall, ParcPrinter problems are very complex. Due to their variability and many components, we do not expect that ASNets are capable of learning policies which generalise well and therefore can be applied to problems outside the used training instances.

Sokoban Sokoban originally is a japanese computer game developed by Hiroyuki Imabayashi which was published in 1982. In the game, the player has to move objects on predefined goal locations in a given map by pushing them forward. The objects can only be pushed if the field behind is empty. Usually, the maps contain many walls blocking potential paths.

We use 30 problem instances from the sequential satisficing planning track of IPC 2008. These do not share a common difficulty measure but are still ordered in a meaningful way representing the expected complexity. E.g. instance *d-10* only includes a single box which needs to be moved but the map is significantly larger than for most problems and is arranged like a maze.

Sokoban does not include many different actions or patterns like e.g. the ParcPrinter domain. However, the problem instances can be constructed in various ways and as a consequence demand different strategies. We expect that ASNets are able to solve some larger problem instances after training which require a shared approach but they will probably not be able to generalise well on problem instances demanding other strategies.

TurnAndOpen The TurnAndOpen domain is closely related to the Gripper problems previously used by Sam Toyer for his evaluation of ASNets in classical planning. It contains robots with two gripper arms each which have to carry balls across rooms. In contrast to the Gripper domain, the rooms are connected by doors which need to be opened first. Opening a door requires two free hands to turn the doorknob and opening the door simultaneously.

We use 19 problem instances of IPC 2014. However, we needed to slightly alter the domain and problem files as these were built for the temporal satisficing planning track. We removed temporal conditions and replaced duration measures for actions with costs. The problem instances contained between two robots, eight rooms and ten balls up to five robots, fourteen rooms and sixty balls.

Generally, we consider this domain to be fairly easy. The Gripper domain is very simple and the addition of doors which need to be opened does not seem complex. Also, one can exploit the same strategy on all TurnAndOpen problems independently of the number of robots, rooms or balls. Therefore, we expect that ASNets are capable of learning policies and generalise well on this domain.

Tyreworld The Tyreworld domain describes problems in which flat tyres have to be replaced by new ones. In order to do so one has to get all necessary tools, remove the nuts and assembly the new tyre.

We generated problem instances of size one up to twenty with the respective FF-Domain generator. The problem size for Tyreworld instances represents the number of tyres, which need to be replaced.

Problems of this domain can be considered simple because the same steps are used for each tyre. Therefore, the network is only required to learn this process for a single tyre and then should be able to perform well on further instances.

Domain	number of evaluation problems	number of training problems	expected difficulty
Blocksworld	35	3	hard
Elevator	30	1	hard
Floortile	20	1	mediocre - hard
Hanoi	20	3	mediocre
ParcPrinter	10	4	hard
Sokoban	30	2	simple - mediocre
TurnAndOpen	19	3	simple
Tyreworld	20	2	simple

Table 7.1: Overview over all domains and our expectation regarding the difficulty

7.2.2 Baseline Planners

In order to conclusively evaluate the performance of ASNets, we will compare it to competitive baseline planners for classical planning. These are all based on heuristic search being the dominant approach in this planning field. The baseline planners are implemented in the Fast-Downward planning system and their search time for each problem during the evaluation will be limited at 30 minutes.

A^* with h^{LM-cut}, h^{add} The A^* heuristic search maintains a prioritized open list containing states ordered by their *f-value* which is the sum of the cost $g(s)$ to reach the state and the heuristic value $h(s)$. Initially, the open list only contains the initial state. In each following iteration, the state with the lowest *f-value* is expanded. If it is a goal, the search is terminating and found a solution. Otherwise all states reachable from actions applicable in the state are collected and added to the open list with their respective *f-value*. However, if a state was already found previously it is not added again to the open list. Should the open list be empty at the beginning of an iteration then the task is unsolvable. This search algorithm is almost exclusively used in optimal planning because it guarantees optimal plans whenever used with an admissible heuristic [23]. Due to its generally expensive computation, it is rarely seen in non-optimal planning.

In our evaluation, we use this search with the admissible LM-cut heuristic h^{LM-cut} introduced by Helmert and Domshlak in 2009 [26] as well as the additive heuristic h^{add} by Bonet and Geffner [7]. Note, that h^{add} is not admissible and therefore A* with this heuristic will not necessarily provide optimal solutions.

Greedy best-first search with $\mathbf{h^{FF}}$ Greedy best-first search (GBFS) is the satisficing twin of A* proposed by Russell and Norvig [47]. The algorithms are closely related as they share the same structure with the only difference being the values used for open list ordering. While A* uses the more informative *f-values*, GBFS solely relies on the heuristic values. Despite this seemingly minor difference, the algorithms share almost no use-cases because GBFS looses the optimality guarantee from A* due to not considering the g cost values. However, it is significantly faster for most domains whenever using an informative heuristic. Therefore, it is a popular heuristic search approach for satisficing planning.

We use greedy best-first search in our evaluation with the relaxed plan heuristic h^{FF}[27] using a dual-queue with preferred operators.

LAMA-2011 The LAMA-2011 planning system [44] was one of the winners at the 2011 IPC. It combines various approaches from the previous search engines. First, the LAMA planner conducts greedy best-first searches combining the relaxed plan heuristic with landmark heuristics [43] and also using a dual-queue of preferred operators. The goal of this search is to quickly find a plan. Afterwards multiple iterations of (weighted) A* using the same heuristics and pruning strategies are run repeatedly until a guaranteed optimal plan is found or the time limit is reached. Weighted A* in contrast to "pure" A* multiplies the heuristic values on states with the given weight before adding the costs. For weights > 1 weighted A* does not guarantee optimality (even for admissible heuristics) but serves as a middle ground between the fast greedy best-first search and typical A* providing optimal solution quality. Therefore, the LAMA planner executes multiple iterations with the weighted search and steadily decreases the weight until pure A* is used. This approach allows the LAMA planner to find reasonable plans very quickly and steadily improve the plan quality until an optimal solution is found or the time limit for the search is reached.

7.2.3 ASNet Configurations

For all empirical evaluations we will use the same ASNet configuration with a hidden representation size $d_h = 16$ and two layers, i.e. three action layers and two proposition layers, already used by Sam Toyer. This also includes the ELU activation function. To minimize the risk of overfitting, we apply L2-regularization to all modules in intermediate layers with $\lambda = 0.001$ as well as dropout with a probability of $p = 0.25$.

During the training of our ASNets, we use up to $T_{max-epochs} = 10$ epochs with each executing $T_{prob-epochs} = 3$ problem epochs. Alternatively, we also terminate whenever the early stopping criteria described in Section 5.1.3 is met or the soft time limit of two hours is exceeded. This time limitation is checked at the beginning of each problem epoch, but we do not interrupt currently running sampling searches or training steps. Similarly, the network creation for each problem evaluated was limited at one hour, so that if the process took over an hour for one problem instance the following larger problems would not be processed.

It is worth noting, that we accumulate our samples for one problem in each epoch, i.e. we use the sampled data for every of the three sequential problem epochs.

For the training, we apply $T_{train-epochs} = 100$ training epochs in each training call for which we use the Adam optimizer with a learning rate of $\alpha = 0.001$. As the loss function we use our proposed binary crossentropy loss outlined and explained in Section 5.2.

As teacher searches we use three different search configurations. As an optimal teacher search, we use A^* with h^{LM-cut} while for satisficing planning A^* with h^{add} and greedy best-first search with h^{FF} and a dual-queue of preferred operators are used.

7.2.4 Metrics

We want to be able to compare the ASNet performance with the baseline planners. Therefore, we provide detailed metrics which can be analysed. For each evaluated domain, we will provide the coverage indicating how many of the problems in the domain were solved by each planner. Additionally, we will track the cost of plans and search time for every solved problem. In the case of the network evaluation, we will also include the time spent to create the model used in the following search.

In the training process of ASNets for each domain, we will measure the development of the loss throughout the training and provide the time spent in each phase of the training cycle, so how much time was spent during model creation, sampling and training epochs. Lastly, we will observe the success rate of the network search in the sampling process during training indicating the percentage of solved problems to check whether the ASNets were already able to perform well early and if this translates to good performance in the evaluation thereafter. Notice however, that the success rate indicates how many problems were solved during the sampling, which happens immediately after the training for the given problem. Therefore, the success rate does not necessarily reflect the percentage of training problems the network would solve after the entire epoch but will usually overestimate the network performance.

The entire set of tables including these metrics can be found in the appendix.

7.3 Results

In this section, we will outline the performance of all baseline and ASNet planners for the evaluated domains and provide an analysis regarding visible or assumed shortcomings wherever possible. For all domains, we will first look at the data already collected during training followed up with the performance on all problem instances. The entire data set in more detail regarding the training can be found in the appendix A, while the evaluation tables can be found in appendix B.

7.3.1 Domains

Blocksworld The training data for the Blocksworld domain implies convincing performance given that all ten epochs were finished after about 45 minutes for each network configuration. A stable success rate, i.e. percentage of problem instances solved by the network exploration during the sampling, of 70 to 80 percent was reached for all training configurations after the third epoch at latest. It is also worth noting that the predominant time consumption during the entire training process were the training epochs taking considerably more time than the sampling search or the network constructions.

Despite this positive data during training, all ASNets were only able to generalise to some extent. The networks solved all instances already used during training but only few problems outside the training set. The configurations trained with an A* teacher performed better in terms of plan quality, which is to be expected, and number of solved problems. All baseline planners solved significantly more problem instances in mostly shorter amount of time.

Domain	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Coverage	28/35	35/35	35/35	35/35	7/35	7/35	4/35

Table 7.2: Coverage for evaluated Blocksworld Domain

For most solved problem instances, the cost of plans found by the ASNet policies were identical to the ones of their respective teacher searches. In some cases, the plan quality was worse but rarely even better. For problem instance *d-13-1* the best plan was found by the ASNet trained with the optimal A* h^{LM-cut} teacher search. The search itself did not terminate and even the LAMA planner was seemingly unable to reach an optimal solution in the given time.

However, no common property among problem instances solved by the networks could be found. As expected, the approaches needed to solve Blocksworld tasks are too diverse. Therefore, the networks are unable to learn a single policy which performs well on the majority of instances in this domain.

Elevator Already the training for the Elevator domain was entirely unsuccessful. Only up to three epochs could be finished in the given two hours but not a single problem could be solved during training. It seems that our expectation was correct that instances of this domain would be too complicated and diverse to learn a single policy which is capable of solving problems. Therefore, the ASNets were all unable to solve any problems during the evaluation.

Domain	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Coverage	2/30	15/30	30/30	30/30	0/30	0/30	0/30

Table 7.3: Coverage for evaluated Elevator Domain

The trajectories of the network exploration confirm our observation. All networks only did small progress during the problem epochs on one problem trying to follow the teacher search solution which was insufficient to solve even the single problem used during training.

Floortile The Floortile domain was considerably more difficult for all planners. While the A* h^{add} baseline planner solved all twenty problem instances, the other planners were incapable of solving even half of the problem instances with the given time limitation.

This performance of the teacher searches clearly translated to the network policies. Only a single epoch of training could be finished with the A* h^{LM-cut} and GBFS h^{FF} teacher searches due to the duration of the sampling search. While the ASNet using A* with the inadmissible h^{add} was able to finish four epochs, even this network policy was unable to generalise at all. It was only able to solve a single problem, which was not even the instance used during training.

Domain	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Coverage	6/20	20/20	9/20	9/20	0/20	1/20	0/20

Table 7.4: Coverage for evaluated Floortile Domain

The actions applied by the network policies during search with the corresponding trajectories imply that the inability to solve Floortile problems was not caused by reaching dead-ends, as we first expected. The networks were all able to learn to only paint tiles above them. However, it seems that the policy was indecisive regarding the movement of robots leading to almost identical probabilities for the move actions which as a consequence were often reverted with the invertible actions leading to an already explored state, so the search terminated. Therefore, it can be stated that the networks were able to learn some approach for this domain despite the performance of the teacher searches, but invertible actions seem to be an issue for ASNets with our current approach.

Hanoi Training for the Tower of Hanoi domain went comparably to Blocksworld. The most time during training was used for the training epochs with a stable success rate among all configurations finishing all epochs in just half an hour.

Domain	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Coverage	13/20	15/20	16/20	15/20	3/20	2/20	2/20

Table 7.5: Coverage for evaluated Hanoi Domain

However, the networks did not generalise outside the training set at all. While the three instances used for training were almost all solved by the different ASNets, not one problem beyond was solved by the network policies. It seems that the networks do not recognize the repetitive pattern of Hanoi and are therefore only able to replicate the exact same actions seen during training. While the same strategy can be applied to all Hanoi tasks, the necessary action pattern increases in size for growing problems despite its structure remaining. This minor gradual change in the action patterns seems to be too complicated for ASNets to learn. As a consequence, the network is too uncertain regarding the actions needed and often undoes its steps ending in already explored states which terminates the search early.

ParcPrinter Training performed on the ParcPrinter domain seemed successful. While not all ten epochs could be finished in two hours, the success rate was stable above 70 percent from the second epoch onwards and neither network construction nor sampling required major time.

ASNets trained with A* teacher searches performed very poorly in the evaluation. Only the smallest problem instance was solved, which is disappointing given that four problem instances were already frequently solved during training. The network search trained with GBFS and the FF-heuristic was able to solve two out of four ParcPrinter tasks used for training and two larger instances with the exact same plan quality as its teacher search. The baseline planners all outperformed even the best ASNet search.

Domain	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Coverage	6/10	8/10	10/10	10/10	1/10	1/10	4/10

Table 7.6: Coverage for evaluated ParcPrinter Domain

Usually, one would assume that the improved performance of the ASNet trained with the significantly faster GBFS from the satisficing planning branch was caused by its more efficient sampling and training. However, all training problems were solved by all three teacher searches in under 0.02s, so while GBFS is faster, the search time did not seem to be the limiting factor for any ASNet configuration.

When looking at the trajectories during search, it can be observed that whenever the network search failed, the policy printed an image on the wrong sheet leading to a dead-end. The network was simply not confident enough in its decision regarding the printing of images on corresponding sheets. Still, the networks were able to learn large parts of the strategy needed to solve ParcPrinter tasks. They performed the scheduling of the processes involving the ordering of sheets and selecting the correct colour-schemes almost perfectly. It seems that the uncertainty regarding the printing of images and corresponding sheets in the network policies led to the disappointing performance during evaluation.

Sokoban The Sokoban domain was entirely unsuccessful for ASNets. While the sampling search did not demand an overwhelming amount of time as for the Floortile domain, not a single problem was solved during up to six epochs across all configurations. It seems, they were entirely unable to learn any policy at all and therefore it is no surprise that they also did not solve any instances during the evaluation.

Domain	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Coverage	28/30	29/30	29/30	29/30	0/30	0/30	0/30

Table 7.7: Coverage for evaluated Sokoban Domain

The actions applied by the policy indicate a similar behaviour and issue as for the Floortile domain. In many situations the network had to choose between multiple movement actions which were all assigned almost identical probabilities. Additionally, after applying one move action the network frequently moves back again ending in an already explored state and terminating the search.

This might have several reasons. It could be that ASNets are incapable of reliably deciding on one path when multiple of equal quality exist. But on some problem instances it seems like only a single path towards any progress exists and is still not chosen or at least followed until the mentioned progress towards the goal. Another reasons might be the limited receptive field of ASNets as we only used networks with three action and two proposition layers, so they might not be able to recognize the upcoming progress as it is too far away still.

TurnAndOpen The TurnAndOpent domain was arguably the worst performing for ASNets. Due to the inability of the A* h^{LM-cut} search to solve even the smallest problem in the 30 minutes time limitation, it is not surprising that the ASNet trained with such a teacher search was incapable of progressing. We terminated the training after the first sampling search took already over nine hours. While we finished the training for the other two configurations, it did not look considerably better either, only finishing a single epoch without any successful network explorations. The single epoch took almost three hours for the GBFS trained ASNet and nearly four hours for the A* h^{add} teacher network.

Domain	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Coverage	0/19	17/19	15/19	19/19	0/19	0/19	0/19

Table 7.8: Coverage for evaluated TurnAndOpen Domain

However, the applied actions show a similar weakness as in the Floortile or Sokoban domain. The networks successfully learned to open doors first and carrying the balls to their destinations afterwards but in the end failed when they applied an inverted action going back to the previous state.

Tyreworld Training on the Tyreworld domain was interesting in multiple ways. First, it is the only domain in which training stopped due to the early stopping criteria, i.e. the success rate was 100% from the second or third epoch until termination. Secondly, a significant difference in time distribution for all configurations can be observed. While training with the GBFS h^{FF} teacher search terminated after about 20 minutes with most time spend in training epochs, the sampling search with the A* h^{LM-cut} teacher took 45 minutes itself across all epochs and was the predominant time during training. The ASNet using the h^{add} teacher search lies in between.

Domain	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Coverage	3/20	6/20	20/20	20/20	20/20	0/20	20/20

Table 7.9: Coverage for evaluated Tyreworld Domain

The impressive performance during training also translated to the evaluation for ASNets trained with the GBFS and LM-cut teacher search. These ASNet policies were able to solve every problem of this domain significantly outperforming A* with LM-cut itself which only terminated for the first three out of twenty problem instances. The plan quality was identical to GBFS h^{FF} and the LAMA planner. While GBFS was considerably faster than the ASNet, the network search terminated earlier than the LAMA planner which did not finish all its iterations.

However, the ASNet trained with the h^{add} teacher search, which performed almost identical during training, was incapable of solving any problem instance including the ones already solved during training. After inspecting the trajectories of this ASNet policy, it appears that it was comparable to the other two but with minor confidence. The network policy failed due to applying repetitive actions at the very end of the plan when putting away the used tools. Hence, the search reached a state already explored and terminated without finding a solution. It is unclear, why the policy was able to apply the correct actions during the training process and failed thereafter.

7.3.2 Interpretation

For the interpretation of our evaluation and the assessment, we will answer the initial questions posed in Section 7.1.

1. Are ASNets able to learn good and potentially even optimal policies with respect to the teacher search? ASNets do learn close to optimal policies in almost all cases with respect to their teacher search if a solution has been found at all. Sometimes, they even improved upon their (non-optimal) teacher performance. While this is the case, the network search was unable to solve the majority of evaluated problem instances across the domains. The Tyreworld domain was the only exception for which ASNets performed very well (for two configurations). Besides, the network search was able to generalise to some extent on the Blocksworld domain as well as for the ParcPrinter domain for the networks trained with the GBFS h^{FF} teacher. Outside of these domains, the networks failed to generalise at all, sometimes they did not even solve the problems used during training.

2. On which domains do ASNets perform well? Looking at the evaluation result, ASNets only performed well on the Tyreworld domain. However, the whole answer is not as simple and therefore we will look at all domains grouped by performance during training and evaluation in more detail.

The training already indicated poor performance in the case of the Elevator, Floortile, Sokoban and TurnAndOpen domains. In the case of Sokoban and Elevator, the networks were incapable of solving any problems during the sampling search despite finishing multiple epochs. For the Floortile domain, the teacher searches A^* with LM-cut and GBFS with FF were too expensive on even the smallest instance used for training with the same problem occurring with the LM-cut teacher on the TurnAndOpen domain. Therefore, this training could hardly be successfull. While the A^* h^{add} teacher was significantly faster for Floortile and the network solved the training problem over the epochs during training, the ASNet was still unable to solve any problems during the evaluation including the instance used for training.

The actions applied by the network policy during evaluation for Floortile, Sokoban and TurnAndOpen problems indicate a similar difficulty. Unlike previously expected, dead-ends in the Floortile domain were not the main issue. The network was indecisive regarding movement for all those domains, frequently reverting its own actions with the inverted action leading to an already explored state. Such behaviour has likely multiple reasons. It can be assumed that the receptive field and length of action chains possibly considered by the network due to its limited depth is one problem. Another is the fact that move actions are often applied on a larger field, so there are multiple paths towards to the goal which can be chosen interchangeably. It would be logical for the network to assign the actions leading to any of the paths with (almost) equal probabilities. These can lead to indecisiveness of ASNets and as a consequence the risk of inverting its own action arises.

It appears that these situations closely relate to the concept of symmetry which is an already established problem in automated planning with multiple approaches proposed by research during the last two decades [20, 16]. There exist multiple pruning techniques to remove symmetric parts of the state space. These could be applied in a more sophisticated search based on ASNet policies which might improve the performance in domains as

Floortile, Sokoban and TurnAndOpen.

The limitation of the receptive field of ASNets can partly be overcome by additional input features, as already successfully applied by Sam Toyer in his work.

For the domains Blocksworld, Hanoi and ParcPrinter the ASNets were able to perform well during training but mostly did not generalise to problems beyond the training set. It seems that solutions for Blocksworld problems are too diverse, so that the networks are incapable of finding a single policy with the help of the teacher search which is able to solve all tasks. One could try to teach ASNets a simple strategy where first all blocks are unstacked and thereafter stacked to the towers as described in the goal. It would be interesting to observe the learning and generalisation ability for such a case, but none of the used teacher searches make use of a comparable, repetitive approach. A similar problem occurs for the Hanoi domain. While generally speaking the same strategy is applied to all Hanoi problems, it involves a chain of actions growing for larger instances. Despite the remaining pattern, the networks are not able to recognize and learn the repetitive structure and therefore do not generalise at all.

The ParcPrinter domain is very different. The task itself is very complex and challenges the baseline planners, due to its large amount of different action schemas and patterns which have to be processed. However, it is important to note that these various situations, which have to be dealt with when solving a ParcPrinter instance, can always be solved with the exact same action sequence after considering the properties like e.g. using colour when printing instead of black and white. We originally assumed that the variety of these tasks would be too challenging for ASNets. It turns out, that they were able to learn the scheduling almost perfectly. The only prominent issue, preventing the network policies from solving further problems, was its indecisiveness whenever printing the correct image on the corresponding sheet. In this situation, the network frequently chose the wrong image. It is unclear, why the network is capable of choosing the correct patterns beforehand but not in this last step to fulfil goal propositions.

Lastly, the Tyreworld domain should be looked at as the positive example of a domain, ASNets are not just successful during training but generalise well on. One major reason is certainly the repetitious pattern for changing one tyre which identically translates to all further tyres. While this also leads to many symmetries, as the tyres can be handled in an arbitrary order, these do not seem problematic because the network does not necessarily need to decide on one path and strictly follow it. There are merely various subproblems which have to be solved and are mostly independent from each other. E.g. at the beginning it does not matter in which order the tools are picked as long as all are picked before the network starts working on the tyres and the tyres can all be replaced independently. This can also be seen in the ASNet policy probabilities. These are almost identical for all tyres at the beginning and the network as a consequence arbitrarily chooses one but as this decision is not of significance for solving the task, any indecisiveness is not harming the network performance.

