UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

RAFAEL VALES BETTKER

Understanding Sample Generation Strategies for Learning Heuristic Functions in Classical Planning

Thesis presented in partial fulfillment of the requirements for the degree of Master of Computer Science

Advisor: Prof. Dr. André Grahl Pereira

Porto Alegre
June 2023

CIP — CATALOGING-IN-PUBLICATION

Bettker, Rafael Vales

Understanding Sample Generation Strategies for Learning Heuristic Functions in Classical Planning / Rafael Vales Bettker. – Porto Alegre: PPGC da UFRGS, 2023.

65 f.: il.

Thesis (Master) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2023. Advisor: André Grahl Pereira.

I. Pereira, André Grahl. II. Título.

To my mother.

"God does not play dice"

— ALBERT EINSTEIN

ACKNOWLEDGMENTS

First of all, I would like to thank my supervisors, Prof. Dr. André Grahl Pereira and Prof. Dr. Marcus Ritt, for their guidance during the past two years. Their expertise and continuous feedback were invaluable to my research and significantly contributed to my growth as an academic and professional. I would like to extend my gratitude to my research partner, Pedro Probst Minini, with whom I shared every moment of this master's degree. Without his contribution in our day-to-day work and countless discussions, everything would have been more arduous.

I would like to express my sincere gratitude to my mother, Fátima Antunes Vales, for her love and support throughout my entire life. I would not have reached this point without her unwavering dedication. I would also like to thank my father, Roberto dos Reis Bettker, who I know continues to cheer me on from wherever he may be. Lastly, I am grateful to those in my family who have closely followed my academic journey and offered their support and well wishes.

ABSTRACT

Heuristic functions are essential in guiding search algorithms to solve planning tasks. We study the problem of learning good heuristic functions for classical planning tasks with neural networks based on samples that are states with their cost-to-goal estimates. It is well known that the learned model quality depends on the quality of training data. Our main goal is to better understand the influence of sample generation strategies on the performance of a greedy best-first search guided by a learned heuristic function. In a set of controlled experiments, we find that two main factors determine the quality of the learned heuristic: the distribution of samples in the state space and the quality of the cost-to-goal estimates. These two factors are interdependent: having perfect cost-to-goal estimates is insufficient if the samples are not well distributed across the state space. We study the effects of restricting samples to only include states that could be visited when solving a given task and the effects of adding samples with high-value estimates. Based on our findings, we propose practical strategies to improve the quality of learned heuristics: three strategies that aim to generate more representative states and two strategies that improve the cost-to-goal estimates. Our resulting neural network heuristic has higher coverage than a basic satisficing heuristic. Compared to a baseline learned heuristic, our best neural network heuristic almost doubles the mean coverage and can increase it for some domains by more than six times.

Keywords: Classical Planning. Heuristic Search. Machine Learning. Sample Quality.

Compreendendo Estratégias de Amostragem para Aprendizagem de Funções Heurísticas em Planejamento Clássico

RESUMO

Funções heurísticas são essenciais para guiar algoritmos de busca na resolução de tarefas de planejamento. Nós estudamos o problema de aprender boas funções heurísticas para tarefas de planejamento clássico usando redes neurais baseadas em amostras que são estados acompanhados de suas estimativas de custo-para-objetivo. É conhecido que a qualidade do modelo aprendido depende da qualidade dos dados de treinamento. Nosso objetivo principal é entender melhor a influência das estratégias de geração de amostras no desempenho do *greedy best-first search* guiado por uma função heurística aprendida. Em um conjunto de experimentos controlados, descobrimos que dois fatores principais determinam a qualidade da heurística aprendida: a distribuição de amostras no espaço de estados e a qualidade das estimativas de custo-para-objetivo. Esses dois fatores são interdependentes: ter estimativas ótimas de custo-para-objetivo é insuficiente se as amostras não estiverem bem distribuídas ao longo do espaço de estados. Nós estudamos os efeitos de restringir as amostras para incluir apenas estados que poderiam ser visitados ao resolver uma determinada tarefa e os efeitos de adicionar amostras com altos valores de estimativas. Com base em nossas descobertas, propomos estratégias práticas para melhorar a qualidade das heurísticas aprendidas: três estratégias que visam gerar estados mais representativos e duas estratégias que melhoram as estimativas de custo-para-objetivo. Nossa heurística resultante da rede neural possui uma cobertura maior do que uma heurística de *satisficing* básica. Em comparação com uma heurística *baseline* aprendida, nossa melhor heurística de rede neural quase dobra a cobertura média e aumenta para alguns domínios em mais de seis vezes.

Palavras-chave: Planejamento Clássico. Busca Heurística. Aprendizado de Máquina. Qualidade das Amostras.

LIST OF ABBREVIATIONS AND ACRONYMS

ASNet	Action Schema Network
BFS	Breadth-First Search
BSS	Backward State Space
DFS	Depth-First Search
FF	Fast-Forward
FNN	Feedforward Neural Network
FSM	Focused Sampling Method
FSS	Forward State Space
GBFS	Greedy Best-First Search
HGN	Hypergraph Network
IPC	International Planning Competition
MSE	Mean Squared Error
NN	Neural Network
PDDL	Planning Domain Description Language
ReLU	Rectified Linear Unit
ResNet	Residual Neural Network
RW	Random Walk
SAI	Sample Improvement
SAS^+	Simplified Action Structures Plus
STRIPS	Stanford Research Institute Problem Solver
SUI	Successor Improvement

LIST OF SYMBOLS

$\mathcal{V}$	Set of variables
$\mathcal{F}$	Set of facts
$\mathcal{O}$	Set of operators
$\mathcal{S}$	State space
s_0	Initial state
s^*	Goal condition
s'	Successor state
$\perp$	Undefined value
Π	Planning task
π	Plan
d^*	Distance from state furthest from goal
L	Maximum regression limit
F	Facts regression limit
$\bar{F}$	Facts per mean effects regression limit
h	Heuristic function
h^*	Perfect heuristic function
h^{FF}	FF heuristic
h^{GC}	Goal-count heuristic
$\hat{h}$	Learned heuristic function
$\hat{h}_0$	Baseline learned heuristic function
$\hat{h}'$	Logic-independent learned heuristic function

LIST OF FIGURES

LIST OF TABLES

CONTENTS

1 INTRODUCTION

Classical planning provides a method for representing and solving various problems. Formulating these problems as planning tasks can model real-world challenges such as route planning, robotics, automated system verification, and computational biology (EDELKAMP; SCHRÖDL, 2012). This approach enables automated systems to reason, make decisions, and generate plans to achieve specific objectives. The ability to represent and solve problems using classical planning techniques has garnered significant attention and has proven instrumental in various domains.

Planning tasks are typically defined by the initial state and the desired outcome (goal state). States capture the relevant information about the system's condition at a particular time. Each domain provides a set of actions that describe how a state can be transformed. A plan is constructed to reach the goal from the initial state by applying a sequence of actions. The plan provides a step-by-step guide for an agent or a system to follow to achieve the desired objective. By using various planning techniques and algorithms, planning systems can efficiently explore the space and generate plans to solve complex problems, including those classified as PSPACE-complete (BYLANDER, 1994).

Several approaches can find a sequence of actions that transforms an initial state into one that satisfies the goal condition. One successful strategy for solving such tasks is to apply algorithms of the best-first search family, guided by a heuristic function that estimates the cost to reach a goal for each state. Generally, best-first search algorithms are more effective when the heuristic function better estimates the perfect cost-to-goal.

Relaxations of planning tasks create some of the most successful heuristic functions, e.g., the delete relaxation (HOFFMANN; NEBEL, 2001), critical paths (HASLUM; GEFFNER, 2004), landmarks (KARPAS; DOMSHLAK, 2009; HOFFMANN; PORTEOUS; SEBASTIA, 2004), or the state equation (BONET, 2013). Many of these heuristics come with additional properties, such as admissibility.

The emergence of interest in learning heuristic functions with neural networks has been driven by rapid progress in other application areas. Some works in this area include those by Samadi, Felner and Schaeffer (2008), Arfaee, Zilles and Holte (2011), Agostinelli et al. (2019), Yu, Kuroiwa and Fukunaga (2020), Shen, Trevizan and Thiebaux (2020), Ferber, Helmert and Hoffmann (2020), Toyer et al. (2020), Ferber et al. (2022), and O'Toole et al. (2022). The basic approach is simple: one generates a set of samples of pairs of states and estimates of cost-to-goal and trains a supervised model over the set of samples. However, a successful approach to planning has to solve several challenges:

C1) State spaces are implicitly defined and mostly exponential in the size of a compact description. Therefore, random samples are hard to generate and may be infeasible, unreachable from an initial state, or unable to reach the goal. Samples are usually generated by expanding the state space through forward search or backward search (regression).