However, even for the Tyreworld domain ASNets are not perfect. While the networks are able to find good solutions for all problem instances, the search still takes significantly longer than the GBFS with the FF-heuristic. Additionally, the main time spent to be able to evaluate the network for these problems is used to build the ASNet models. One shortcoming of the current implementation of ASNets lies in the model creation and processing. ASNets contain one module for each grounding by definition which can lead to enormous networks expensive to create and do any computations with. This limits the training and evaluation alike due to memory and time consumption. It is still important

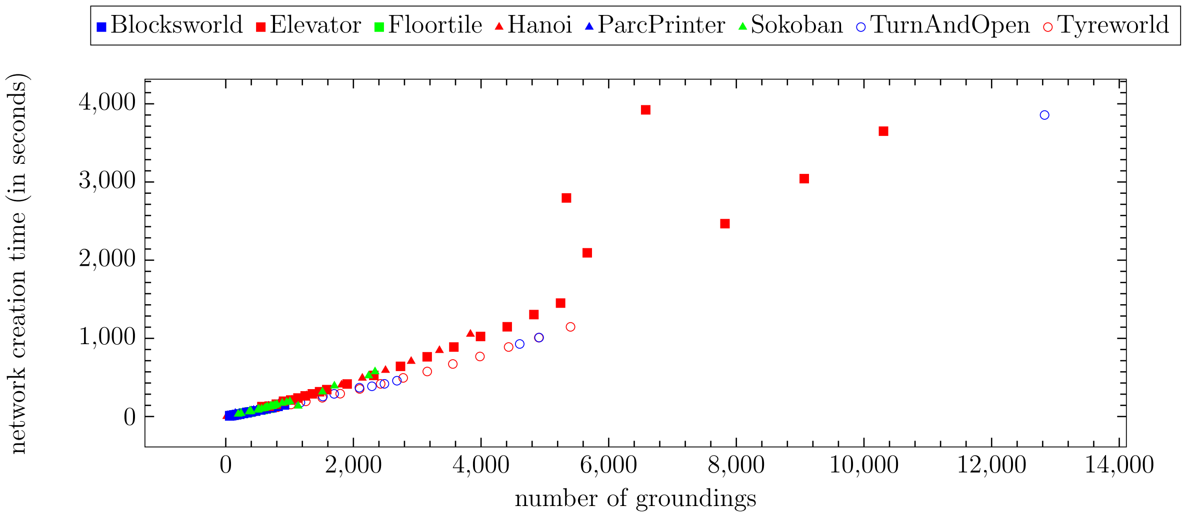


Figure 7.1: Network creation time with respect to the number of groundings

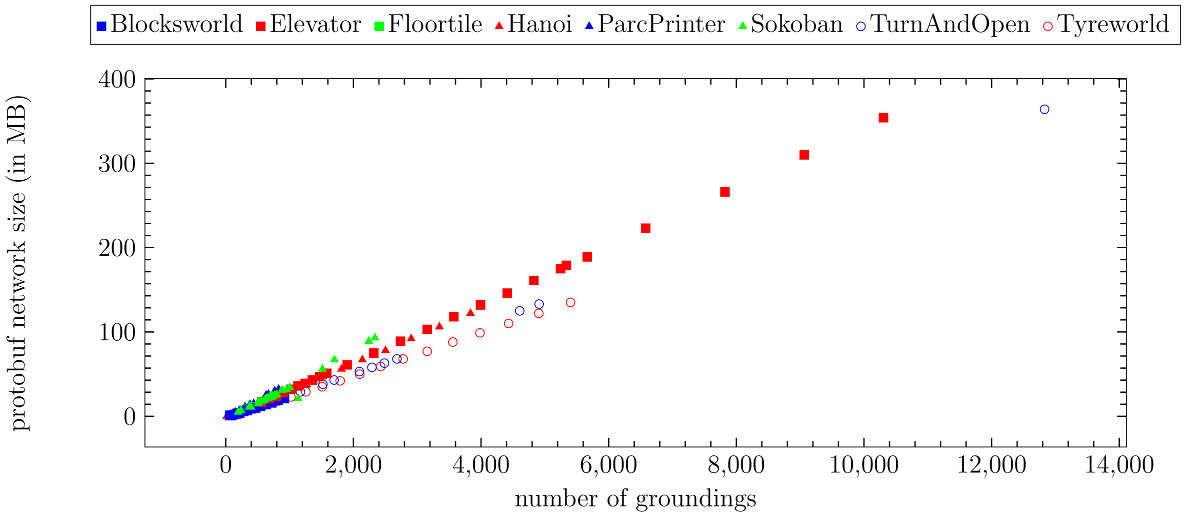


Figure 7.2: *Protobuf* network size with respect to the number of groundings

to note, that this limitation does not necessarily lead to poor training or generalisation, because the Tyreworld problems contain many groundings and ASNets performed best on this domain, but it is an issue with respect to scaling which should be addressed.

Figures 7.1 and 7.2 illustrate the scaling of the *Protobuf* network file size as well as the time needed to construct the models. It can be seen that most problems in the domains contain up to thousand groundings but there are a few domains like Tyreworld, Elevator and TurnAndOpen whose inherent structure leads to an immense increase of groundings for larger problem instances. It should be emphasized that these networks need to be constructed for each single problem before the network policy can be exploited during search. For domains with a large amount of groundings, this can take considerably more time than the training process for the entire domain which is certainly suboptimal.

3. For which period of time do we need to train ASNets until they perform reasonably well? The necessary training time is obviously heavily dependent on the domain, but it can be observed that only few epochs seem to be sufficient in most cases. When looking at the success rate during training, no further progress can be recognized after two or three epochs.

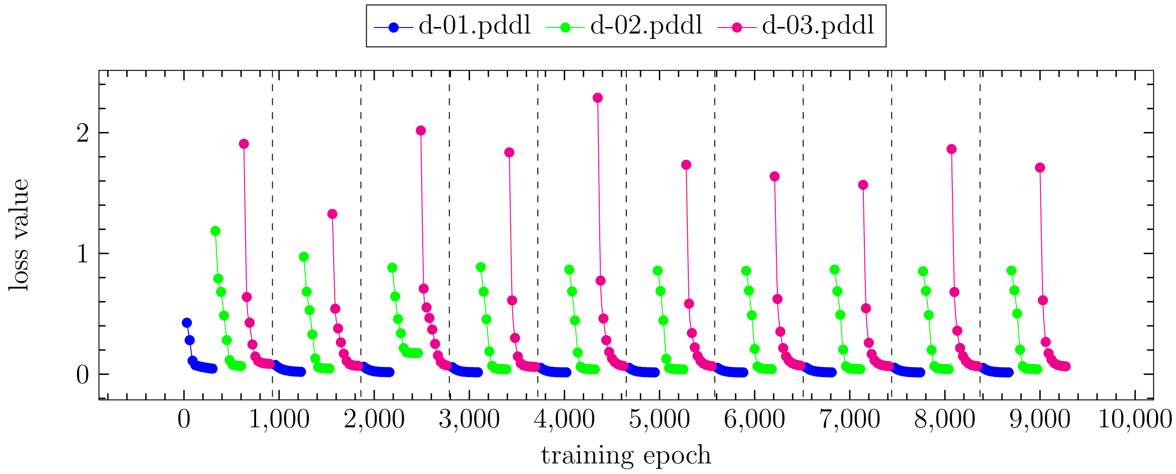


Figure 7.3: Loss development during training for the Hanoi domain with A* h^{add} teacher

Furthermore, one can analyse the development of the loss values during training. It can be found that these are very volatile for ASNets in our training algorithm. Most progress made for one problem is set back significantly after applying few problem epochs on a second problem instance. This behaviour can be observed for all domains. Figure 7.3 illustrates the loss development for the Hanoi domain with the A* h^{add} teacher search. Even after using only a single problem epoch, such a graph could still be observed. These values imply that our networks overfit and are too specialised on a given task. Therefore, whenever going to the next problem we will almost restart at the beginning which makes steady progress over multiple epochs seemingly impossible. One major reason for this limited generalisation of training on single problems is probably the sampling process in which the predominant amount of states are sampled using the teacher search. This causes the network to learn to replicate the search of the used teacher which usually does not translate well to other problems. However, this does not seem to be the only reason for the inability of ASNets to learn or generalise outside the training set in most domains because such a development can also be found for Tyreworld as shown in Figures 7.4 and 7.5.

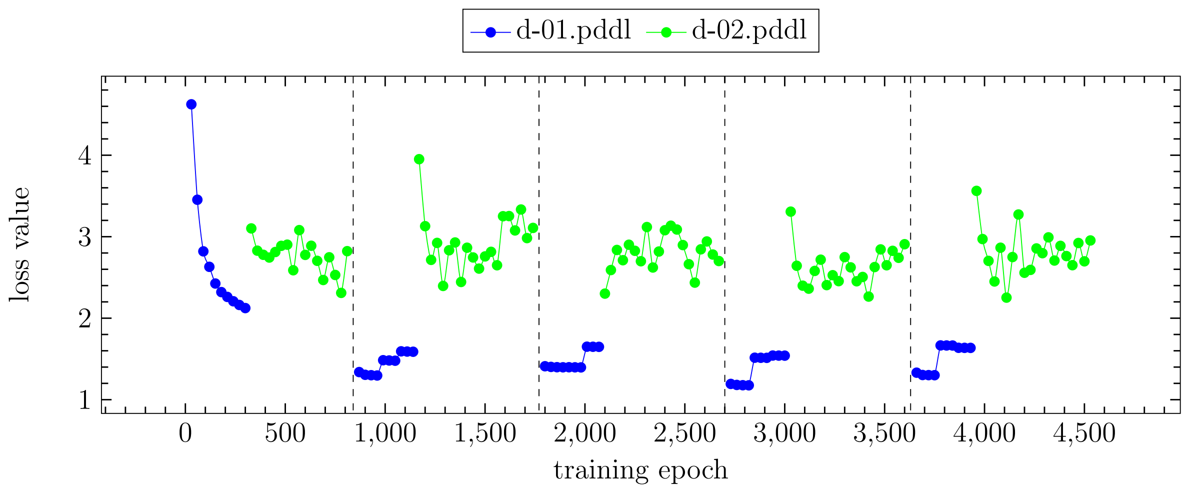


Figure 7.4: Tyreworld loss development with A* h^{LM-cut} teacher

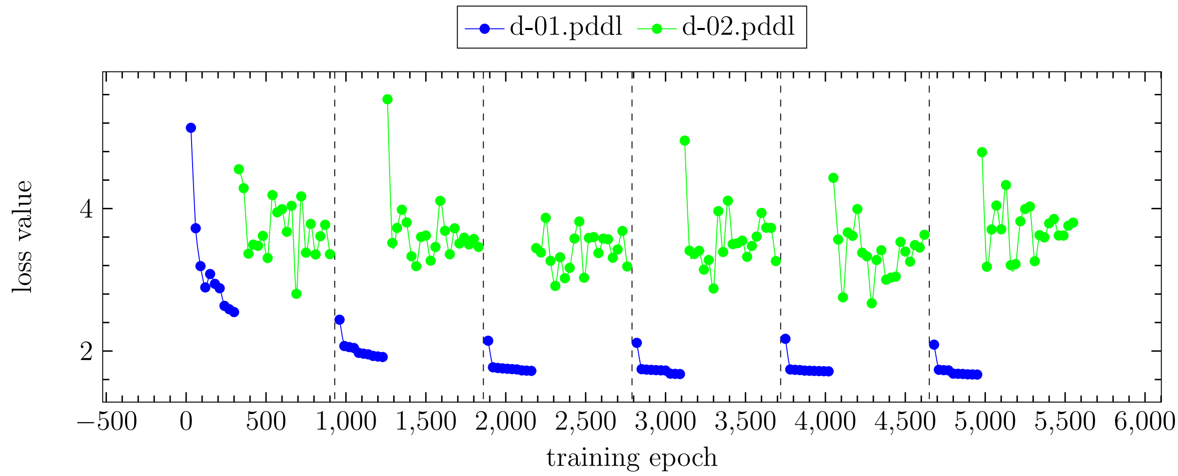


Figure 7.5: Tyreworld loss development with A* h^{add} teacher

The graphs even look almost identical for both A* teacher search trainings on the Tyreworld domain, despite the LM-cut trained ASNets were able to solve all problems and the h^{add} ones not a single instance.

Loss development graphs, as well as illustrations of the success rate and time distribution during training for all domains can be found in the Appendix A.

Chapter 8

Future Work

This thesis serves as a starting point to learning domain-dependent policies with neural networks for application in classical planning mainly based on the work of Sam Toyer with his introduction of Action Schema Networks. However, there are still logical extensions and related approaches worth considering regarding future research in this field.

8.1 Additional input features

One straight-forward extension, which should be implemented for classical planning, are additional heuristic input features as they were already proposed and evaluated by Sam Toyer. While our work considered such inputs and already provides the framework for such additions, they were not implemented yet.

Sam Toyer found, that binary values indicating landmark features computed by the LM-cut heuristic function were able to assist ASNets in overcoming potential limits regarding their receptive field. However, we propose to use non-binary values. While such simple input data can be helpful, neural networks are usually capable of processing and learning based on more complex inputs carrying more information. This way, one could not only encode simple boolean input values but e.g. provide the exact number of disjunctive action landmarks in which an action occurs and the amount of disjunctive action landmarks only containing one specific action.

Besides heuristic features as additional inputs, one could also provide action costs for all actions in the input. Currently, the network can not directly reason about action costs which are only passively recognized due to the teacher search considering it for the sampling and the corresponding $y_{s,a}$ values. While the evaluation shows, that with such sampling the network is capable of learning cost-optimal solutions (whenever an optimal teacher search is provided), such inputs might speed up the learning and lead to less dependency on the teacher search for good network policies.

8.2 Lifting implementation limits

Currently, our implementation of ASNets for classical planning is still posing some limitations regarding their use. As already indicated by Sam Toyer, the networks can only be defined for planning tasks not containing any quantifications. The reason is fairly simple. Whenever quantifiers occur in action conditions or effects, it is not guaranteed anymore that every grounded action instantiated from an underlying action schema has the same number of related propositions. This variation would make the current form of weight sharing for action modules impossible. However, this limitation can be overcome fairly easily by introducing pooling to action modules just as it is already done for propositions.

Another remaining difficulty lies in the ASNet architecture which include one module for each grounded action and proposition. As seen in the evaluation (and visible in the appendix data tables), this leads to huge networks in some domains which do pose significant challenges in memory, processing time and their learning capabilities. Due to the seemingly inefficient computation on those domains, training and sampling does not progress as intended or can not be finished at all. Closely related to this issue, it would mean considerable progress in usability if ASNets could not only share the weights but the entire network among all problems of a given domain. This would save significant time in which we had to construct the networks whenever processing a problem. But the definition of relatedness, as one of the core concepts of ASNets, is dependent on the instantiated problems of a domain, so it remains unclear how to overcome this challenge.

8.3 Sampling strategies

Our current sampling strategy used during training is a slightly altered version of the original approach proposed by Sam Toyer modified for classical planning application. During this sampling search we collect sample data in form of states with additional information based on (goal) trajectories followed by our network policy or the teacher search. While the data collected by this process is often essential to solve the problem at hand, it will only represent a small part of the state space. This can be potentially problematic given that states sampled from connected trajectories share a strong correlation and dependency. Such a connection can lead to a strong bias of the trained network policy which might limit its "understanding" of the problem as a whole and therefore its adaptability and ability to generalise. As we train our networks to follow the already biased trajectories from the teacher search, we increasingly push the policy towards merely imitating the teacher instead of learning concepts for the entire domain from it. This might be one of the reasons why stable progress during training with respect to the loss development was not achieved.

Such a bias could be diminished by not sampling from (goal) trajectories of the teacher search for each explored state. These are highly repetitive for most states explored leading to states along the teacher trajectory being sampled several times which only enforces the network to imitate its teacher. One could only sample the teacher trajectory once from the initial state just as it is done with the network search for exploration. This will lead to significantly smaller sampling data sets and it has to be analysed whether it is still sufficient for progress. However, this might reduce the bias of the ASNet policy towards the teacher search without major changes.

Another approach to avoid such a bias entirely would be to use a uniform sampling strategy, i.e. collecting uncorrelated data randomly. This means, no teacher search to collect connected trajectories could be used due to their dependency. Sampling truly random states without an underlying bias from a planning problem state space is very challenging, but it could improve the quality of the sampling data and therefore the learning considerably.

8.4 Integrate ASNets in search

One logical addition directly based on this work would be the implementation of sophisticated search engines for policies in general or our network policy in particular. The current policy search, which was used with the ASNet policies in our evaluation, naively follows the most probable action in a state according to the given policy. While such a simple approach allowed us to truly evaluate the network policies without any assistance, it does not seem very promising for further planning applications.

Search with backtracking The already existing policy search could be extended with a backtracking algorithm or an open list containing already found states sorted by their policy probability. This would allow the search to continue whenever a duplicate or dead-end is explored by e.g. simply choosing the second best action in the last state according to the policy instead of choosing the most probable one. Despite its simplicity, such an addition could lead to significant improvements in domains containing many dead-ends or circular paths in their state space.

Combine ASNets with heuristics Another approach for searching with a network policy would be to combine already established heuristic functions and pruning techniques with the policy. The action probabilities of the policy could e.g. be used for tiebreaking between states with equal heuristic values. Another idea would be to combine heuristic values with the action probabilities. When expanding state s with $s \xrightarrow{a} s'$ during the search, a value based on the action probability $\pi^\theta(a \mid s)$, the heuristic value $h(s')$ and potentially the cost $g(s')$ could be computed and used as the priority measure. E.g. one could prioritize states which minimize the value $h(s')/\pi^\theta(a \mid s)$ which increases for a rising heuristic value or a decreasing policy probability.

Furthermore, as already implied in the evaluation, pruning techniques capable of removing symmetric paths, which can be chosen interchangeably, could assist the network policy. Given the visible difficulty during the evaluation regarding indecisiveness of ASNet policies in these situations, this could considerably improve the search performance.

8.5 Recurrent approach

Lastly, instead of going the apparent way and finding sophisticated search engines to apply the network policy in, it might also be worth considering alternative network architecture approaches. Currently, the network is often incapable of adapting its behaviour if a problem of the domain has a new property not encountered during training. One could state, that ASNets do not necessarily learn “intelligent” behaviour, as previously mentioned, but just strictly recollect a provided teacher search. Closely related, the network policy frequently guides the search towards already encountered states. Especially at the beginning of the training, this is a serious challenge in domains containing (many) circles. So far, this situation led to an early termination of the search because the policy would just retake the same path running in a circle due to the exact same inputs.

Therefore, it might enhance the networks generalisation capability to not just represent the current state of the problem as the input, but also include previous choices or already applied actions. With such additional information, a network could potentially learn to adapt its behaviour if it recognizes repetitions in its trajectory.

Such structures are already successfully applied to language processing and machine translation in the form of recurrent neural networks [40]. These networks receive results of previous computations or iterations as inputs and are therefore able to constantly process new input data while also considering previous steps.

Chapter 9

Conclusion

The objective of this thesis was to evaluate the suitability of domain-dependent policy learning with Action Schema Networks for application in classical automated planning. Therefore, we integrated this novel neural network architecture into the Fast-Downward planning system as the primary framework for heuristic search in automated planning. To do so, we first extended the PDDL translation process of Fast-Downward to compute the relations between abstract action schemas and predicates as well as among their groundings, which are essential for the network structure of ASNets. We implemented the neural networks using custom layers of the *Keras* machine learning library.

Following, we modified the original training algorithm of Sam Toyer for application in classical automated planning. This primarily involved the implementation of a sampling search to collect states from the state spaces of planning tasks which can be used as data sets during training. Instead of using a teacher policy as proposed by Sam Toyer for predominantly probabilistic planning, our sampling allows for an arbitrary search configuration implemented in the Fast-Downward system to be used as a teacher search. This leads to large flexibility given the popularity of the framework and the many planning configurations already implemented in Fast-Downward including search algorithms, heuristic functions and various pruning techniques.

To reliably represent ASNets in Fast-Downward we extended the system with policies as a second form of evaluation besides heuristics. These were based upon already existing concepts in the framework and simplify the addition of any further policies in Fast-Downward. The network policy itself serves as an interface to the ASNet model, which is contained in a *Protobuf* network file, feeding any input into the network and later extracting the policy. As the search algorithms previously implemented in Fast-Downward are all based on heuristic functions, we added a simple policy search to exploit these new evaluators on planning tasks.

Finally, we conducted an extensive empirical evaluation of ASNets to answer our main question whether ASNets are suited for application in classical planning. We considered eight domains of varying complexity and structure and compared the networks after training with multiple teachers to the performance of competitive baseline planners from the classical planning field. Although the networks only generalised well and performed impressively on a single domain, significant learning could be found for most tasks. Therefore, we do not consider ASNets unsuitable for classical planning but rather found shortcomings of the approaches currently followed regarding the network architecture, its training and sampling process. We provide analysis for the training progress and policy behaviour during evaluation for each domain in order to find common issues which prevented the network search to be successful on these domains. Based on the findings of our examination, we provide suggestions for further research which might alleviate or even solve the identified problems.

However, the final assessment regarding the suitability of Action Schema Networks for classical planning will depend on the results of further research building upon our work.

Appendix A

Training

For the training process of each domain, we provide graphs indicating

- the **time distribution**, i.e. how much time was spent for network creation, sampling and training epochs respectively
- the **success rate** development
- the **loss** development

Note, that we do not provide such information for the ASNet trained with the A* h^{LM-cut} teacher for the TurnAndOpen domain as the training was interrupted after nine hours without finishing a single sampling process.