C2) Estimates of cost-to-goal are typically hard to obtain. Finding the perfect cost-to-goal amounts to solving the task on the samples, and often the learned function is only useful if it is close to the perfect cost-to-goal.

C3) Planning domains are very different, and traditional heuristics apply to any domain. This results in the problem of transferring a learned heuristic to new domains, tasks, or state spaces.

C4) Planners depend on evaluating many states per second, so computing the heuristic function should be fast, or the learned heuristic must be more informed. However, there is a trade-off between a more informed learned heuristic and the complexity of the model.

By addressing these core issues, new possibilities can be found for advancing the field of classical planning and enabling more efficient and effective problem-solving capabilities in planning systems. For this, we investigate current state sampling methods and propose new techniques that directly affect the quality of samples for training heuristic functions.

1.1 Contributions

Through controlled experiments on planning tasks with small state spaces, we identify several techniques that improve the quality of the samples used for training. Our contributions include:

- A sample generation algorithm that can better generate a representative subset of the state space through a combination of breadth-first search (expanding states close to the goal) followed by random walks from the breadth-first search's leaves (Section 3.1.1).
- State space-based estimations to limit the sampling regression depth to avoid large cost-to-goal overestimates (Section 3.1.2).

- Two methods to improve cost-to-goal estimates based on detecting samples from the same or neighboring states (Section 3.2).
- A systematic study on sampling quality (Chapter 4).

1.2 Overview and Outline

This thesis explores strategies for generating samples and their influence on heuristic function performance. In Chapter 2, we provide the necessary background for our work, covering important topics such as classical planning (Section 2.1), search algorithms (Section 2.2), and neural networks (Section 2.3). Additionally, we review relevant previous research and related work in Section 2.4. By establishing this, we present the sample generation techniques and propose new approaches to improve them in Chapter 3. We focus mainly on the quality of the learned heuristic and its influence on the number of expanded states and coverage. To this end, Chapter 4 presents a systematic study of the contributions of each strategy when solving distinct initial states of a single state space, aiming to understand better how to learn high-quality heuristics. In Section 4.1, we present our settings to learn state space-specific heuristics using a feedforward neural network. In experiments on small state spaces (Section 4.2), we investigate the effect of different sampling strategies, the quality of the learned heuristic with an increasing number of samples, and the effect of a different subset of states part of the sample set on the learned heuristic. We also evaluate how the quality of the estimates of cost-to-goal influences the effectiveness of learned heuristic to guide a search algorithm. Then, in Section 4.3, we compare our best techniques with a baseline and traditional heuristics over large state spaces. Furthermore, we qualitatively compare existing methods in Section 4.4 and discuss the limitations of learned heuristics in Section 4.5. Finally, we conclude and highlight possible future works in Chapter 5.

2 BACKGROUND

This chapter provides an overview of the fundamental concepts and techniques that form the background for this thesis.

2.1 Classical Planning

Typically we provide a formal description of the problem to find a solution from search algorithms. In classical planning, a problem is represented as a planning task. We define the various concepts that constitute a planning task, which are addressed throughout the thesis.

Definition 2.1 (Fact)**.** *A fact f is a statement that represents a condition or characteristic of the environment and can be either true or false.*

Typically, a domain has a set of facts denoted as $\mathcal{F} = \{f_1, \ldots, f_n\}$. For example, consider the domain VisitAll, where a robot explores a grid and aims to visit all its cells. In this scenario, facts can represent the robot's position and the status of each cell, indicating whether it has been visited or not. Consider a grid with two cells labeled a and b. In this case, the set of facts can be represented as $\mathcal{F} = \{\text{at-robot}(a), \text{at-robot}(b), \text{visited}(a), \text{visited}(b)\}$.

Definition 2.2 (Mutex)**.** *A mutex (mutual exclusion) is a condition where two or more facts cannot occur simultaneously.*

In VisitAll, the robot cannot be in two positions simultaneously. Therefore, the set of facts $\{\text{at-robot}(a), \text{at-robot}(b)\}$ represents a mutex.