A.1 Blocksworld

1st configuration: A* h^{LM-cut} teacher

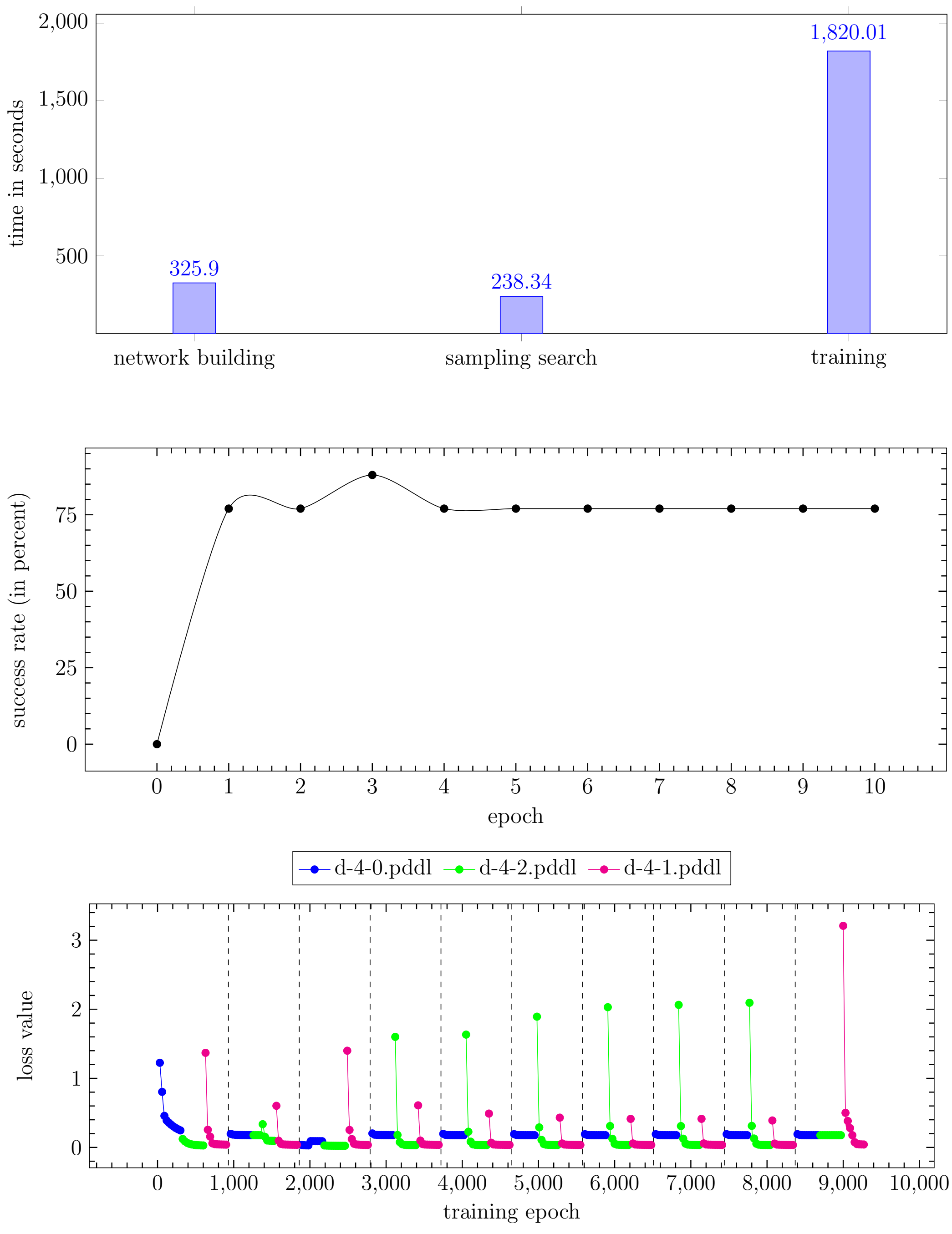


Figure A.1: Time distribution, success rate and loss development during training

2nd configuration: A* h^{add} teacher

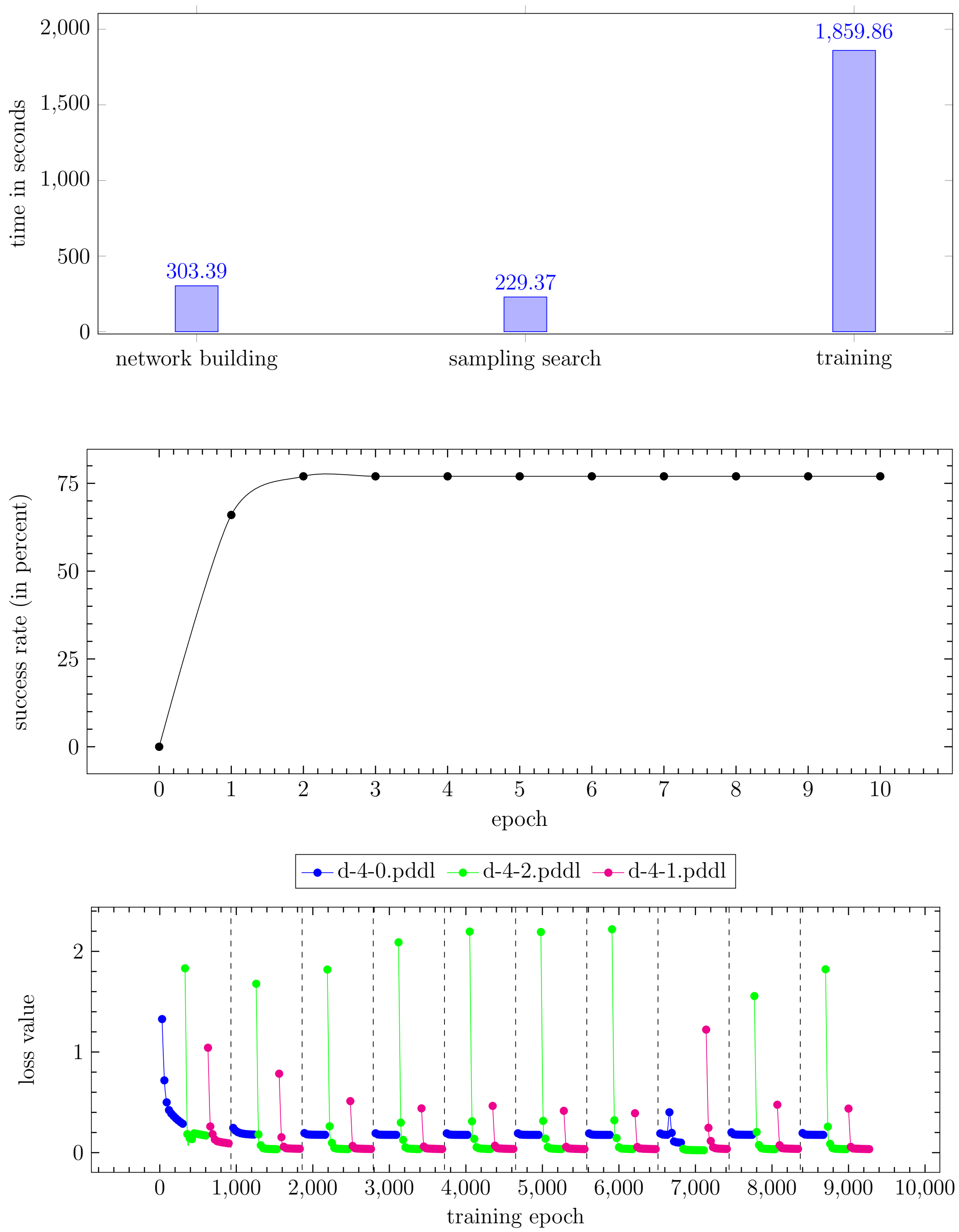


Figure A.2: Time distribution, success rate and loss development during training

3rd configuration: GBFS h^{FF} teacher

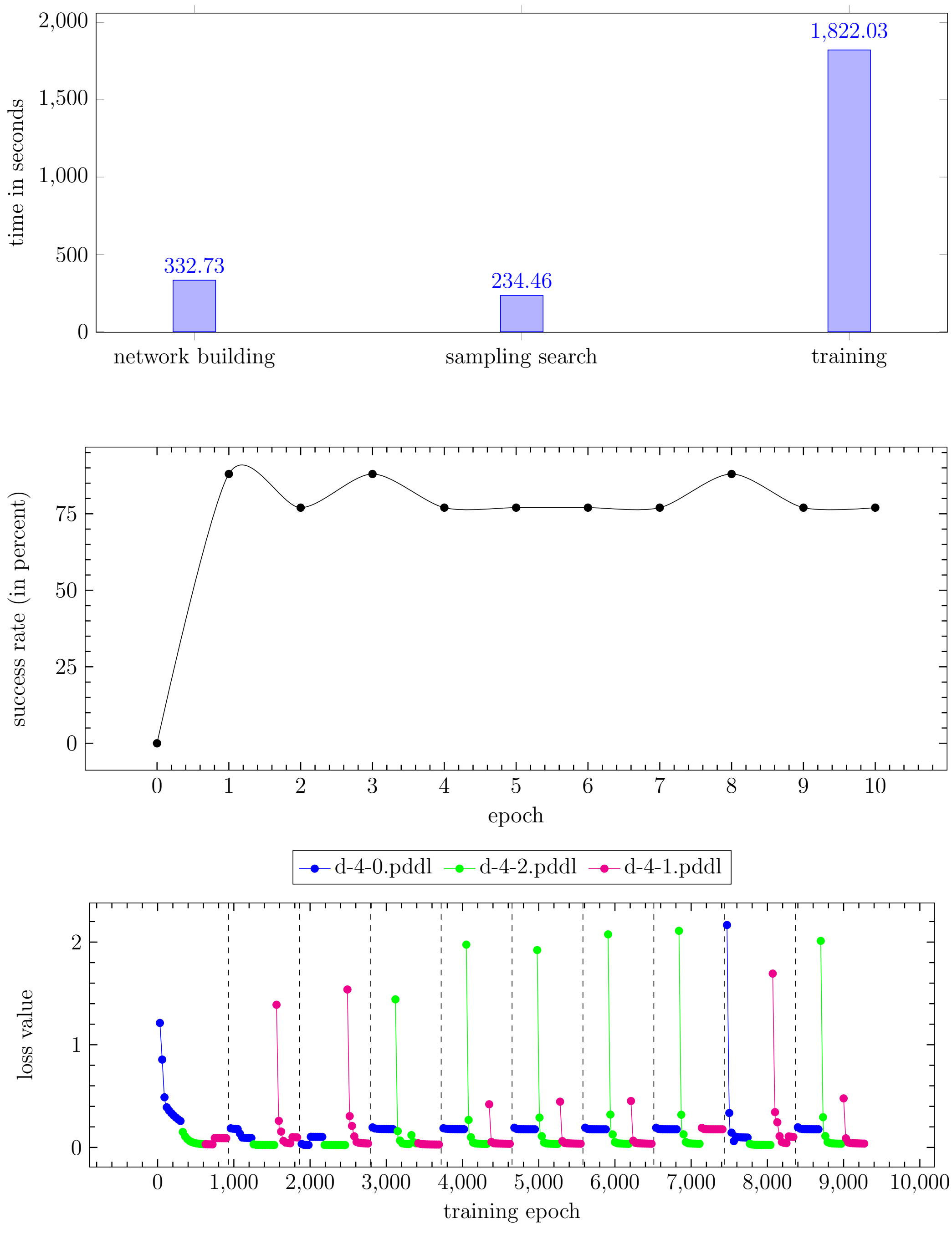


Figure A.3: Time distribution, success rate and loss development during training

A.2 Elevator

1st configuration: A* h^{LM-cut} teacher

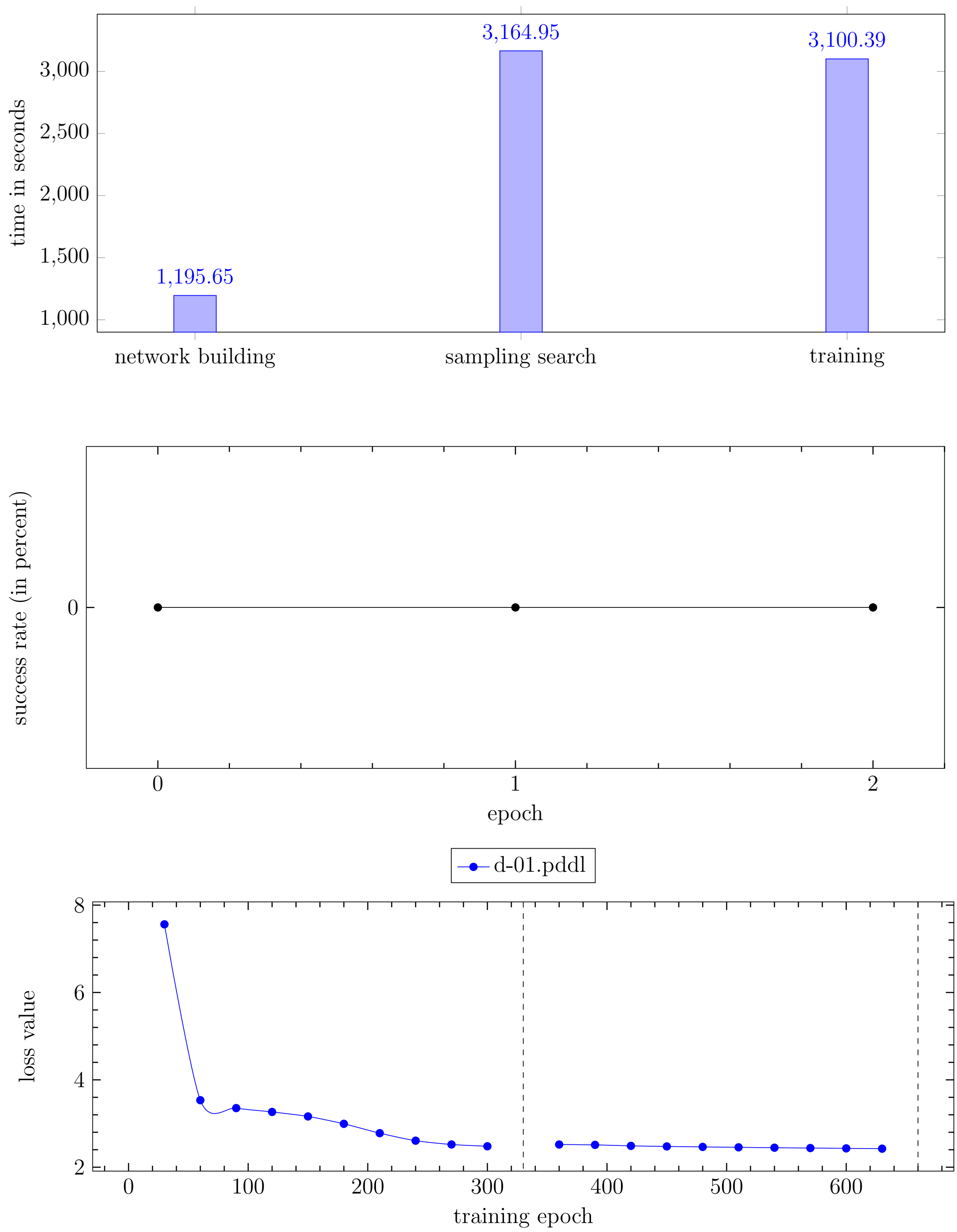


Figure A.4: Time distribution, success rate and loss development during training

2nd configuration: A* h^{add} teacher

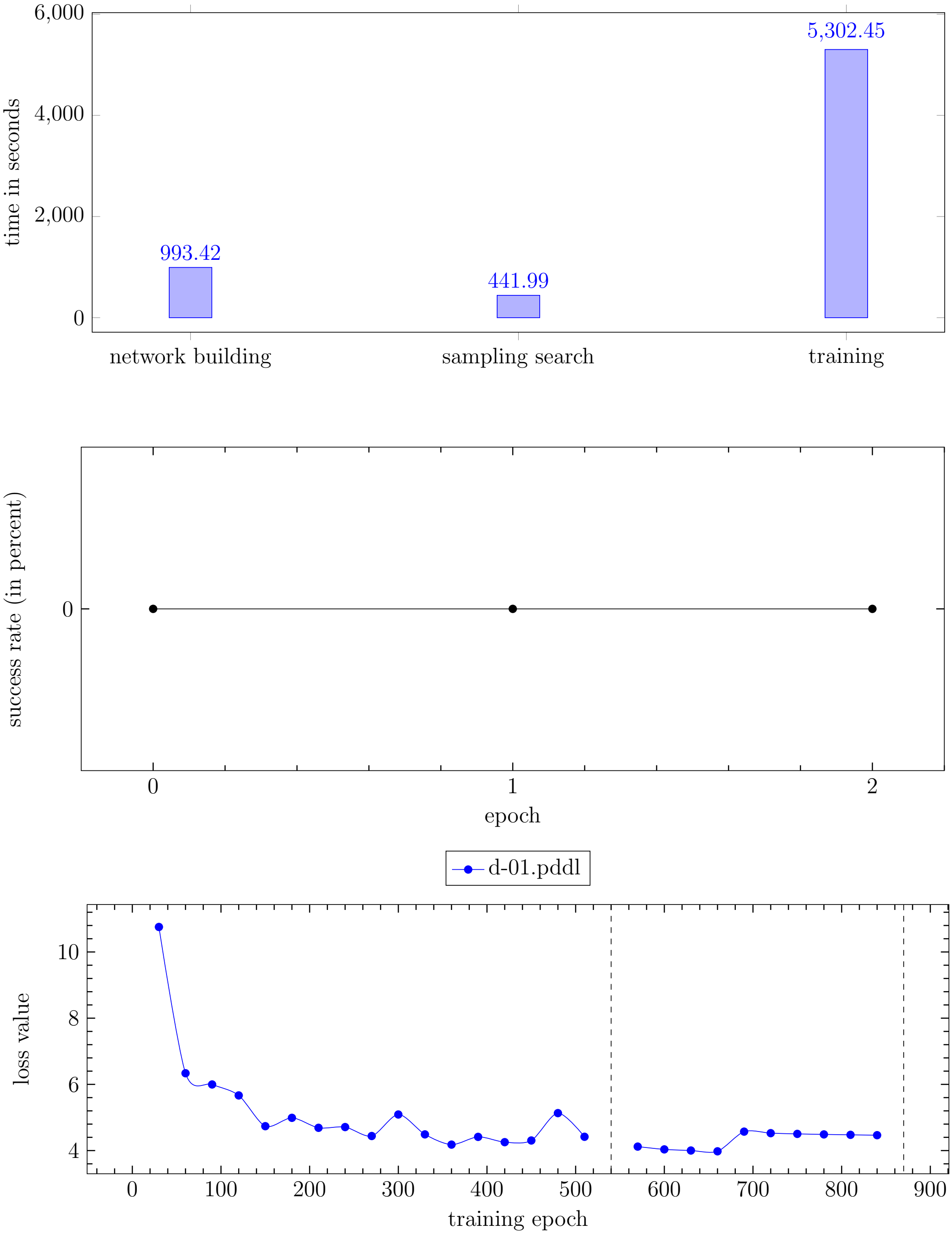


Figure A.5: Time distribution, success rate and loss development during training

3rd configuration: GBFS h^{FF} teacher

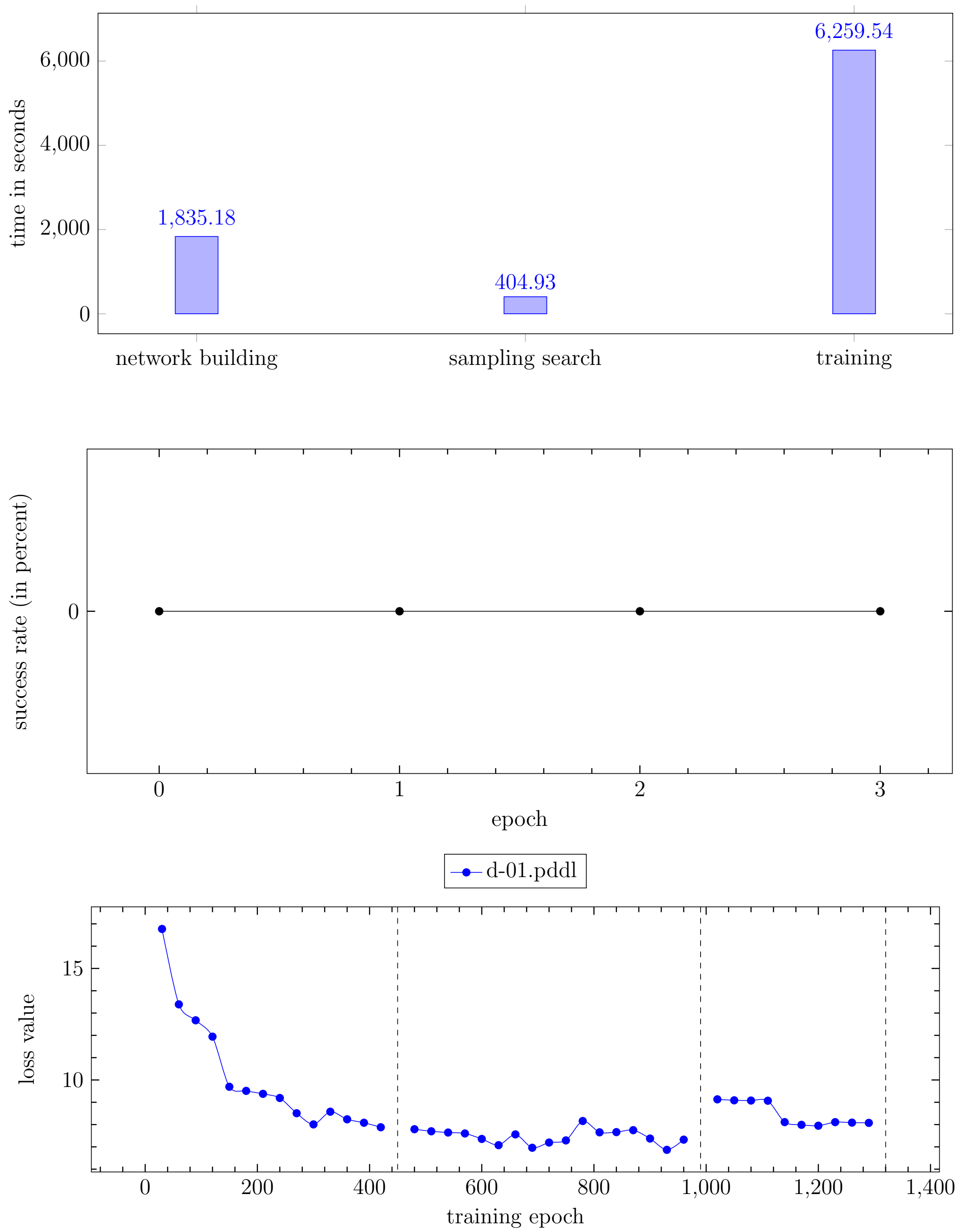


Figure A.6: Time distribution, success rate and loss development during training

A.3 Floortile

1st configuration: A* h^{LM-cut} teacher

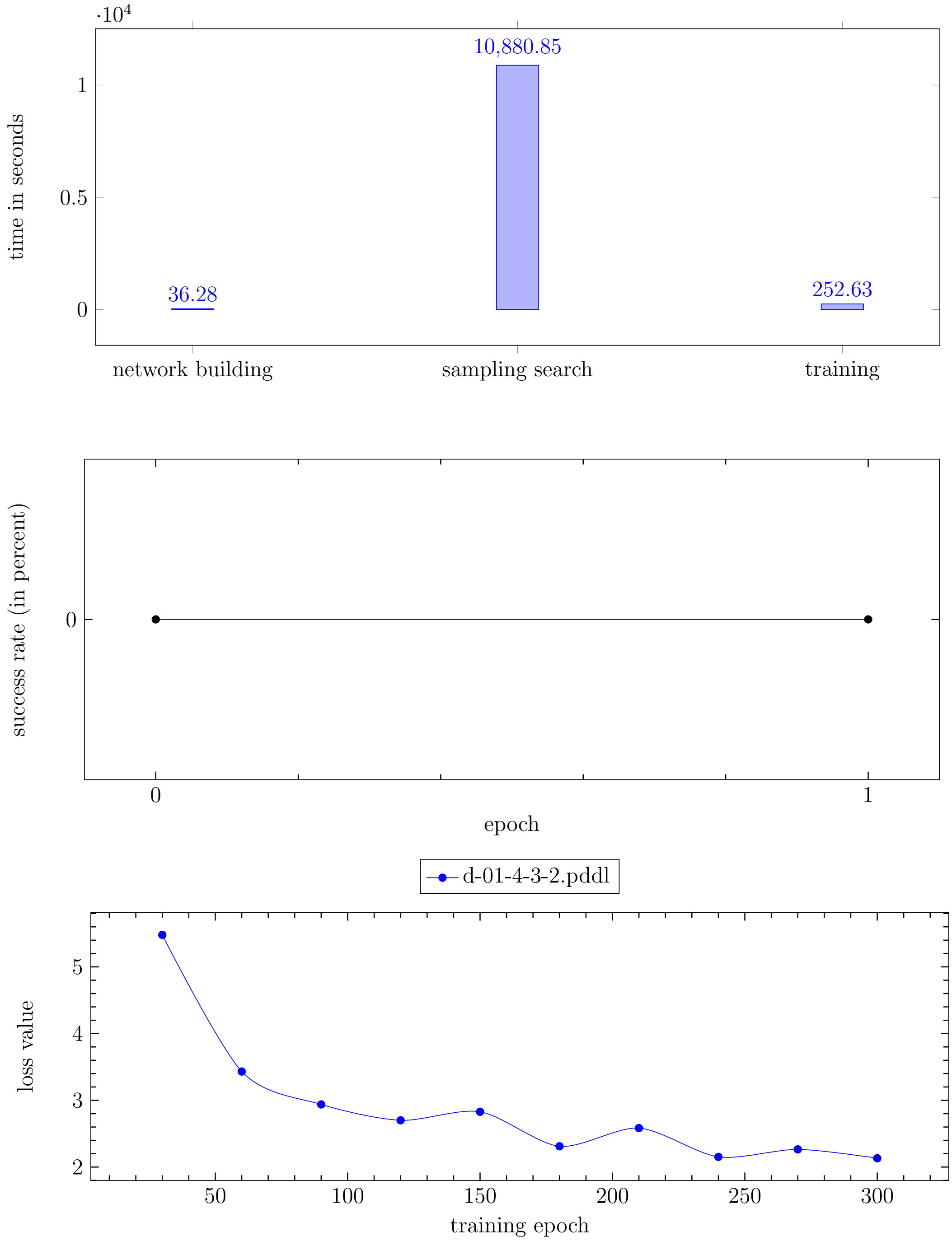
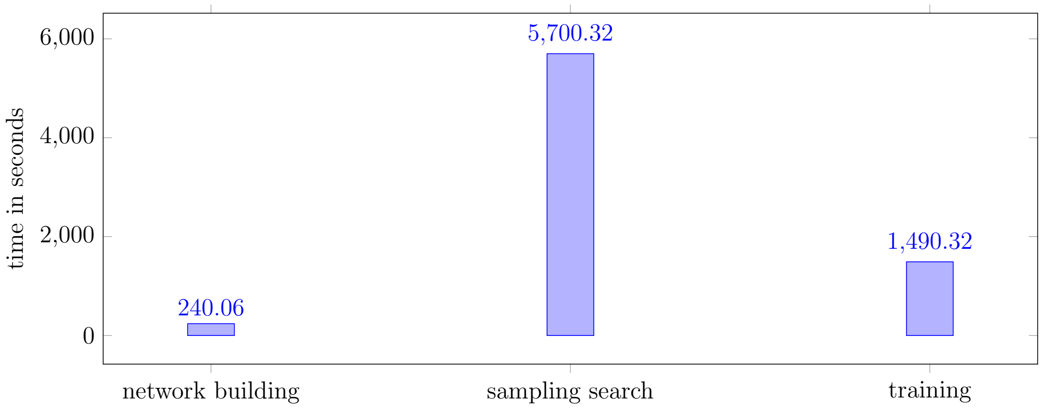
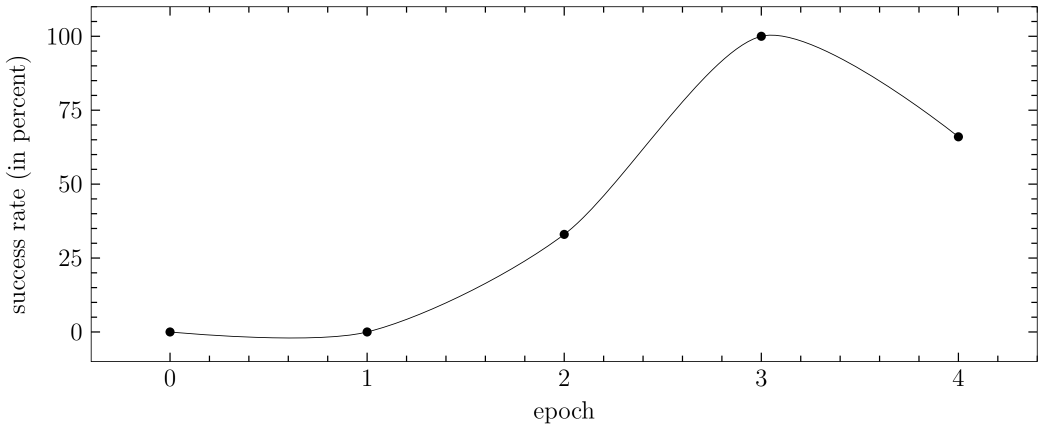


Figure A.7: Time distribution, success rate and loss development during training

2nd configuration: A* h^{add} teacher





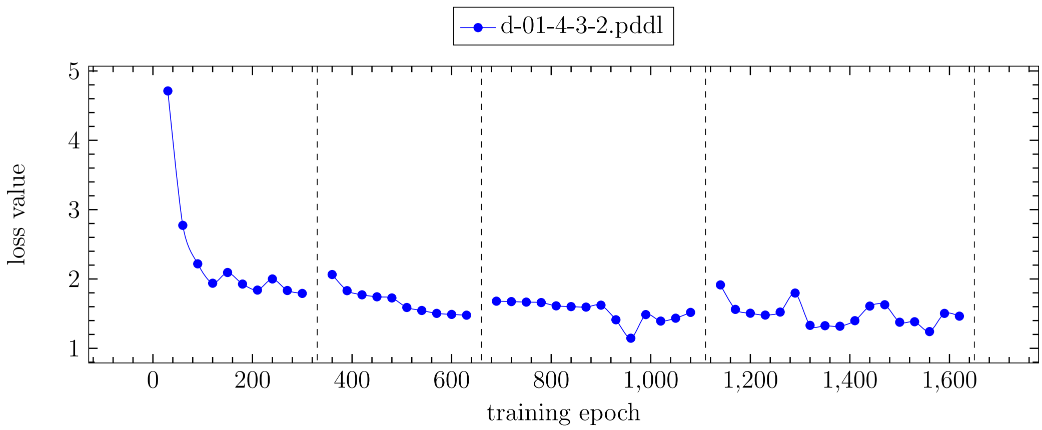


Figure A.8: Time distribution, success rate and loss development during training

3rd configuration: GBFS h^{FF} teacher

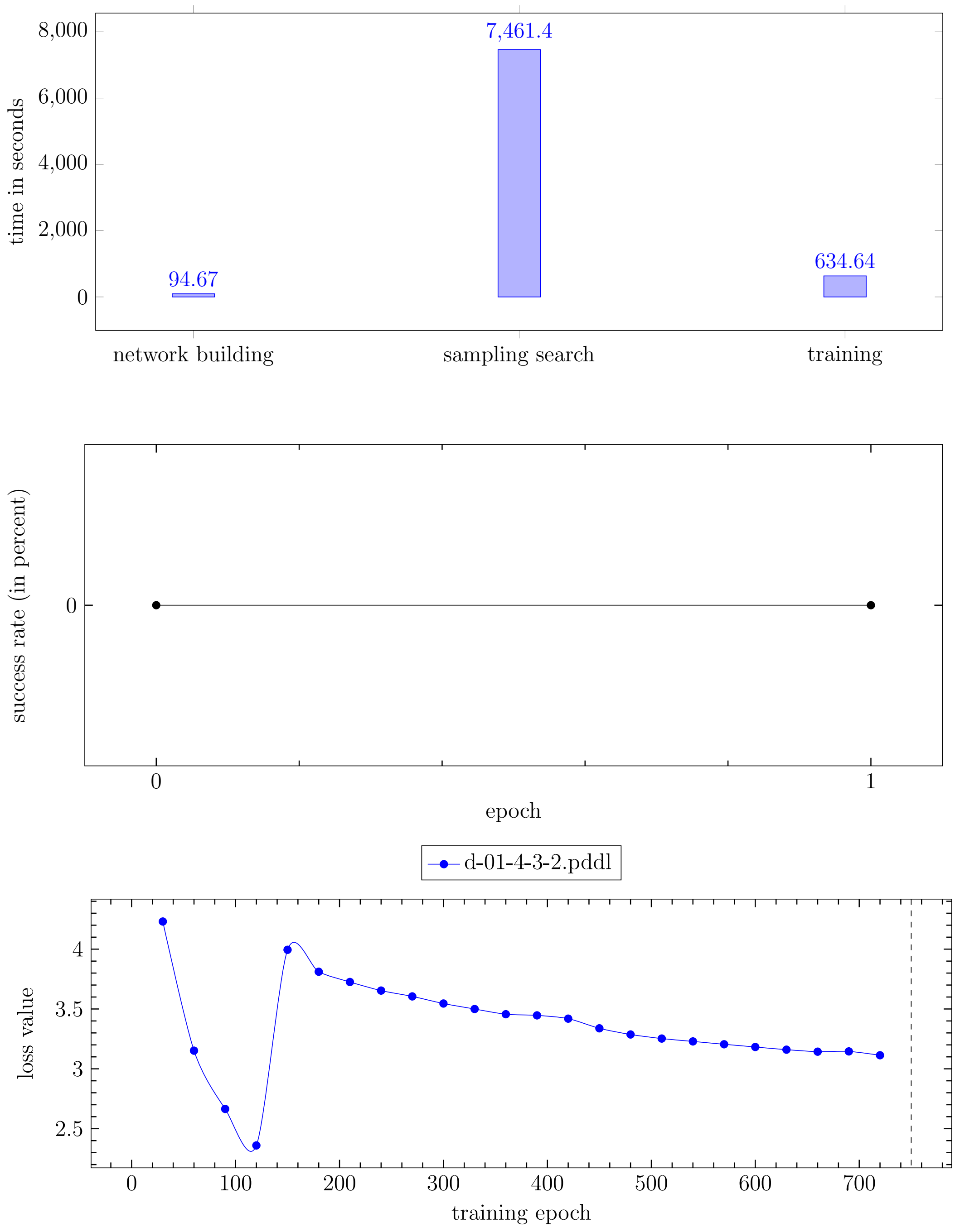


Figure A.9: Time distribution, success rate and loss development during training

A.4 Hanoi

1st configuration: A* h^{LM-cut} teacher

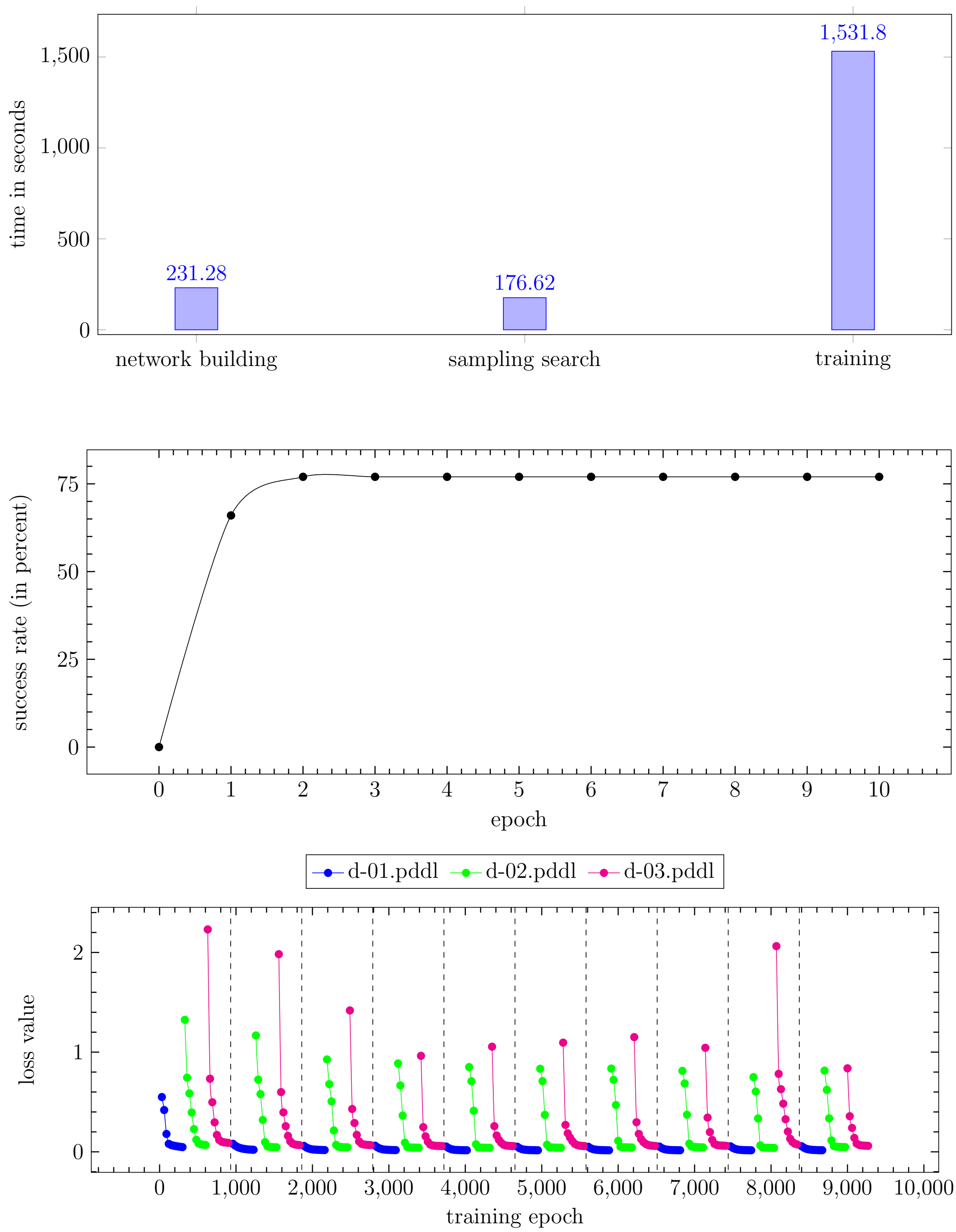


Figure A.10: Time distribution, success rate and loss development during training

2nd configuration: A* h^{add} teacher

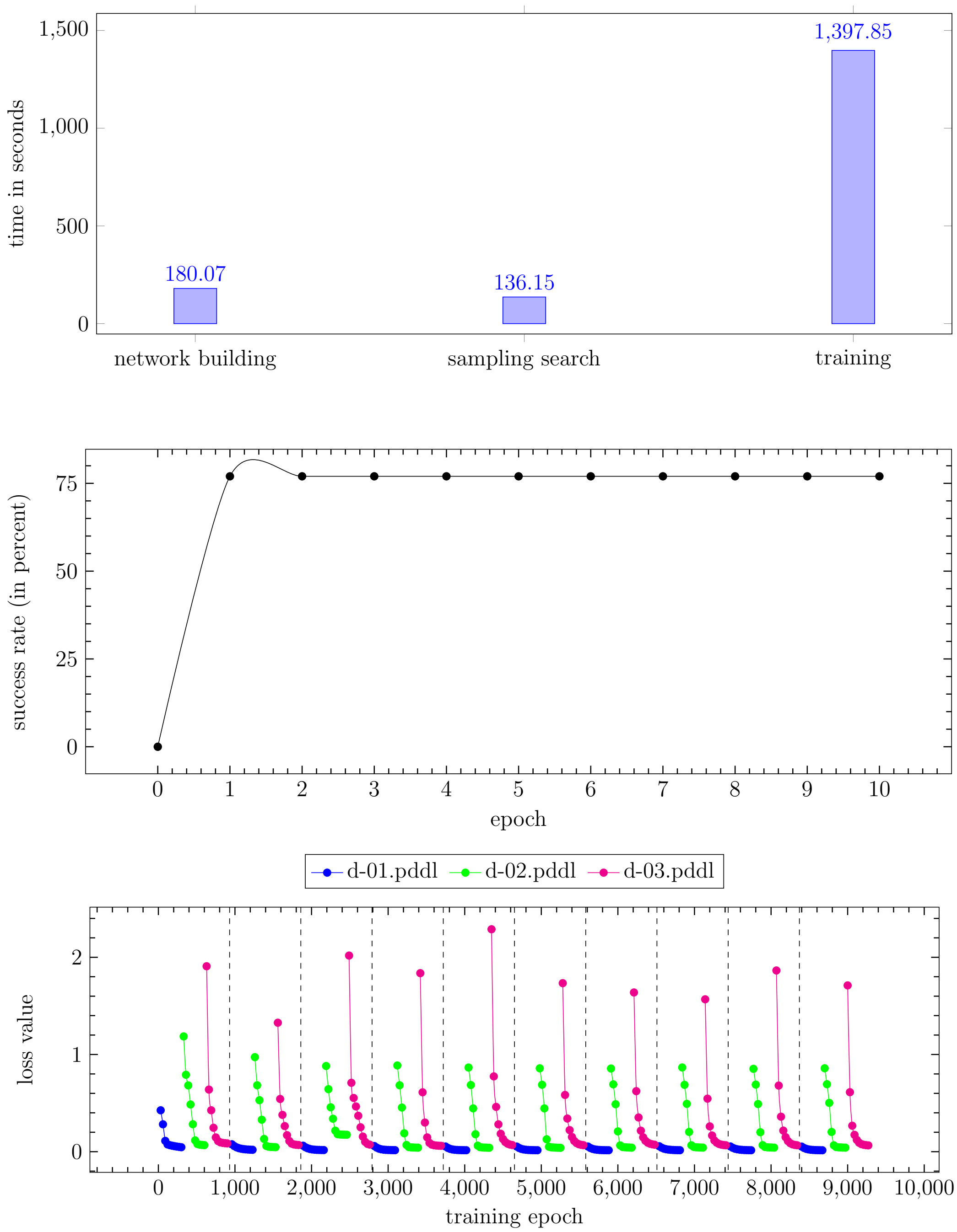


Figure A.11: Time distribution, success rate and loss development during training

3rd configuration: GBFS h^{FF} teacher

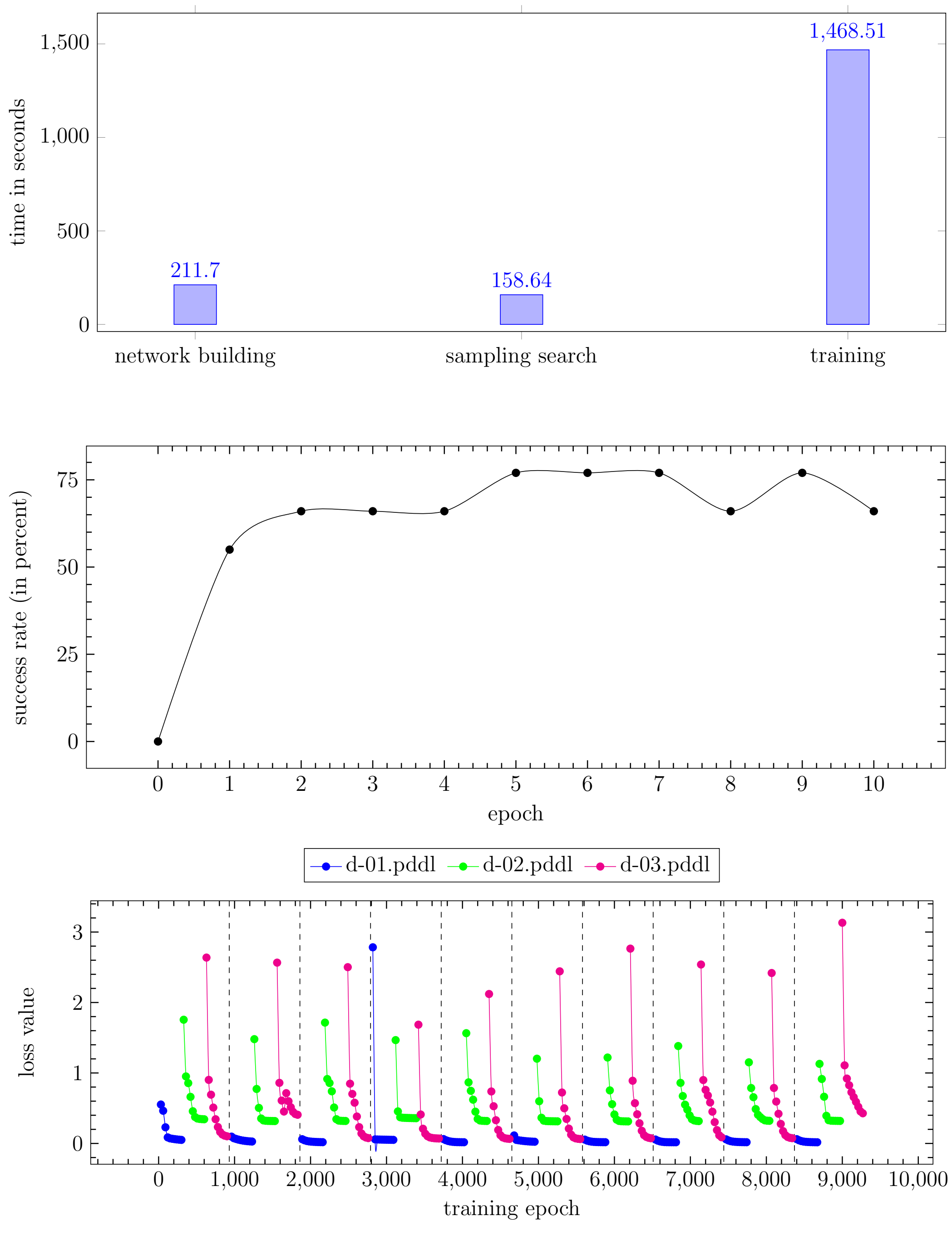


Figure A.12: Time distribution, success rate and loss development during training