Definition 2.3 (Variable)**.** *A variable v is a concise representation of a condition or characteristic of the environment. It can assume any value from a predefined domain $D(v)$.*

A variable groups a collection of mutually exclusive facts, where each fact corresponds to a value $d \in D(v)$ that can be assigned to the variable v at any given moment, allowing for a single assignment at a time. (Note that a mutex can also arise between values of different variables.) For instance, in the VisitAll domain, all the facts representing the robot's position can be combined into a variable. This approach significantly reduces the size required to represent the robot's position. Instead of necessitating n facts in a grid with n cells, we have a single variable v_p with $|D(v_p)| = n$. Similarly to facts, each domain task has a set of variables $\mathcal{V} = \{v_1, \ldots, v_n\}$.

Definition 2.4 (State)**.** *A state s is an assignment of all variables $v \in \mathcal{V}$.*

A state where all variables are defined is also called a complete state. When the set of variables is not fully assigned, i.e., one or more variables $v \in \mathcal{V}$ do not have a defined value $d \in D(v)$, it is a partial state. Let $s(v)$ be the value of variable v in state s. The value of an undefined variable v is written as $s(v) = \bot$. We say that $s \subseteq t$ if $s(v) = t(v)$ for all $v \in \mathcal{V}$ such that $t(v) \neq \bot$. This implies that there is an assignment of undefined variables such that $s = t$. Therefore, a partial state t represents a set of states containing every state whose $s \subseteq t$. The initial state s_0 is a complete state corresponding to the initial variable assignment. The goal s^* can be defined as a partial state.

Definition 2.5 (Operator)**.** *An operator o, also known as action, is defined as a pair of preconditions and effects $(\mathrm{pre}(o), \mathrm{eff}(o))$, both partial states. The preconditions specify the conditions that must hold in the current state for an operator to be applicable, while the effects describe the changes in the state that occur when the operator is applied.*

Let $\mathcal{O}$ be the set of all operators in a given task. An operator $o \in \mathcal{O}$ is applicable to a state s if $s \subseteq \mathrm{pre}(o)$, and produces a successor state $s' = \mathrm{succ}(s, o) := \mathrm{eff}(o) \circ s$, where $s' = t \circ s$ is defined by $s'(v) = t(v)$ for all v such that $t(v)$ is defined, and $s'(v) = s(v)$ otherwise. The set of all successor states of state s is $\mathrm{succ}(s) = \{\mathrm{succ}(s, o) \mid o \in \mathcal{O}, s \subseteq \mathrm{pre}(o)\}$. Each operator is assigned a cost according to the mapping function $\mathit{cost} : \mathcal{O} \to \mathbb{R}_+$; when the cost is omitted, it assumes a unit cost, i.e., $\mathit{cost}(o) = 1$ for each operator $o \in \mathcal{O}$.

A sequential application of operators is called a progression. Alternatively, a regression is a backward sequential application of operators. For regression, we construct a backward operator $o_r \in \mathcal{O}_r$ for each operator $o \in \mathcal{O}$. Given an operator $o = (\mathrm{pre}(o), \mathrm{eff}(o))$, we generate the corresponding backward operator $o_r = (\mathrm{eff}(o), \mathrm{pre}(o))$ if $\mathrm{pre}(o)$ is defined, otherwise $o_r = (\mathrm{eff}(o), \bot)$.

Similar to progression, a partial state s has predecessors $\mathrm{pred}(s) = \{\mathrm{pred}(s, o_r) \mid o_r \in \mathcal{O}_r, s \subseteq \mathrm{pre}(o_r)\}$. A regression sequence from state s_0 then is valid if o_i can be applied to s_{i-1} and produces $s_i = \mathrm{pred}(s_{i-1}, o_i)$. All partial states s_k can reach a partial state $s_0 \subseteq s$ in at most k forward applications of the reversed operator sequence.

Definition 2.6 (Plan)**.** *A plan is a sequence of operators $\pi = (o_1, \ldots, o_k)$.*

A plan is valid for state s_0, referred to as an s_0-plan, if for $i \in [k]$ operator o_i can be applied to s_{i-1} and produces $s_i = \mathrm{succ}(s_{i-1}, o_i)$, where $s_k \subseteq s^*$. The cost of plan π is $\sum_{o \in \pi} \text{cost}(o)$. When a plan has the lowest cost among all s_0-plans, it is called an optimal s_0-plan.