A.5 ParcPrinter

1st configuration: A* h^{LM-cut} teacher

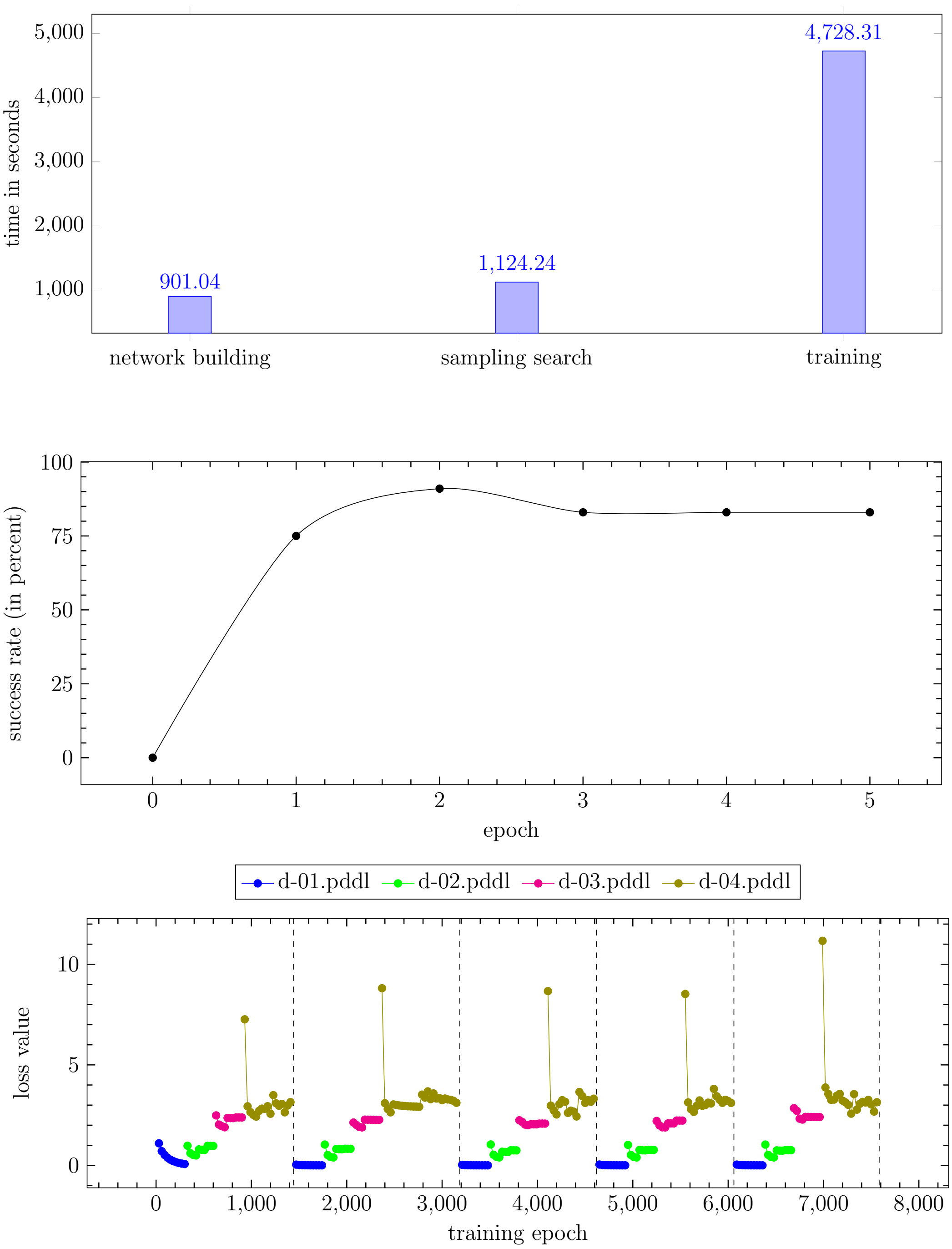


Figure A.13: Time distribution, success rate and loss development during training

2nd configuration: A* h^{add} teacher

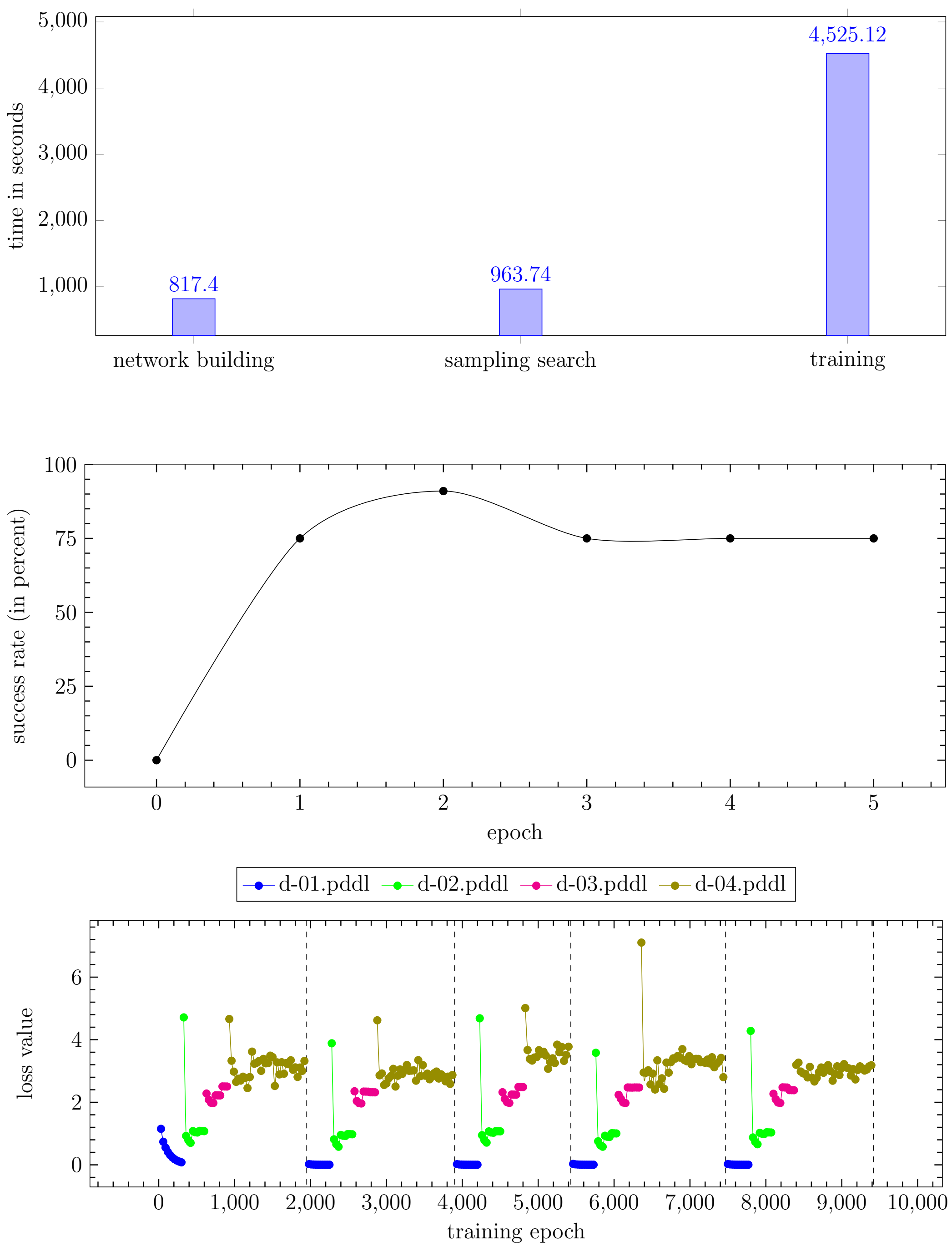


Figure A.14: Time distribution, success rate and loss development during training

3rd configuration: GBFS h^{FF} teacher

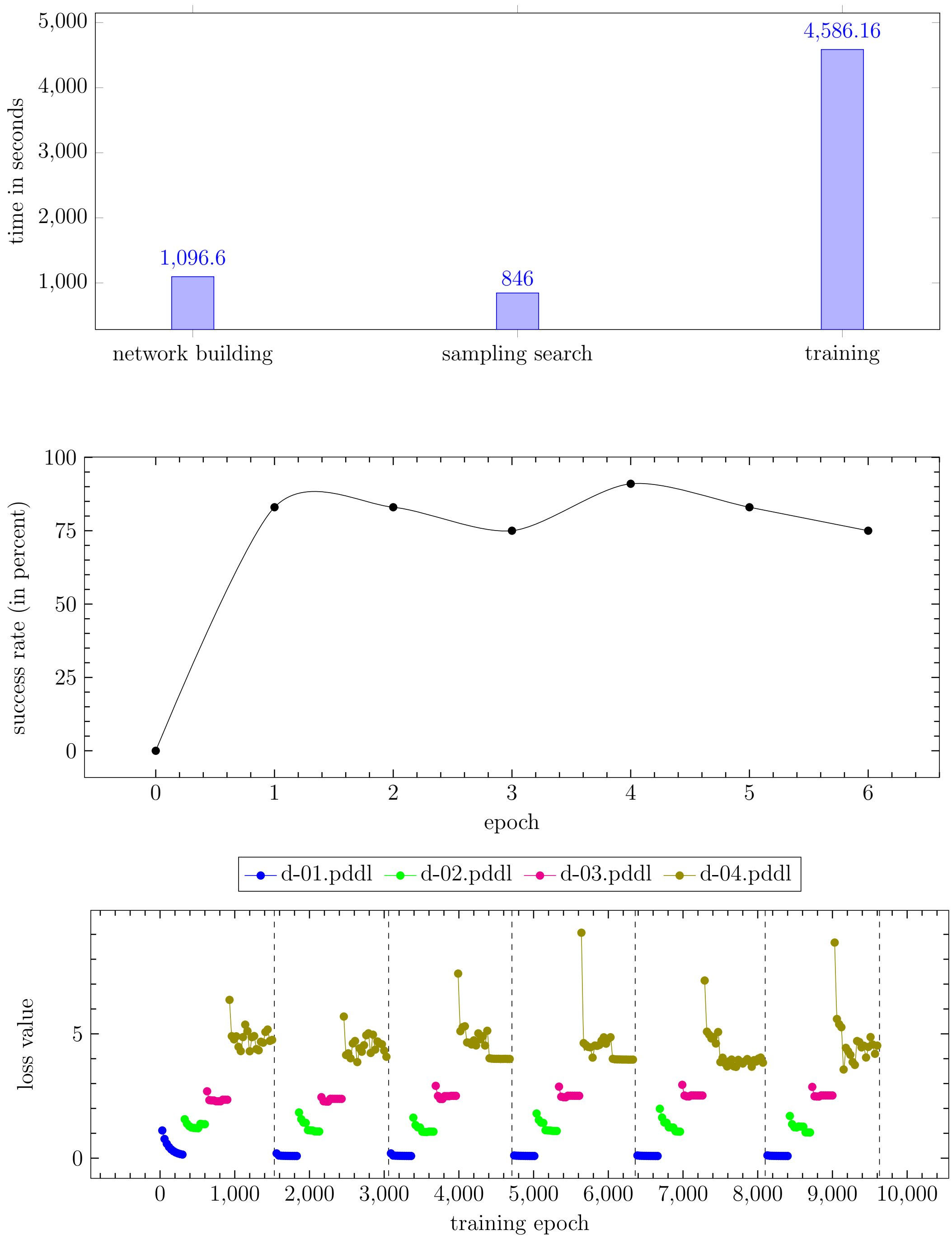
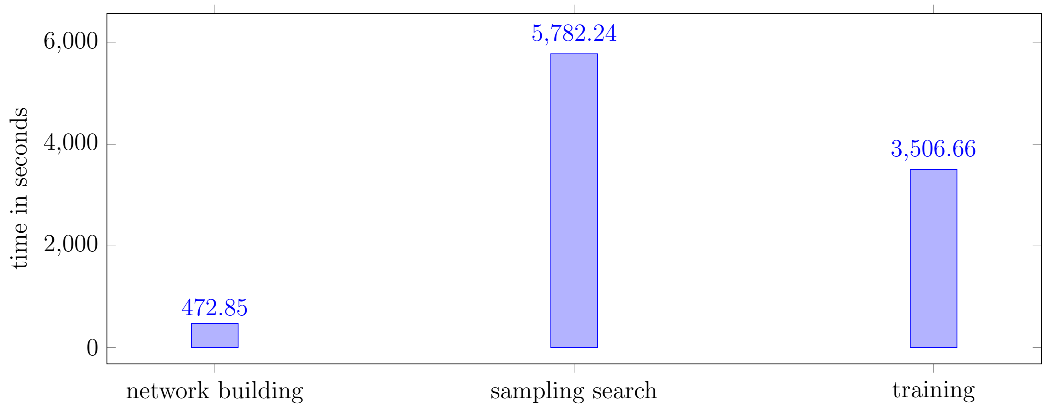
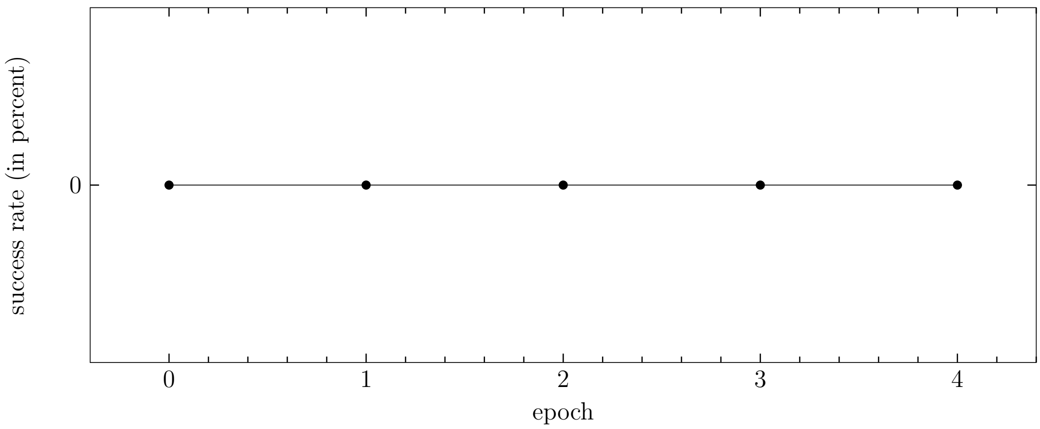


Figure A.15: Time distribution, success rate and loss development during training

A.6 Sokoban

1st configuration: A* h^{LM-cut} teacher





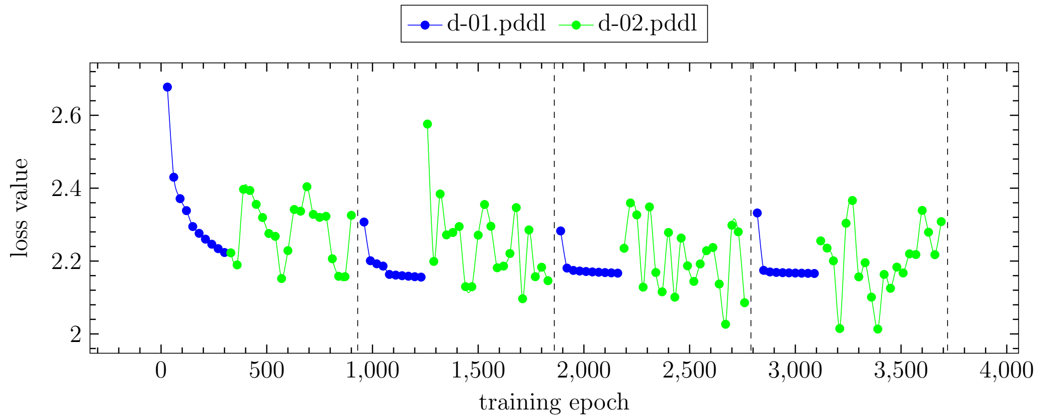


Figure A.16: Time distribution, success rate and loss development during training

2nd configuration: A* h^{add} teacher

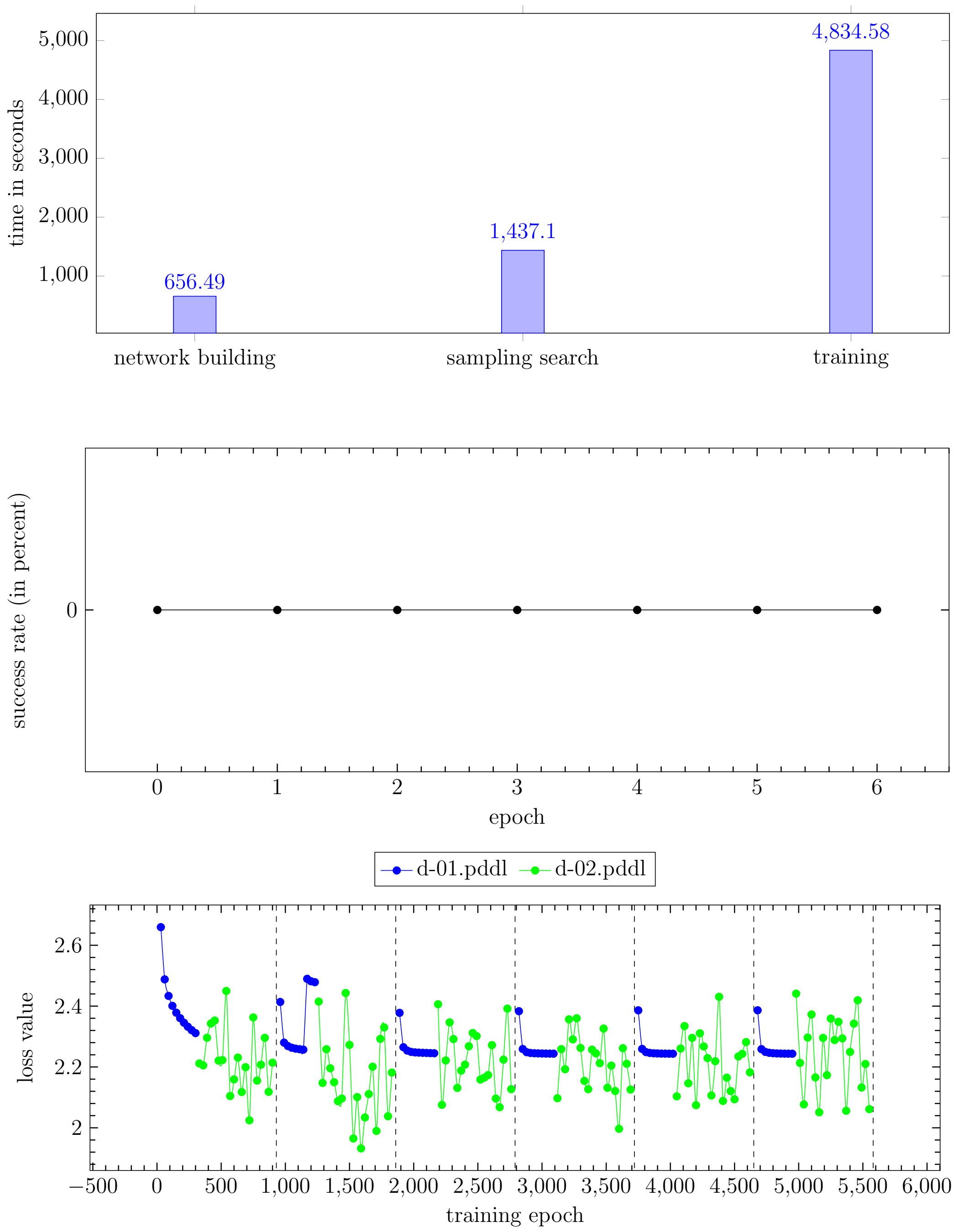


Figure A.17: Time distribution, success rate and loss development during training

3rd configuration: GBFS h^{FF} teacher

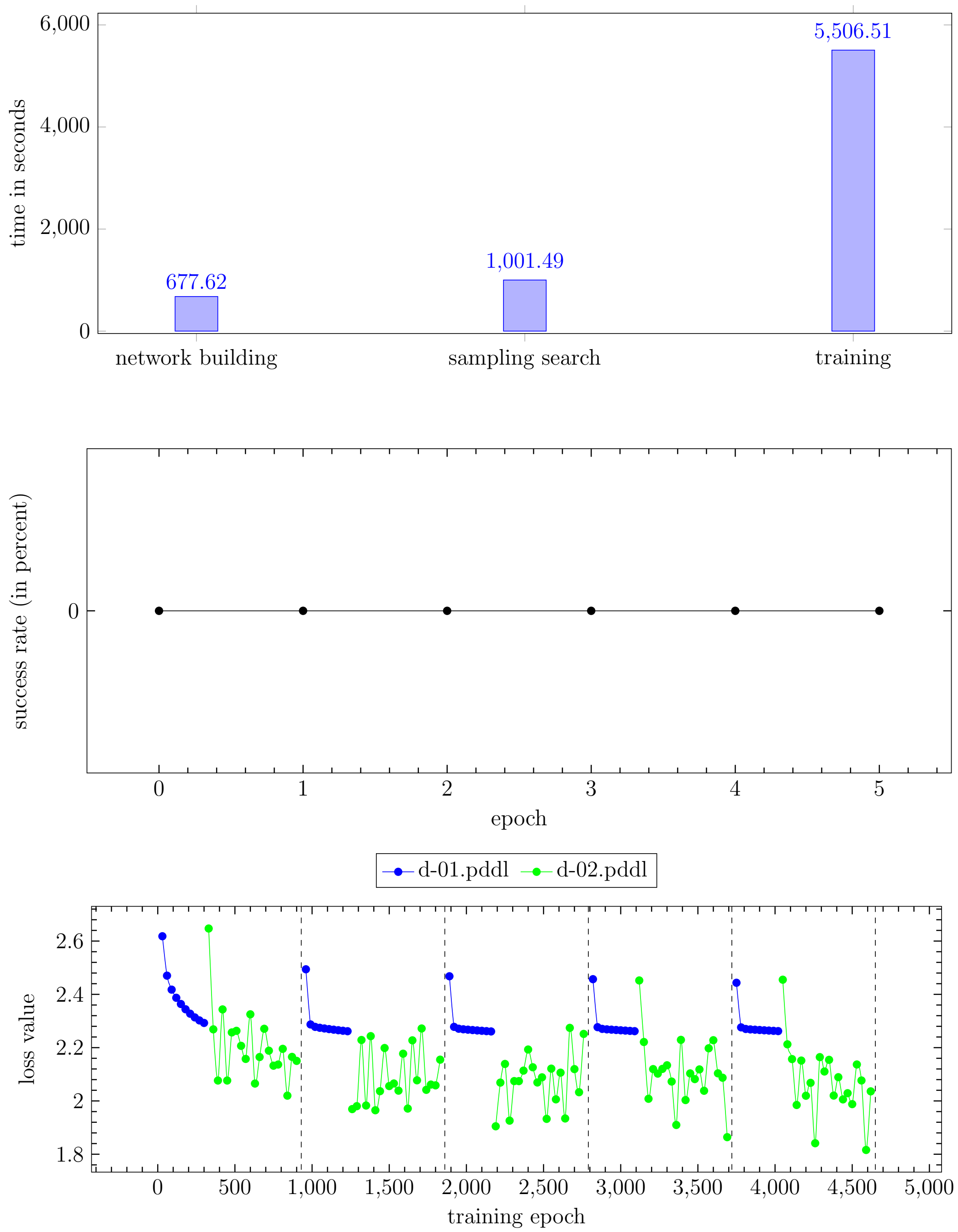
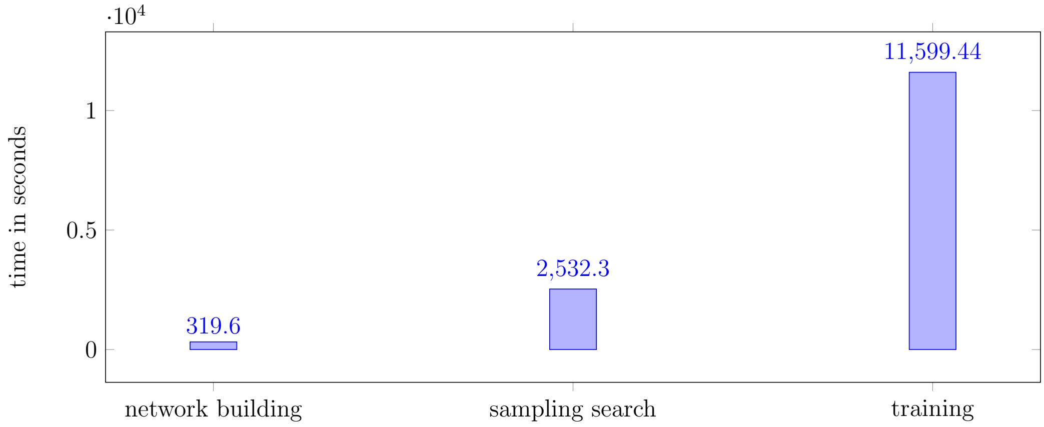
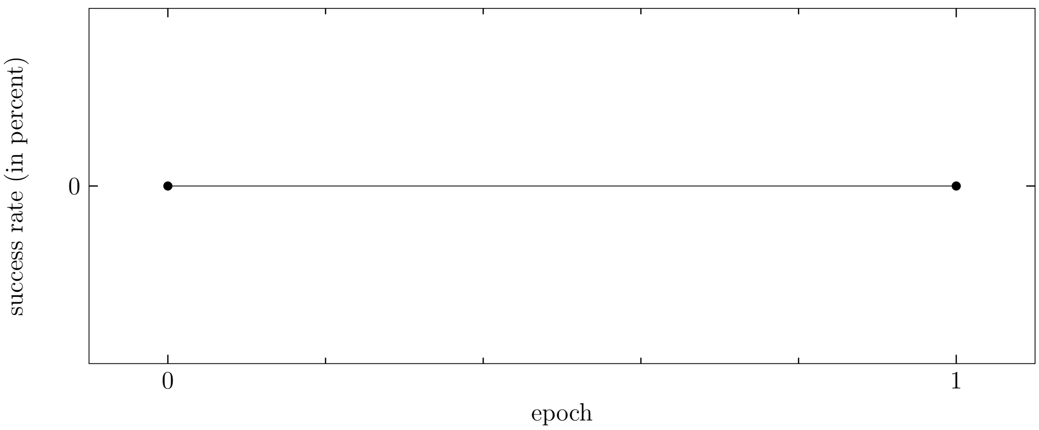


Figure A.18: Time distribution, success rate and loss development during training

A.7 TurnAndOpen

2nd configuration: A* h^{add} teacher





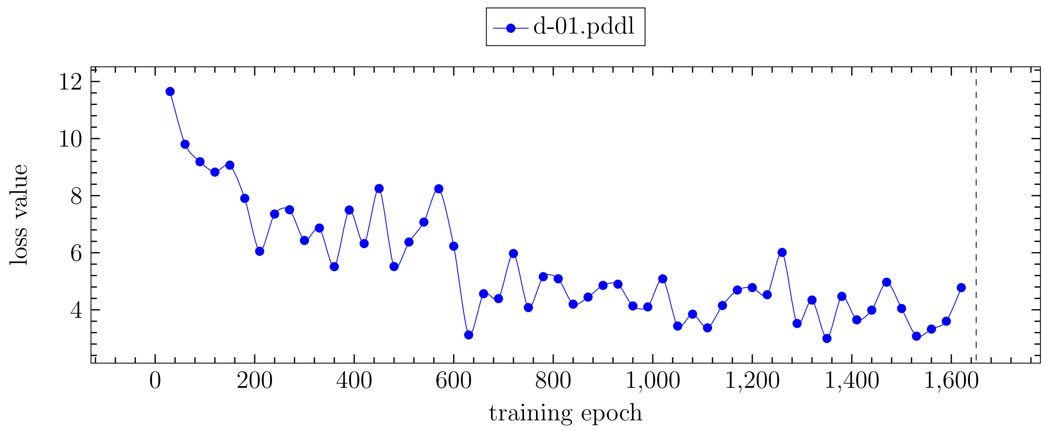
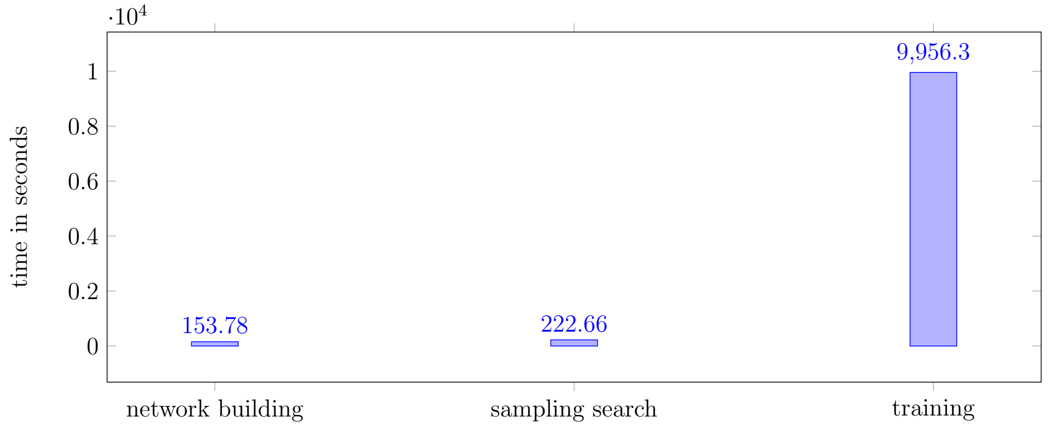
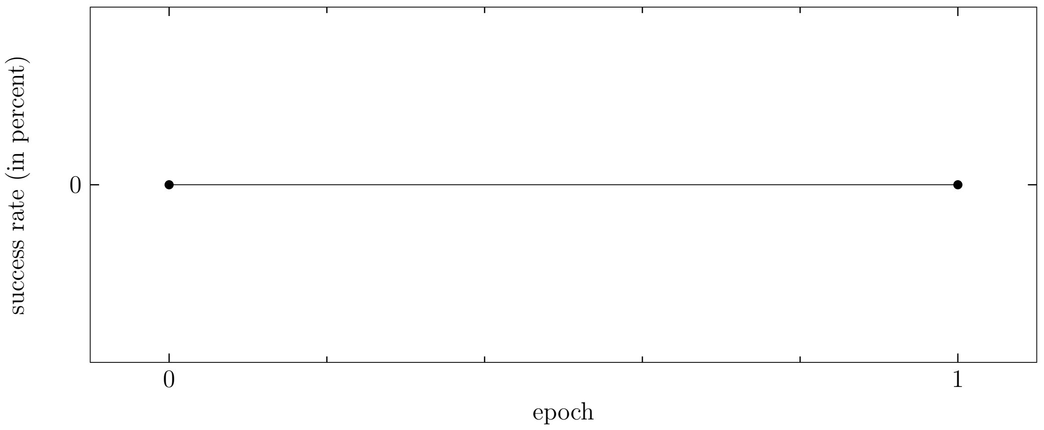


Figure A.19: Time distribution, success rate and loss development during training

3rd configuration: GBFS h^{FF} teacher





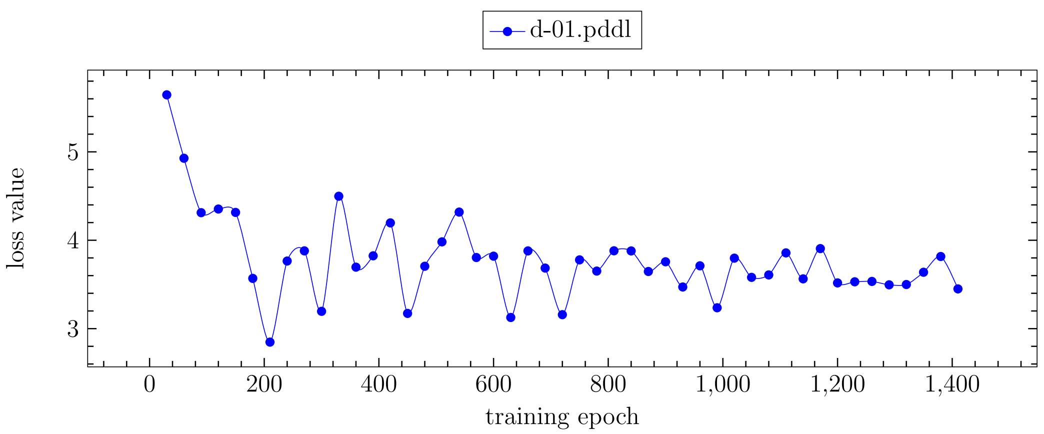


Figure A.20: Time distribution, success rate and loss development during training

A.8 Tyreworld

1st configuration: A* h^{LM-cut} teacher

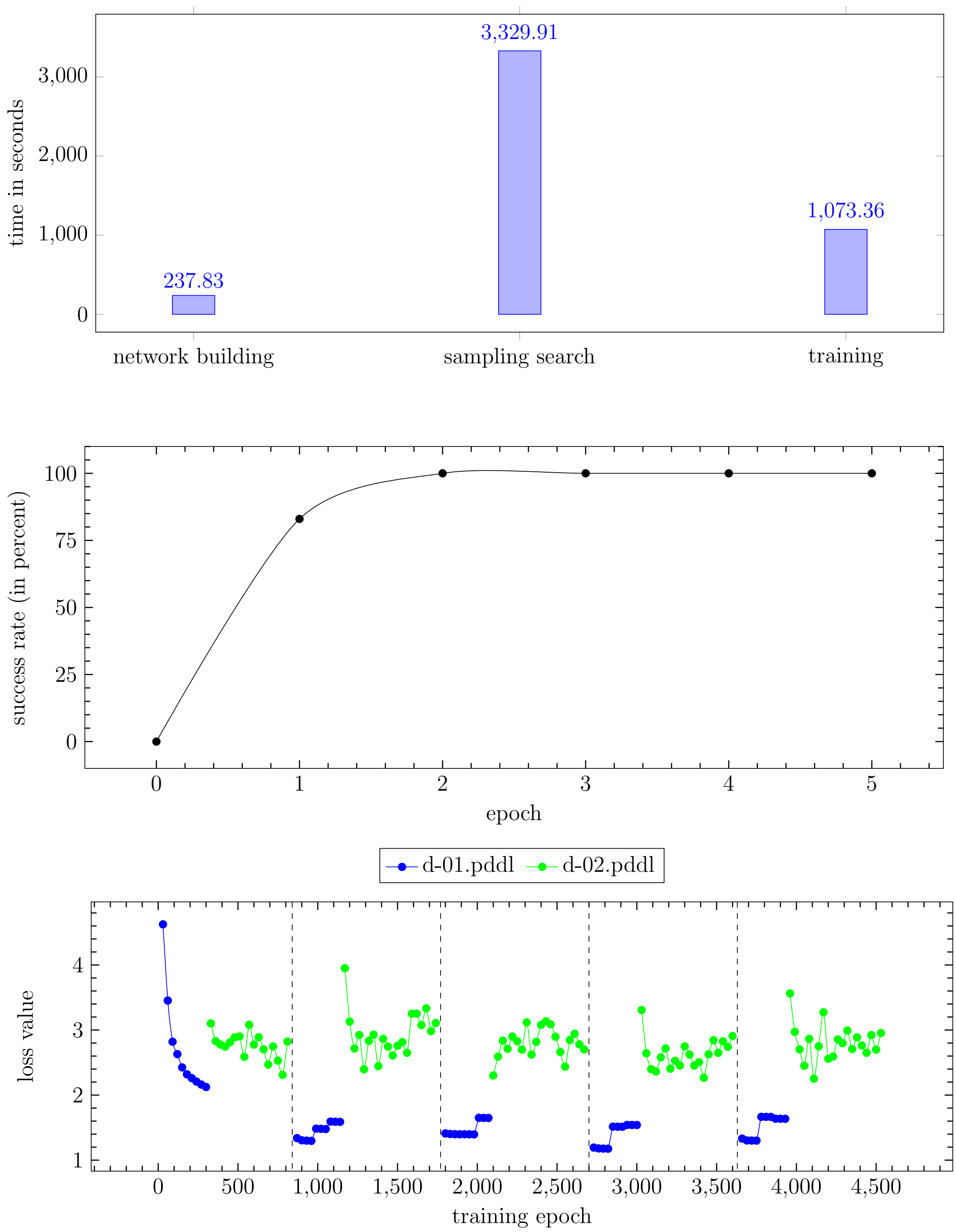


Figure A.21: Time distribution, success rate and loss development during training

2nd configuration: A* h^{add} teacher

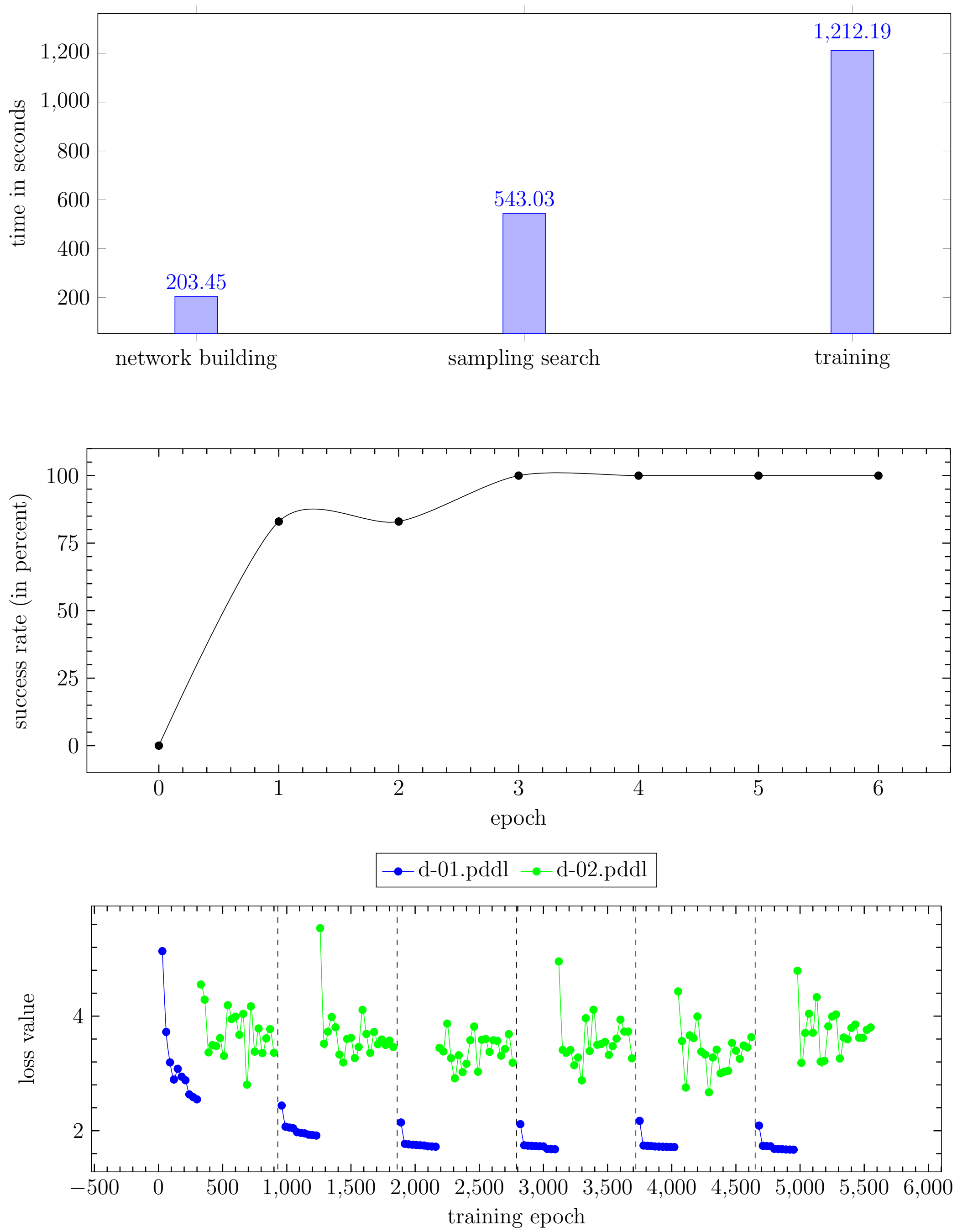


Figure A.22: Time distribution, success rate and loss development during training

3rd configuration: GBFS h^{FF} teacher

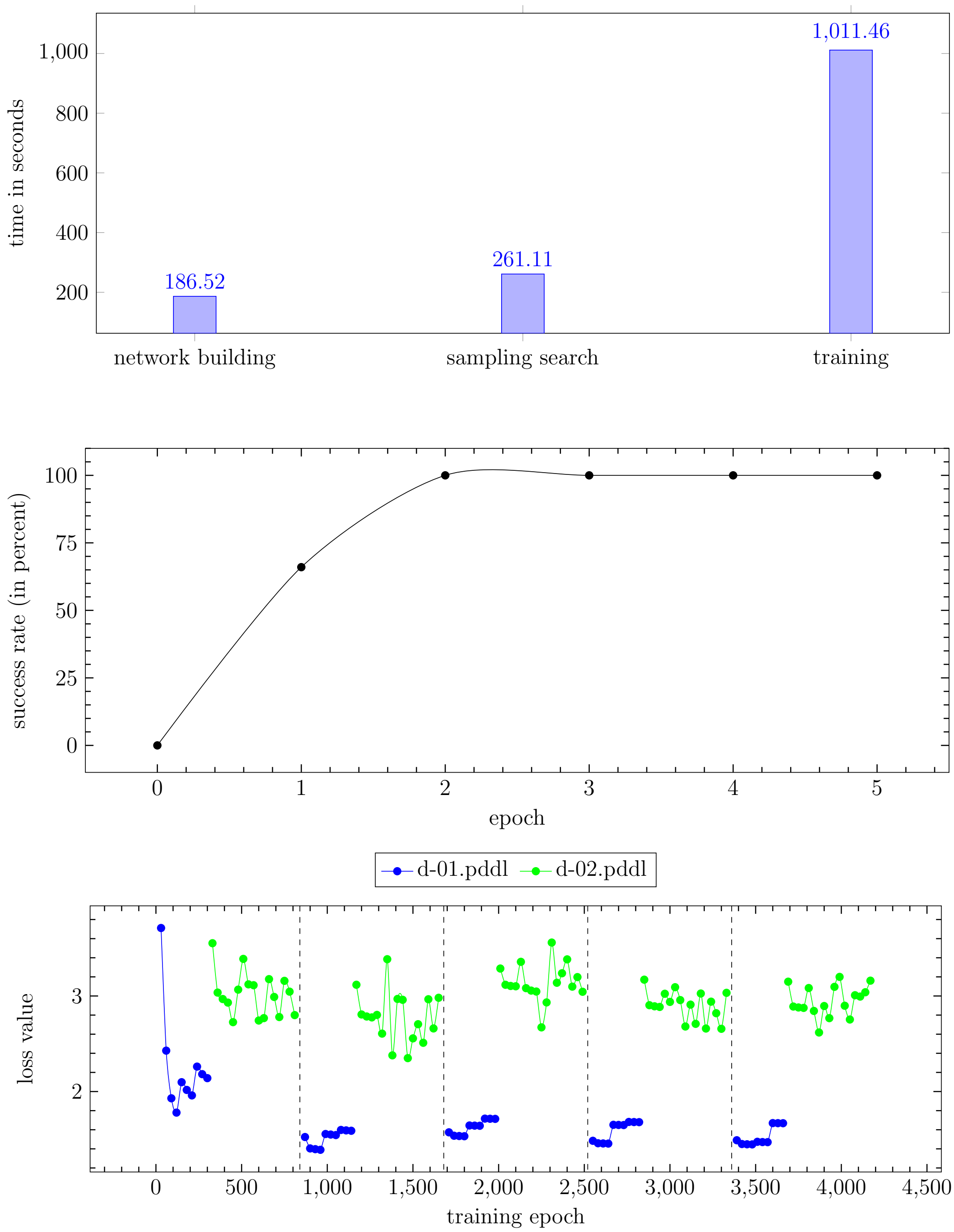


Figure A.23: Time distribution, success rate and loss development during training

Appendix B

Evaluation

In this appendix, we first provide an overview over the coverage of all baseline and network planners followed by detailed tables for each problem evaluated. Note that the networks are annotated as follows:

- ASNet LM: ASNets trained with A* h^{LM-cut} teacher search
- ASNet add: ASNets trained with A* h^{add} teacher search
- ASNet FF: ASNets trained with GBFS h^{FF} teacher search using a dual-queue of preferred operators

B.1 Coverage

Domain	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Blocksworld	28/35	35/35	35/35	35/35	7/35	7/35	4/35
Elevator	2/30	15/30	30/30	30/30	0/29	0/30	0/30
Floortile	6/20	20/20	9/20	9/20	0/20	1/20	0/20
Hanoi	13/20	15/20	16/20	15/20	3/20	2/20	2/20
Parcprinter	6/10	8/10	10/10	10/10	1/10	1/10	4/10
Sokoban	28/30	29/30	29/30	29/30	0/30	0/30	0/30
Turnandopen	0/19	17/19	15/19	19/19	0/19	0/19	0/19
Tyreworld	3/20	6/20	20/20	20/20	20/20	0/20	20/20

B.2 Problem Evaluations

For all problems of the considered domains, we provide the cost of extracted plans, the search time as well as the duration of the model creation for the ASNet configurations. For the LAMA baseline planner the search time is annotated as "/*" whenever the LAMA planner did not terminate in the 30 minutes time limit. Due to the iterative searches of this planner, we are still able to provide the plan costs from the last finished iteration. Additionally, it should be noted that training for the Turnandopen domain with the A* h^{LM-cut} teacher search did not finish and therefore no evaluation for this network configuration could be done in this domain.

Network creation for a domain was stopped after the process took longer than one hour (3600s) for a given problem, leading to the following larger problems not being evaluated.