Definition 2.7 (State Space)**.** *A state space is the set of all the states over the variables $v \in \mathcal{V}$.*

The forward state space (FSS) is the set of all the states reachable from the initial state s_0 by applying a sequence of operators. Similarly, the backward state space (BSS) is the set of all the partial states reachable from a goal s^* by applying a sequence of backward operators.

2.1.1 STRIPS Representation

The STRIPS (Stanford Research Institute Problem Solver) representation (FIKES; NILSSON, 1971) is a fundamental approach used in planning tasks to model and reason about the state of the environment. In this representation, a planning task is defined by a set of propositions (facts) that describe the various attributes and conditions of the problem domain. Each fact represents a specific property that can be true or false in a given state. The preconditions and effects of operators are expressed as a set of facts.

Definition 2.8 (STRIPS Planning Task)**.** *A STRIPS planning task is defined as a tuple $\Pi = \langle \mathcal{F}, \mathcal{O}, s_0, s^*, cost \rangle$, where $\mathcal{F}$ is a set of facts, $\mathcal{O}$ is a set of operators over $\mathcal{F}$, s_0 an initial state, s^* the goal condition, and cost $: \mathcal{O} \to \mathbb{R}_+$ a function mapping operators to costs.*

Throughout this thesis, we present samples represented in the STRIPS formalism. The motivation behind this approach lies in the compatibility between the propositional nature of STRIPS, where facts can be either true or false, and the proposed neural network input in binary format, which is more suitable for training (Section 2.3.2). Additionally, the Fast Downward planning system used in this thesis has PDDL (Planning Domain Definition Language) as its input language. PDDL provides a formal and widely accepted syntax for describing planning tasks compatible with the STRIPS representation.

Figure 2.1 shows an example of a domain description in PDDL. The VisitAll domain represents a scenario where a robot navigates a grid and marks each cell it steps on as visited. The objective of this domain is typically to visit all cells in the grid. The initial and goal states are described in a second file, the problem PDDL, along with the object declarations. The domain PDDL specifies the predicates and actions (operators).

Figure 2.1 – VisitAll domain description in PDDL.

```
(define (domain grid-visit-all)
    (:requirements :typing)
    (:types place - object)
    (:predicates (connected ?x ?y - place)
                 (at-robot ?x - place)
                 (visited ?x - place))
    (:action move
        :parameters (?curpos ?nextpos - place)
        :precondition (and
            (at-robot ?curpos)
            (connected ?curpos ?nextpos))
        :effect (and
            (at-robot ?nextpos)
            (not (at-robot ?curpos))
            (visited ?nextpos)))
)
```

Source: International Planning Competition (IPC) 2014.

In the VisitAll example, we have one object type (place), three predicates (connected, at-robot, and visited), and one action (move). The combination of predicates with objects forms the facts in a process called grounding. For instance, in a grid with two connected cells, a and b, we have the set of facts $\mathcal{F} = \{$connected(a, b), connected(b, a), at-robot(a), at-robot(b), visited(a), visited$(b)\}$.

2.1.2 SAS$^+$ Representation

Another approach to modeling planning tasks is using the SAS$^+$ representation (BÄCKSTRÖM; NEBEL, 1995). Although Fast Downward takes propositional representation with PDDL as its input, internally, it uses finite domains to represent states. The SAS$^+$ enhances the capabilities of a propositional representation by using finite domains D, which explicitly define the possible values for each variable. This representation enables a more compact and structured representation of the problem.

Definition 2.9 (SAS$^+$ Planning Task). *A SAS$^+$ planning task is defined as a tuple $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s^*, cost\rangle$, where $\mathcal{V}$ is a set of variables, $\mathcal{O}$ is a set of operators over $\mathcal{V}$, s_0 an initial state, s^* the goal condition, and cost : $\mathcal{O} \rightarrow \mathbb{R}_+$ a function mapping operators to costs.*

SAS$^+$ and STRIPS differ in their method of representing states and can be converted between each other. For example, consider the VisitAll domain depicted in Figure 2.1, specifically the at-robot predicate. In PDDL, the action always replaces one robot position with another. Thus, SAS$^+$ represents this predicate using a variable v and a value $d \in D(v)$ for each possible robot position. Consequently, in large grids with hundreds of positions where propositional representation would require many facts, SAS$^+$ efficiently represents them using a single variable.