B.2.1 Blocksworld

d-4-0	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	6	6	6	6	6	6	/
Search time	0.01s	0.01s	0.01s	0.01s	2.70s	2.67s	2.71s
Model creation time	-	-	-	-	10.00s	8.90s	8.80s

d-4-1	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	10	10	10	10	10	10	10
Search time	0.01s	0.01s	0.01s	0.01s	2.90s	2.92s	3.02s
Model creation time	-	-	-	-	12.66s	11.00s	11.06s

d-4-2	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	6	6	6	6	6	6	6
Search time	0.01s	0.01s	0.01s	0.01s	2.89s	2.41s	2.89s
Model creation time	-	-	-	-	9.62s	8.66s	8.70s

d-5-0	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	12	12	12	12	/	/	/
Search time	0.01s	0.01s	0.01s	0.07s	4.56s	4.54s	4.81s
Model creation time	-	-	-	-	14.47s	13.03s	12.95s

d-5-1	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	10	10	10	10	10	/	/
Search time	0.01s	0.01s	0.01s	0.06s	4.66s	4.44s	4.24s
Model creation time	-	-	-	-	14.59s	13.13s	13.05s

d-5-2	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	16	16	24	16	/	/	/
Search time	0.01s	0.01s	0.01s	0.08s	4.34s	4.62s	5.10s
Model creation time	-	-	-	-	14.47s	13.06s	13.07s

d-6-0	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	12	18	12	12	/	14	14
Search time	0.01s	0.01s	0.01s	0.28s	8.21s	7.33s	6.82s
Model creation time	-	-	-	-	21.18s	18.29s	18.22s

d-6-1	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	10	10	10	10	10	10	10
Search time	0.01s	0.01s	0.01s	0.60s	7.39s	7.52s	6.35s
Model creation time	-	-	-	-	20.65s	19.76s	18.41s

d-6-2	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	20	22	32	20	/	/	/
Search time	0.02s	0.01s	0.01s	0.50s	8.32s	5.88s	6.59s
Model creation time	-	-	-	-	20.43s	18.64s	18.42s

d-7-0	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	20	22	22	20	/	20	/
Search time	0.01s	0.01s	0.01s	3.23s	9.94s	10.72s	8.69s
Model creation time	-	-	-	-	27.08s	24.88s	24.72s

d-7-1	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	22	24	32	22	/	/	/
Search time	0.10s	0.01s	0.01s	5.33s	9.18s	9.10s	9.26s
Model creation time	-	-	-	-	27.31s	24.87s	24.43s

d-7-2	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	20	24	38	20	/	/	/
Search time	0.02s	0.01s	0.01s	5.03s	10.34s	8.90s	10.60s
Model creation time	-	-	-	-	27.68s	24.79s	24.62s

d-8-0	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	18	22	46	18	/	/	/
Search time	0.03s	0.01s	0.01s	48.11s	12.98s	14.44s	13.17s
Model creation time	-	-	-	-	35.25s	31.77s	31.69s

d-8-1	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	20	22	24	20	22	/	/
Search time	0.13s	0.01s	0.01s	86.47s	13.32s	16.22s	12.29s
Model creation time	-	-	-	-	35.37s	32.11s	31.92s

d-8-2	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	16	20	26	16	/	/	/
Search time	0.01s	0.01s	0.01s	60.76s	14.08s	11.52s	12.64s
Model creation time	-	-	-	-	35.63s	31.92s	31.85s

d-9-0	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	30	30	64	30	/	/	/
Search time	2.73s	0.05s	0.01s	455.69s	15.92s	16.29s	16.63s
Model creation time	-	-	-	-	44.71s	40.48s	40.32s

d-9-1	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	28	34	64	28	/	/	/
Search time	0.07s	0.05s	0.01s	362.49s	18.65s	16.15s	16.27s
Model creation time	-	-	-	-	46.23s	40.14s	40.29s

d-9-2	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	26	26	44	26	/	26	/
Search time	0.13s	0.02s	0.01s	1185.92s	16.82s	15.95s	15.75s
Model creation time	-	-	-	-	44.57s	40.54s	40.57s

d-10-0	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	34	38	48	34	/	/	/
Search time	65.74s	0.05s	0.01s	/*	24.61s	20.64s	19.87s
Model creation time	-	-	-	-	53.18s	53.57s	49.93s

d-10-1	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	32	36	56	32	/	/	/
Search time	9.09s	0.08s	0.01s	/*	20.92s	20.05s	23.20s
Model creation time	-	-	-	-	50.45s	49.73s	49.81s

d-10-2	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	34	40	52	34	/	/	/
Search time	25.76s	0.10s	0.01s	/*	25.10s	20.00s	20.12s
Model creation time	-	-	-	-	50.28s	49.73s	49.74s

d-11-0	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	32	38	46	32	/	/	/
Search time	25.09s	0.03s	0.01s	/*	25.17s	24.80s	28.34s
Model creation time	-	-	-	-	60.64s	60.27s	60.34s

d-11-1	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	30	38	110	30	/	/	/
Search time	29.93s	0.07s	0.01s	/*	28.18s	26.14s	24.01s
Model creation time	-	-	-	-	60.86s	60.59s	60.62s

d-11-2	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	34	38	68	34	/	/	/
Search time	20.70s	0.07s	0.01s	/*	26.02s	22.72s	24.62s
Model creation time	-	-	-	-	60.68s	60.36s	60.15s

d-12-0	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	34	46	78	34	/	/	/
Search time	34.08s	0.39s	0.01s	/*	41.86s	26.06s	31.96s
Model creation time	-	-	-	-	72.26s	71.86s	71.81s

d-12-1	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	34	48	58	34	/	/	/
Search time	3.32s	1.48s	0.01s	/*	32.68s	32.01s	37.49s
Model creation time	-	-	-	-	72.56s	72.04s	71.96s

d-13-0	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	/	50	88	44	/	/	/
Search time	/	3.31s	0.02s	/*	37.87s	43.33s	35.76s
Model creation time	-	-	-	-	85.50s	85.06s	85.04s

d-13-1	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	/	52	104	48	44	/	/
Search time	/	0.38s	0.01s	/*	41.54s	37.10s	34.95s
Model creation time	-	-	-	-	85.17s	84.73s	84.75s

d-14-0	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	38	54	84	38	/	/	/
Search time	101.66s	1.91s	0.01s	/*	49.24s	41.42s	51.06s
Model creation time	-	-	-	-	99.44s	98.89s	98.53s

d-14-1	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	36	48	74	36	/	/	/
Search time	200.32s	1.12s	0.01s	/*	47.95s	41.58s	55.72s
Model creation time	-	-	-	-	99.44s	98.89s	98.84s

d-15-0	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	/	52	158	54	/	/	/
Search time	/	3.39s	0.06s	/*	56.66s	49.55s	55.11s
Model creation time	-	-	-	-	114.58s	114.25s	114.38s

d-15-1	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	/	60	120	66	/	/	/
Search time	/	0.46s	0.03s	/*	49.75s	58.93s	50.30s
Model creation time	-	-	-	-	114.71s	114.58s	114.33s

d-16-1	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	/	64	106	54	/	/	/
Search time	/	1.52s	0.04s	/*	60.56s	54.91s	65.03s
Model creation time	-	-	-	-	131.01s	130.97s	128.79s

d-16-2	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	/	64	160	60	/	/	/
Search time	/	5.99s	0.12s	/*	63.51s	71.35s	55.39s
Model creation time	-	-	-	-	135.12s	130.50s	129.10s

d-17-0	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	/	62	132	46	/	/	/
Search time	/	2.43s	0.07s	/*	73.00s	64.92s	63.02s
Model creation time	-	-	-	-	164.68s	148.77s	147.06s

B.2.2 Elevator

d-01	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	52	64	86	52	/	/	/
Search time	1.66s	0.04s	0.01s	35.91s	41.58s	56.27s	44.84s
Model creation time	-	-	-	-	144.42s	127.53s	1138.62s

d-02	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	53	70	88	53	/	/	/
Search time	6.59s	0.06s	0.01s	460.76s	62.87s	55.21s	53.25s
Model creation time	-	-	-	-	147.29s	133.10s	13540.41s

d-03	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	/	77	130	66	/	/	/
Search time	/	0.12s	0.01s	/*	69.84s	66.10s	101.08s
Model creation time	-	-	-	-	174.97s	158.46s	186.57s

d-04	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	/	112	169	89	/	/	/
Search time	/	0.80s	0.02s	/*	80.11s	76.39s	76.88s
Model creation time	-	-	-	-	201.74s	196.64s	216.86s

d-05	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	/	101	171	73	/	/	/
Search time	/	4.35s	0.02s	/*	88.81s	86.48s	95.20s
Model creation time	-	-	-	-	230.32s	211.49s	7067.69s

d-06	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	/	113	140	138	/	/	/
Search time	/	3.06s	0.03s	/*	94.40s	99.30s	93.06s
Model creation time	-	-	-	-	259.85s	237.61s	276.52s

d-07	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	/	137	215	165	/	/	/
Search time	/	14.09s	0.04s	/*	123.50s	102.59s	97.82s
Model creation time	-	-	-	-	287.90s	264.07s	348.60s

d-08	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	/	143	150	192	/	/	/
Search time	/	42.57s	0.03s	/*	123.28s	118.41s	124.53s
Model creation time	-	-	-	-	318.45s	290.13s	371.44s

d-09	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	/	132	237	99	/	/	/
Search time	/	41.42s	0.04s	/*	127.67s	124.31s	161.21s
Model creation time	-	-	-	-	359.87s	317.66s	397.52s

d-10	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	/	206	289	314	/	/	/
Search time	/	82.43s	0.07s	/*	141.54s	156.93s	173.19s
Model creation time	-	-	-	-	360.66s	344.52s	5741.19s

d-11	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	/	107	162	115	/	/	/
Search time	/	3.45s	0.04s	/*	195.73s	151.39s	172.85s
Model creation time	-	-	-	-	470.82s	416.38s	493.89s

d-12	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	/	175	283	130	/	/	/
Search time	/	6.43s	0.07s	/*	240.62s	191.65s	225.97s
Model creation time	-	-	-	-	595.36s	526.57s	888.80s

d-13	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	/	189	259	154	/	/	/
Search time	/	135.75s	0.08s	/*	241.56s	257.60s	262.86s
Model creation time	-	-	-	-	720.94s	642.54s	743.57s

d-14	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	/	/	336	346	/	/	/
Search time	/	/	0.17s	/*	295.56s	298.22s	304.40s
Model creation time	-	-	-	-	859.26s	764.18s	895.33s

d-15	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	/	/	405	287	/	/	/
Search time	/	/	0.19s	/*	327.75s	371.98s	319.88s
Model creation time	-	-	-	-	1008.58s	889.63s	1036.81s

d-16	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	/	/	434	278	/	/	/
Search time	/	/	0.20s	/*	366.44s	421.13s	357.89s
Model creation time	-	-	-	-	1164.60s	1024.25s	2603.03s

d-17	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	/	306	468	543	/	/	/
Search time	/	578.21s	0.31s	/*	409.36s	416.47s	441.96s
Model creation time	-	-	-	-	1303.02s	1148.05s	3373.55s

d-18	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	/	/	358	503	/	/	/
Search time	/	/	0.34s	/*	515.35s	482.11s	466.46s
Model creation time	-	-	-	-	1311.53s	1304.27s	1493.18s

d-19	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	/	/	545	532	/	/	/
Search time	/	/	0.63s	/*	519.93s	505.37s	517.24s
Model creation time	-	-	-	-	1451.45s	1451.56s	1872.24s

d-20	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	/	/	638	619	/	/	/
Search time	/	/	0.48s	/*	563.44s	553.81s	531.74s
Model creation time	-	-	-	-	1602.99s	2094.33s	1932.06s

d-21	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	/	264	375	259	/	/	/
Search time	/	1720.34s	0.22s	/*	606.27s	520.85s	554.17s
Model creation time	-	-	-	-	1506.29s	2796.94s	3520.01s

d-22	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	/	/	397	411	/	/	/
Search time	/	/	0.35s	/*	717.98s	643.01s	756.26s
Model creation time	-	-	-	-	1971.70s	3923.51s	3540.82s

d-23	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	/	/	508	502	/	/	/
Search time	/	/	0.61s	/*	940.72s	822.53s	1036.31s
Model creation time	-	-	-	-	2508.22s	2467.26s	2441.84s

d-24	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	/	/	736	962	/	/	/
Search time	/	/	1.02s	/*	1027.24s	884.21s	1140.35s
Model creation time	-	-	-	-	3106.12s	3043.61s	3057.74s

d-25	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	/	/	634	902	/	/	/
Search time	/	/	1.82s	/*	/	1176.45s	1335.53s
Model creation time	-	-	-	-	3709.23s	3650.00s	3674.30s

d-26	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	/	/	891	741	/	/	/
Search time	/	/	1.46s	/*	/	/	/
Model creation time	-	-	-	-	/	/	/

d-27	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	/	/	845	1071	/	/	/
Search time	/	/	3.45s	/*	/	/	/
Model creation time	-	-	-	-	/	/	/

d-28	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	/	/	1131	1015	/	/	/
Search time	/	/	4.69s	/*	/	/	/
Model creation time	-	-	-	-	/	/	/

d-29	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	/	/	1285	896	/	/	/
Search time	/	/	21.99s	/*	/	/	/
Model creation time	-	-	-	-	/	/	/

d-30	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	/	/	805	1006	/	/	/
Search time	/	/	3.03s	/*	/	/	/
Model creation time	-	-	-	-	/	/	/

B.2.3 Floortile

d-43-0	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	56	56	110	56	/	/	/
Search time	32.90s	0.05s	11.67s	341.01s	19.08s	16.08s	20.74s
Model creation time	-	-	-	-	48.21s	43.01s	42.93s

d-44-0	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	68	72	127	102	/	/	/
Search time	1166.37s	0.45s	101.54s	/*	23.16s	25.12s	26.07s
Model creation time	-	-	-	-	64.40s	57.97s	58.00s

d-53-0	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	63	63	158	63	/	75	/
Search time	167.24s	0.17s	142.93s	/*	21.99s	24.94s	20.68s
Model creation time	-	-	-	-	58.50s	52.84s	53.05s

d-54-0	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	/	93	/	/	/	/	/
Search time	/	0.16s	/	/	32.59s	31.27s	34.59s
Model creation time	-	-	-	-	80.32s	72.18s	72.07s

d-55-0	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	/	121	/	/	/	/	/
Search time	/	2.15s	/	/	43.92s	47.45s	37.34s
Model creation time	-	-	-	-	101.22s	91.71s	91.78s

d-64-0	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	/	114	/	/	/	/	/
Search time	/	1.16s	/	/	34.15s	34.90s	39.93s
Model creation time	-	-	-	-	95.20s	85.75s	85.74s

d-65-0	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	/	136	/	/	/	/	/
Search time	/	2.30s	/	/	53.52s	52.61s	48.88s
Model creation time	-	-	-	-	122.78s	109.54s	109.65s

d-44-1	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	/	75	169	147	/	/	/
Search time	/	0.12s	476.51s	/*	23.10s	24.27s	26.94s
Model creation time	-	-	-	-	67.55s	59.16s	59.08s

d-53-1	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	65	65	140	65	/	/	/
Search time	240.32s	0.05s	74.94s	/*	22.46s	21.80s	22.65s
Model creation time	-	-	-	-	59.02s	53.04s	53.09s

d-54-1	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	/	98	/	/	/	/	/
Search time	/	0.17s	/	/	28.92s	30.97s	28.42s
Model creation time	-	-	-	-	80.53s	71.60s	71.76s

d-55-1	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	/	116	/	/	/	/	/
Search time	/	1.95s	/	/	36.90s	38.32s	51.90s
Model creation time	-	-	-	-	101.99s	91.98s	90.88s

d-64-1	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	/	117	/	/	/	/	/
Search time	/	1.28s	/	/	38.87s	38.36s	39.54s
Model creation time	-	-	-	-	95.38s	85.56s	84.64s

d-65-1	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	/	157	/	/	/	/	/
Search time	/	2.32s	/	/	49.23s	45.98s	48.53s
Model creation time	-	-	-	-	123.74s	109.32s	108.28s

d-43-2	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	56	56	115	56	/	/	/
Search time	45.89s	0.05s	22.09s	377.68s	19.21s	16.92s	16.97s
Model creation time	-	-	-	-	50.68s	44.41s	43.99s

d-44-2	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	/	76	133	116	/	/	/
Search time	/	0.38s	208.12s	/*	26.64s	23.83s	23.92s
Model creation time	-	-	-	-	64.44s	58.16s	57.71s

d-53-2	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	65	65	138	65	/	/	/
Search time	251.60s	0.07s	65.28s	/*	23.28s	21.42s	20.87s
Model creation time	-	-	-	-	60.15s	53.08s	52.79s

d-54-2	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	/	85	196	188	/	/	/
Search time	/	0.20s	126.06s	/*	33.55s	30.88s	35.51s
Model creation time	-	-	-	-	79.08s	71.50s	71.19s

d-55-2	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	/	112	/	/	/	/	/
Search time	/	2.09s	/	/	36.30s	43.52s	48.95s
Model creation time	-	-	-	-	101.11s	91.80s	91.07s

d-64-2	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	/	124	/	/	/	/	/
Search time	/	1.36s	/	/	38.10s	33.80s	39.82s
Model creation time	-	-	-	-	97.02s	86.46s	86.05s

d-65-2	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	/	144	/	/	/	/	/
Search time	/	2.40s	/	/	52.85s	53.00s	45.96s
Model creation time	-	-	-	-	121.40s	109.24s	109.09s

B.2.4 Hanoi

d-01	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	1	1	1	1	1	1	1
Search time	0.01s	0.01s	0.01s	0.01s	0.50s	0.54s	0.54s
Model creation time	-	-	-	-	4.61s	3.95s	4.02s

d-02	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	3	3	3	3	3	3	/
Search time	0.01s	0.01s	0.01s	0.01s	1.72s	1.40s	1.47s
Model creation time	-	-	-	-	5.55s	4.64s	4.62s

d-03	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	7	7	7	7	7	/	7
Search time	0.01s	0.01s	0.01s	0.01s	3.43s	3.23s	3.02s
Model creation time	-	-	-	-	10.90s	9.16s	9.11s

d-04	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	15	15	15	15	/	/	/
Search time	0.01s	0.01s	0.01s	0.01s	6.67s	5.68s	7.60s
Model creation time	-	-	-	-	19.03s	17.15s	15.66s

d-05	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	31	31	39	31	/	/	/
Search time	0.01s	0.01s	0.01s	0.05s	10.61s	9.39s	9.47s
Model creation time	-	-	-	-	27.91s	27.43s	25.34s

d-06	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	63	63	64	63	/	/	/
Search time	0.03s	0.01s	0.01s	0.19s	17.02s	13.58s	19.41s
Model creation time	-	-	-	-	41.72s	38.43s	37.73s

d-07	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	127	127	151	127	/	/	/
Search time	0.12s	0.04s	0.05s	0.90s	22.35s	20.87s	21.43s
Model creation time	-	-	-	-	59.67s	54.59s	54.35s

d-08	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	255	255	303	255	/	/	/
Search time	0.50s	0.14s	0.09s	4.34s	29.90s	32.23s	28.31s
Model creation time	-	-	-	-	83.30s	75.72s	74.98s

d-09	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	511	511	645	511	/	/	/
Search time	2.22s	0.53s	0.23s	18.18s	46.60s	38.84s	38.85s
Model creation time	-	-	-	-	110.44s	101.08s	100.68s

d-10	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	1023	1023	1280	1023	/	/	/
Search time	9.14s	2.01s	0.71s	80.67s	53.62s	51.50s	54.19s
Model creation time	-	-	-	-	144.51s	132.58s	131.66s

d-11	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	2047	2047	2626	2047	/	/	/
Search time	38.12s	7.81s	2.59s	371.95s	69.98s	68.16s	71.59s
Model creation time	-	-	-	-	187.95s	170.21s	168.86s

d-12	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	4095	4095	5380	4095	/	/	/
Search time	151.88s	29.78s	9.04s	/*	94.42s	77.57s	128.41s
Model creation time	-	-	-	-	241.73s	214.76s	213.17s

d-13	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	8191	8191	10713	9563	/	/	/
Search time	650.62s	115.92s	34.07s	/*	100.31s	145.58s	145.81s
Model creation time	-	-	-	-	299.05s	268.49s	266.07s

d-14	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	/	16383	21701	19297	/	/	/
Search time	/	435.81s	118.62s	/*	132.13s	129.75s	138.88s
Model creation time	-	-	-	-	365.73s	330.91s	330.19s

d-15	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	/	32767	43447	38775	/	/	/
Search time	/	1630.76s	443.40s	/*	187.26s	152.76s	168.93s
Model creation time	-	-	-	-	446.98s	405.94s	407.05s

d-16	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	/	/	90757	/	/	/	/
Search time	/	/	1513.55s	/	187.82s	187.31s	202.53s
Model creation time	-	-	-	-	490.89s	492.77s	492.58s

d-17	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	/	/	/	/	/	/	/
Search time	/	/	/	/	232.84s	263.38s	216.00s
Model creation time	-	-	-	-	590.72s	592.89s	587.40s

d-18	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	/	/	/	/	/	/	/
Search time	/	/	/	/	272.79s	297.85s	253.40s
Model creation time	-	-	-	-	706.25s	707.46s	700.12s

d-19	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	/	/	/	/	/	/	/
Search time	/	/	/	/	311.93s	318.28s	305.88s
Model creation time	-	-	-	-	843.59s	846.80s	836.55s

d-20	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	/	/	/	/	/	/	/
Search time	/	/	/	/	413.44s	371.42s	423.13s
Model creation time	-	-	-	-	1038.27s	1054.61s	987.07s

B.2.5 ParcPrinter

d-01	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	169009	169009	269038	169009	169009	169009	269038
Search time	0.01s	0.01s	0.01s	0.01s	5.23s	4.61s	5.19s
Model creation time	-	-	-	-	19.09s	19.11s	19.05s

d-02	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	438047	438047	538076	438047	/	/	/
Search time	0.01s	0.01s	0.01s	0.04s	9.80s	8.48s	8.93s
Model creation time	-	-	-	-	42.18s	43.16s	42.20s

d-03	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	807114	807114	807114	807114	/	/	807114
Search time	0.01s	0.01s	0.01s	0.12s	7.80s	8.02s	7.70s
Model creation time	-	-	-	-	23.59s	23.72s	23.37s

d-04	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	876094	876094	1076152	876094	/	/	/
Search time	0.02s	0.01s	0.01s	11.52s	21.79s	22.25s	32.17s
Model creation time	-	-	-	-	56.72s	57.54s	56.63s

d-05	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	1145132	1145132	1345190	1145132	/	/	1345190
Search time	0.15s	0.01s	0.01s	162.47s	25.98s	28.94s	35.08s
Model creation time	-	-	-	-	73.39s	73.57s	72.74s

d-06	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	1514199	1514199	1614228	1514199	/	/	1614228
Search time	24.15s	0.04s	0.01s	1097.90s	34.71s	28.48s	30.22s
Model creation time	-	-	-	-	81.71s	82.04s	81.17s

d-07	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	/	1383121	1883266	1883266	/	/	/
Search time	/	0.02s	0.01s	/*	49.68s	54.30s	54.17s
Model creation time	-	-	-	-	128.78s	129.16s	127.98s

d-08	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	/	1852217	2152304	2152304	/	/	/
Search time	/	0.27s	0.01s	/*	52.72s	49.15s	51.99s
Model creation time	-	-	-	-	135.01s	135.19s	133.91s

d-09	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	/	/	2421342	2421342	/	/	/
Search time	/	/	0.01s	/*	62.69s	59.56s	62.27s
Model creation time	-	-	-	-	162.80s	154.26s	154.13s

d-10	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	/	/	2690380	2690380	/	/	/
Search time	/	/	0.01s	/*	73.77s	67.16s	70.48s
Model creation time	-	-	-	-	184.14s	166.06s	165.75s

B.2.6 Sokoban

d-01	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	9	13	13	9	/	/	/
Search time	0.01s	0.01s	0.01s	0.11s	12.94s	11.69s	14.65s
Model creation time	-	-	-	-	36.64s	32.49s	32.45s

d-02	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	29	29	37	29	/	/	/
Search time	1.24s	0.10s	0.04s	16.76s	28.63s	24.65s	22.74s
Model creation time	-	-	-	-	68.00s	61.49s	61.18s

d-03	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	9	9	13	9	/	/	/
Search time	0.02s	0.01s	0.01s	0.12s	16.82s	15.32s	16.10s
Model creation time	-	-	-	-	43.96s	40.23s	39.79s

d-04	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	31	31	35	31	/	/	/
Search time	40.07s	1.87s	1.76s	496.74s	38.31s	37.10s	41.52s
Model creation time	-	-	-	-	104.39s	93.22s	93.30s

d-05	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	30	30	40	30	/	/	/
Search time	0.97s	0.29s	0.36s	7.43s	25.21s	25.64s	30.46s
Model creation time	-	-	-	-	72.53s	66.74s	66.56s

d-06	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	32	32	48	32	/	/	/
Search time	2.61s	0.77s	0.53s	20.72s	45.41s	41.58s	45.66s
Model creation time	-	-	-	-	107.49s	99.01s	98.77s

d-07	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	29	29	33	29	/	/	/
Search time	0.46s	0.14s	0.06s	2.25s	34.12s	25.83s	26.20s
Model creation time	-	-	-	-	72.40s	65.64s	65.59s

d-08	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	50	52	62	50	/	/	/
Search time	18.47s	6.00s	0.96s	34.33s	38.10s	33.79s	40.80s
Model creation time	-	-	-	-	93.10s	83.48s	83.13s

d-09	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	49	49	69	49	/	/	/
Search time	13.34s	2.49s	1.41s	54.44s	46.24s	40.45s	48.96s
Model creation time	-	-	-	-	115.69s	104.42s	104.22s

d-10	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	2	2	2	2	/	/	/
Search time	0.04s	0.04s	0.11s	0.65s	64.64s	55.13s	65.29s
Model creation time	-	-	-	-	150.37s	135.51s	135.04s

d-11	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	10	10	10	10	/	/	/
Search time	2.86s	0.65s	0.15s	43.28s	163.57s	153.41s	171.31s
Model creation time	-	-	-	-	432.21s	392.66s	392.23s

d-12	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	44	44	92	44	/	/	/
Search time	107.91s	7.06s	2.27s	1797.71s	60.77s	48.78s	44.44s
Model creation time	-	-	-	-	147.77s	124.30s	124.17s

d-13	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	31	31	31	31	/	/	/
Search time	27.20s	0.77s	0.59s	1247.23s	37.36s	38.88s	44.07s
Model creation time	-	-	-	-	116.65s	98.52s	98.55s

d-14	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	50	50	84	50	/	/	/
Search time	83.27s	12.04s	5.85s	301.75s	48.86s	42.97s	48.71s
Model creation time	-	-	-	-	127.96s	115.78s	115.55s

d-15	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	39	43	59	53	/	/	/
Search time	365.75s	159.86s	337.43s	/*	54.55s	46.85s	61.96s
Model creation time	-	-	-	-	134.13s	121.77s	121.84s

d-16	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	33	33	57	33	/	/	/
Search time	269.42s	8.29s	1.99s	1507.43s	60.29s	56.36s	83.02s
Model creation time	-	-	-	-	146.61s	132.84s	132.67s

d-17	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	23	23	39	23	/	/	/
Search time	117.68s	1.19s	0.65s	/*	75.97s	64.16s	85.80s
Model creation time	-	-	-	-	191.19s	172.59s	172.56s

d-18	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	43	43	61	43	/	/	/
Search time	1323.51s	93.14s	65.24s	/*	48.59s	52.33s	50.20s
Model creation time	-	-	-	-	138.69s	125.27s	125.00s

d-19	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	18	18	24	18	/	/	/
Search time	30.06s	4.17s	1.73s	/*	196.10s	223.91s	200.40s
Model creation time	-	-	-	-	576.74s	522.61s	516.96s

d-20	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	14	/	/	/	/	/	/
Search time	1786.92s	/	/	/	222.88s	267.40s	257.48s
Model creation time	-	-	-	-	597.60s	574.60s	563.15s

d-21	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	51	51	71	51	/	/	/
Search time	1714.65s	85.33s	8.63s	/*	74.79s	65.84s	64.03s
Model creation time	-	-	-	-	154.57s	155.41s	152.36s

d-22	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	60	60	84	60	/	/	/
Search time	1543.80s	97.47s	6.12s	/*	64.47s	76.99s	59.88s
Model creation time	-	-	-	-	142.58s	143.72s	141.12s

d-23	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	46	46	78	46	/	/	/
Search time	601.42s	96.80s	14.22s	929.85s	56.70s	55.43s	58.22s
Model creation time	-	-	-	-	133.96s	135.00s	132.55s

d-24	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	47	47	73	47	/	/	/
Search time	93.40s	7.61s	8.62s	519.83s	53.89s	53.70s	53.80s
Model creation time	-	-	-	-	131.53s	132.41s	130.06s

d-25	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	52	56	78	52	/	/	/
Search time	100.65s	7.28s	3.49s	242.99s	65.42s	57.05s	68.48s
Model creation time	-	-	-	-	144.57s	144.39s	142.10s

d-26	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	/	51	69	85	/	/	/
Search time	/	821.26s	110.82s	/*	84.68s	78.20s	74.31s
Model creation time	-	-	-	-	183.53s	183.59s	179.94s

d-27	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	24	24	24	24	/	/	/
Search time	838.79s	103.06s	593.12s	/*	120.23s	133.44s	133.83s
Model creation time	-	-	-	-	359.85s	316.19s	314.24s

d-28	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	/	111	135	113	/	/	/
Search time	/	136.60s	196.82s	/*	77.72s	81.39s	91.01s
Model creation time	-	-	-	-	215.39s	193.55s	193.74s

d-29	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	42	42	126	96	/	/	/
Search time	585.42s	48.77s	941.57s	/*	58.52s	54.29s	52.97s
Model creation time	-	-	-	-	144.03s	129.22s	128.94s

d-30	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	80	92	114	100	/	/	/
Search time	450.19s	135.29s	38.28s	/*	57.21s	70.69s	69.18s
Model creation time	-	-	-	-	146.95s	130.72s	130.67s

B.2.7 TurnAndOpen

d-01	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	/	79	81	82	-	/	/
Search time	/	0.12s	0.02s	/*	-	77.45s	83.50s
Model creation time	-	-	-	-	-	179.39s	191.34s

d-02	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	/	96	134	106	-	/	/
Search time	/	6.41s	0.10s	/*	-	104.57s	103.94s
Model creation time	-	-	-	-	-	256.86s	241.28s

d-03	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	/	113	125	107	-	/	/
Search time	/	2.52s	0.08s	/*	-	118.59s	124.38s
Model creation time	-	-	-	-	-	290.90s	274.20s

d-04	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	/	115	138	118	-	/	/
Search time	/	0.14s	0.07s	/*	-	135.96s	155.38s
Model creation time	-	-	-	-	-	369.21s	350.47s

d-05	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	/	126	130	122	-	/	/
Search time	/	2.33s	0.10s	/*	-	155.79s	154.22s
Model creation time	-	-	-	-	-	387.68s	385.84s

d-06	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	/	151	173	163	-	/	/
Search time	/	2.01s	0.14s	/*	-	186.73s	184.79s
Model creation time	-	-	-	-	-	418.30s	425.50s

d-07	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	/	151	192	154	-	/	/
Search time	/	1.63s	0.15s	/*	-	184.28s	201.88s
Model creation time	-	-	-	-	-	458.19s	464.11s

d-08	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	/	169	192	199	-	/	/
Search time	/	4.53s	0.42s	/*	-	358.69s	391.77s
Model creation time	-	-	-	-	-	928.07s	941.88s

d-09	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	/	179	222	190	-	/	/
Search time	/	3.75s	0.66s	/*	-	391.29s	437.76s
Model creation time	-	-	-	-	-	1010.68s	1021.51s

d-10	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	/	/	423	389	-	/	/
Search time	/	/	4.99s	/*	-	1399.21s	1409.40s
Model creation time	-	-	-	-	-	3857.58s	3932.59s

d-11	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	/	354	408	350	-	/	/
Search time	/	38.26s	17.71s	/*	-	/	/
Model creation time	-	-	-	-	-	/	/

d-12	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	/	378	449	392	-	/	/
Search time	/	49.10s	9.30s	/*	-	/	/
Model creation time	-	-	-	-	-	/	/

d-13	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	/	335	/	381	-	/	/
Search time	/	201.91s	/	/*	-	/	/
Model creation time	-	-	-	-	-	/	/

d-14	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	/	386	491	430	-	/	/
Search time	/	163.38s	29.58s	/*	-	/	/
Model creation time	-	-	-	-	-	/	/

d-15	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	/	/	521	486	-	/	/
Search time	/	/	39.71s	/*	-	/	/
Model creation time	-	-	-	-	-	/	/

d-16	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	/	429	509	500	-	/	/
Search time	/	751.95s	18.77s	/*	-	/	/
Model creation time	-	-	-	-	-	/	/

d-17	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	/	411	/	420	-	/	/
Search time	/	47.15s	/	/*	-	/	/
Model creation time	-	-	-	-	-	/	/

d-18	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	/	447	/	523	-	/	/
Search time	/	1679.54s	/	/*	-	/	/
Model creation time	-	-	-	-	-	/	/

d-19	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	/	467	/	459	-	/	/
Search time	/	705.29s	/	/*	-	/	/
Model creation time	-	-	-	-	-	/	/

B.2.8 Tyreworld

d-01	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	19	19	19	19	19	/	19
Search time	0.01s	0.01s	0.01s	0.11s	2.68s	2.51s	2.14s
Model creation time	-	-	-	-	9.67s	8.37s	8.39s

d-02	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	30	36	30	30	30	/	30
Search time	0.54s	0.01s	0.01s	44.53s	6.52s	5.99s	5.87s
Model creation time	-	-	-	-	21.43s	18.66s	18.36s

d-03	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	41	53	41	41	41	/	41
Search time	76.52s	0.12s	0.01s	/*	10.50s	13.27s	10.61s
Model creation time	-	-	-	-	32.01s	28.60s	28.73s

d-04	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	/	70	52	52	52	/	52
Search time	/	1.80s	0.02s	/*	19.23s	19.57s	20.32s
Model creation time	-	-	-	-	49.30s	44.58s	44.51s

d-05	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	/	87	63	63	63	/	63
Search time	/	25.31s	0.04s	/*	33.35s	30.93s	32.83s
Model creation time	-	-	-	-	72.52s	65.11s	64.91s

d-06	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	/	104	74	74	74	/	74
Search time	/	274.55s	0.07s	/*	41.74s	45.53s	49.11s
Model creation time	-	-	-	-	100.36s	90.13s	89.58s

d-07	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	/	/	85	85	85	/	85
Search time	/	/	0.14s	/*	59.08s	59.06s	54.07s
Model creation time	-	-	-	-	132.24s	119.52s	119.90s

d-08	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	/	/	96	96	96	/	96
Search time	/	/	0.26s	/*	81.91s	82.87s	76.25s
Model creation time	-	-	-	-	174.35s	154.70s	154.76s

d-09	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	/	/	107	107	107	/	107
Search time	/	/	0.44s	/*	115.87s	110.15s	104.10s
Model creation time	-	-	-	-	217.35s	194.25s	193.91s

d-10	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	/	/	118	118	118	/	118
Search time	/	/	0.74s	/*	126.28s	133.02s	157.91s
Model creation time	-	-	-	-	267.42s	240.09s	239.61s

d-11	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	/	/	129	129	129	/	129
Search time	/	/	1.22s	/*	158.14s	163.61s	159.69s
Model creation time	-	-	-	-	326.68s	292.62s	292.81s

d-12	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	/	/	140	140	140	/	140
Search time	/	/	2.07s	/*	200.79s	193.29s	207.59s
Model creation time	-	-	-	-	394.53s	353.13s	353.47s

d-13	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	/	/	151	151	151	/	151
Search time	/	/	3.78s	/*	242.20s	253.50s	245.16s
Model creation time	-	-	-	-	466.86s	418.36s	418.20s

d-14	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	/	/	162	162	162	/	162
Search time	/	/	6.40s	/*	306.65s	288.69s	307.45s
Model creation time	-	-	-	-	559.37s	494.22s	496.71s

d-15	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	/	/	173	173	173	/	173
Search time	/	/	10.20s	/*	314.63s	359.02s	357.88s
Model creation time	-	-	-	-	646.59s	576.80s	577.01s

d-16	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	/	/	184	184	184	/	184
Search time	/	/	15.81s	/*	366.33s	485.35s	372.99s
Model creation time	-	-	-	-	760.59s	672.66s	672.18s

d-17	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	/	/	195	195	195	/	195
Search time	/	/	23.81s	/*	398.60s	463.46s	431.19s
Model creation time	-	-	-	-	870.98s	768.91s	763.63s

d-18	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	/	/	206	206	206	/	206
Search time	/	/	34.06s	/*	462.32s	544.74s	528.71s
Model creation time	-	-	-	-	1004.92s	889.39s	882.73s

d-19	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	/	/	217	217	217	/	217
Search time	/	/	48.07s	/*	589.43s	660.99s	600.88s
Model creation time	-	-	-	-	1031.48s	1009.42s	1000.82s

d-20	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Plan cost	/	/	228	228	228	/	228
Search time	/	/	66.01s	/*	657.70s	658.12s	638.14s
Model creation time	-	-	-	-	1272.28s	1147.20s	1126.11s

Bibliography

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Fahiem Bacchus and Froduald Kabanza. Using temporal logics to express search control knowledge for planning. *Artificial Intelligence*, 116(1-2):123–191, 2000.

[3] Christer Bäckström and Bernhard Nebel. Complexity results for SAS^+ planning. *Computational Intelligence*, 11(4):625–655, 1995.

[4] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473, 2014.

[5] Avrim L. Blum and Merrick L. Furst. Fast planning through planning graph analysis. *Artificial Intelligence*, 90(1-2):279–298, 1997.

[6] Christian Bohnenberger. A case study exploring the potential of deep learning in ai planning. Bachelor's thesis, Foundations of Artificial Intelligence (FAI) Group, Saarland University, 2016.

[7] Blai Bonet and Héctor Geffner. Planning as heuristic search. *Artificial Intelligence*, 129(1–2):5–33, 2001.

[8] Blai Bonet and Hector Geffner. Labeled RTDP: Improving the convergence of real-time dynamic programming. In Enrico Giunchiglia, Nicola Muscettola, and Dana Nau, editors, *Proceedings of the 13th International Conference on Automated Planning and Scheduling (ICAPS'03)*, pages 12–21, Trento, Italy, 2003. Morgan Kaufmann.

[9] Olivier Buffet and Douglas Aberdeen. The factored policy gradient planner. 173:722–747, 04 2009.

[10] Tom Bylander. The computational complexity of propositional STRIPS planning. *Artificial Intelligence*, 69(1–2):165–204, 1994.

[11] François Chollet et al. Keras. `https://keras.io`, 2015.

[12] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (ELUs). *CoRR*, abs/1511.07289, 2015.

[13] Jacques Cohen. A view of the origins and development of prolog. *Commun. ACM*, 31(1):26–36, January 1988.

[14] Rémi Coulom. Efficient selectivity and backup operators in monte-carlo tree search. In *Proceedings of the 5th International Conference on Computers and Games*, CG'06, pages 72–83, Turin, Italy, 2007.

[15] Toms de la Rosa, Sergio Jimnez, and Daniel Borrajo. Learning relational decision trees for guiding heuristic planning. In Jussi Rintanen, Bernhard Nebel, J. Christopher Beck, and Eric Hansen, editors, *Proceedings of the 18th International Conference on Automated Planning and Scheduling (ICAPS'08)*, pages 60–67. AAAI Press, 2008.

[16] Carmel Domshlak, Michael Katz, and Alexander Shleyfman. Symmetry breaking in deterministic planning as forward search: Orbit space search algorithm. February 2015.

[17] Stefan Edelkamp. Planning with pattern databases. In A. Cesta and D. Borrajo, editors, *Recent Advances in AI Planning. 6th European Conference on Planning (ECP-01)*, Lecture Notes in Artificial Intelligence, pages 13–24, Toledo, Spain, September 2001. Springer-Verlag.

[18] Patrick Ferber. Using neural networks for distance estimation in planning. Master's thesis, Foundations of Artificial Intelligence (FAI) Group, Saarland University, 2017.

[19] Richard E. Fikes and Nils Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.

[20] Maria Fox and Derek Long. The detection and exploitation of symmetry in planning problems. In M. Pollack, editor, *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI-99)*, pages 956–961, Stockholm, Sweden, August 1999. Morgan Kaufmann.

[21] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR.

[22] Eric A. Hansen and Shlomo Zilberstein. LAO: A heuristic search algorithm that finds solutions with loops. 129:35–62, 06 2001.

[23] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *SIGART Bull.*, (37):28–29, December 1972.

[24] Patrik Haslum, Adi Botea, Malte Helmert, Blai Bonet, and Sven Koenig. Domain-independent construction of pattern database heuristics for cost-optimal planning. In *Proceedings of the 22Nd National Conference on Artificial Intelligence - Volume 2*, AAAI'07, pages 1007–1012. AAAI Press, 2007.

[25] Malte Helmert. The fast downward planning system. *J. Artif. Int. Res.*, 26(1):191–246, July 2006.

[26] Malte Helmert and Carmel Domshlak. Landmarks, critical paths and abstractions: What's the difference anyway? In Alfonso Gerevini, Adele Howe, Amedeo Cesta, and Ioannis Refanidis, editors, *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS'09)*, pages 162–169. AAAI Press, 2009.

[27] Jörg Hoffmann and Bernhard Nebel. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:253–302, 2001.

[28] Shahab Jabbari Arfaee, Sandra Zilles, and Robert C. Holte. Learning heuristic functions for large state spaces. *Artif. Intell.*, 175(16-17):2075–2098, October 2011.

[29] Sergio Jiménez, Tomás de la Rosa, Susana Fernández, Fernando Fernández, and Daniel Borrajo. A review of machine learning for automated planning. *Knowledge Eng. Review*, 27(4):433–467, 2012.

[30] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. December 2014.

[31] Levente Kocsis and Csaba Szepesvári. Bandit based Monte-Carlo planning. In Johannes Fürnkranz, Tobias Scheffer, and Myra Spiliopoulou, editors, *Proceedings of the 17th European Conference on Machine Learning (ECML 2006)*, volume 4212 of *Lecture Notes in Computer Science*, pages 282–293. Springer-Verlag, 2006.

[32] Robert A. Kowalski. The early years of logic programming. *Commun. ACM*, 31(1):38–43, January 1988.

[33] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[34] Yann LeCun. Generalization and network design strategies. In R. Pfeifer, Z. Schreter, F. Fogelman, and L. Steels, editors, *Connectionism in Perspective*, Zurich, Switzerland, 1989. Elsevier. an extended version was published as a technical report of the University of Toronto.

[35] Mu Li, Tong Zhang, Yuqiang Chen, and Alexander J. Smola. Efficient mini-batch training for stochastic optimization. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, pages 661–670, New York, NY, USA, 2014. ACM.

[36] Ian Little and Sylvie Thiebaux. Probabilistic planning vs replanning. In *ICAPS Workshop on the International Planning Competition: Past, Present and Future*, 2007.

[37] Derek Long, Henry Kautz, Bart Selman, Blai Bonet, Héctor Geffner, Jana Koehler, Michael Brenner, Jörg Hoffmann, Frank Rittinger, Corin R. Anderson, Daniel S. Weld, David E. Smith, and Maria Fox. The AIPS-98 planning competition. *The AI Magazine*, 21(2):13–33, 2000.

[38] Warren McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *Brain Theory*, pages 229–230, 1986.

[39] Drew McDermott et al. *The PDDL Planning Domain Definition Language*. The AIPS-98 Planning Competition Comitee, 1998.

[40] Tomas Mikolov, Martin Karafit, Lukas Burget, Jan Cernock, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Proceedings of the 11th Annual Conference of the International Speech Communication Association, INTERSPEECH 2010*, volume 2, pages 1045–1048, 01 2010.

[41] Dana S. Nau, Tsz-Chiu Au, Okhtay Ilghami, Ugur Kuter, J. William Murdock, Dan Wu, and Fusun Yaman. Shop2: An htn planning system. *Journal of Artificial Intelligence Research*, 20:379–404, 2003.

[42] Waseem Rawat and Zenghui Wang. Deep convolutional neural networks for image classification: A comprehensive review. *Neural Comput.*, 29(9):2352–2449, September 2017.

[43] Silvia Richter and Matthias Westphal. The LAMA planner: Guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research*, 39:127–177, 2010.

[44] Silvia Richter, Matthias Westphal, and Malte Helmert. LAMA 2008 and 2011 (planner abstract). In *IPC 2011 planner abstracts*, pages 50–54, 2011.

[45] Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 64(6):386–408, 1958.

[46] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Neurocomputing: Foundations of research. chapter Learning Representations by Back-propagating Errors, pages 696–699. MIT Press, Cambridge, MA, USA, 1988.

[47] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Englewood Cliffs, NJ, 1995.

[48] Yelong Shen, Xiaodong He, Jianfeng Gao, Li Deng, and Grégoire Mesnil. Learning semantic representations using convolutional neural networks for web search. In *Proceedings of the 23rd International Conference on World Wide Web*, WWW '14 Companion, pages 373–374, New York, NY, USA, 2014. ACM.

[49] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, January 2016.

[50] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm. *ArXiv e-prints*, December 2017.

[51] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[52] Neal Stephenson. *Snow Crash*. Bantam Books, 1992.

[53] Sam Toyer. Generalised policies for probabilistic planning with deep learning. Research and development, honours thesis, Research School of Computer Science, Australian National University, October 2017.

[54] Sam Toyer, Felipe W. Trevizan, Sylvie Thiébaux, and Lexing Xie. Action schema networks: Generalised policies with deep learning. *CoRR*, abs/1709.04271, 2017.

[55] Felipe Trevizan and Manuela M. Veloso. Short-sighted stochastic shortest path problems. 216, 11 2014.

[56] Håkan L. S. Younes, Michael L. Littman, David Weissman, and John Asmuth. The first probabilistic track of the international planning competition. *jair*, 24:851–887, 2005.