2.2 Search Algorithms

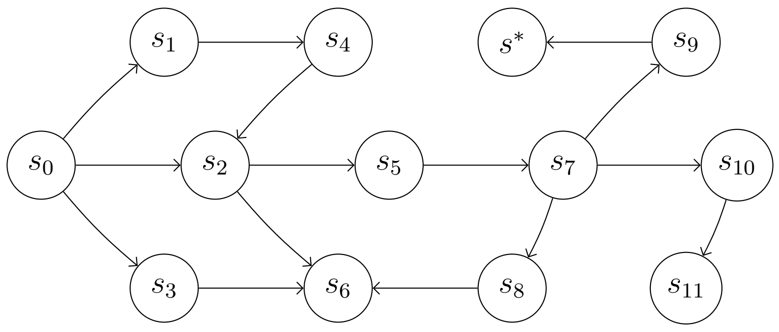
Search algorithms enable expanding the state space to find a solution for a planning task. They systematically traverse the state space of a problem, aiming to reach a goal state from an initial state. Blind search algorithms, in particular, operate without any knowledge about the problem domain and rely solely on the problem representation. On the other hand, heuristic search uses heuristic functions generated from problem-specific knowledge. Heuristics prioritize the expansion of more promising states by using informed estimates of the cost or distance to the goal. The following sections present both approaches.

2.2.1 Blind Search

By expanding the state space exhaustively, blind search algorithms navigate across various states and paths to find potential solutions. An example of a blind search algorithm is the Breadth-First Search (BFS). BFS is a graph search and expands the state space by systematically expanding all the states at the same distance from the initial state before moving to the next distance. This strategy ensures that the shortest path to a goal state is found. However, BFS can be memory-intensive as it maintains a queue of all the generated but not expanded states in memory.

Depth-First Search (DFS) also performs a blind search. In DFS, the search starts from an initial state and expands the state space by iteratively expanding the farthest state from the initial state that has not been expanded yet. This algorithm is memory-efficient as it only needs to keep track of a single path from the initial state to the current state. However, DFS does not guarantee an optimal solution and can get trapped in deep branches of the state space.

Figure 2.2 – Graph representing a state space where the vertices and arcs correspond to states and applicable operators, respectively.



To illustrate, Figure 2.2 presents a state space. BFS starts by expanding the initial state s_0 and then expands s_1, s_2, and s_3. In sequence, it expands the successor of s_1 (s_4), then of s_2 (s_5 and s_6), and so on until a solution is found. On the other hand, DFS expands the initial state s_0 and continues expanding a newly generated successor until it reaches a solution or a state without successors (e.g., s_6 or s_{11}), at which point the algorithm backtracks to the nearest state that has an unexpanded successor and continues the search.

BFS and DFS are unsuitable for solving planning tasks due to their lack of efficiency in expanding the state space, which is typically vast and contains numerous paths. Instead, heuristic search is used, which will be introduced in the next section. However, BFS and DFS can still be valuable in planning as sampling algorithms. By sampling the state space using these algorithms, we can selectively focus on regions either closer to the initial state, using BFS, or more distant, using DFS.

2.2.2 Heuristic Search

Expanding the state space systematically to find a plan can quickly become computationally infeasible for planning domains with large state spaces. Therefore, the primary approach for solving planning tasks is heuristic search. Heuristic search algorithms address this challenge by using heuristics to guide the search toward promising regions of the state space. These heuristics estimate how close a given state is to a goal state and guide which successors to prioritize during the search.

A commonly used heuristic search algorithm in classical planning is A* search (HART; NILSSON; RAPHAEL, 1968). For a given state s, A* is guided by $f(s) = g(s) + h(s)$, i.e., combine the current cost $g(s)$ of the sequence of operators starting in the initial state s_0 and resulting in s, and a heuristic function $h(s)$ that estimates the cost-to-goal from state s. By considering both the past cost and the estimated future cost, A* can efficiently expand the state space to find an optimal solution – provided that the heuristic is admissible (does not overestimate the actual h-value).

Greedy Best-First Search (GBFS) (DORAN; MICHIE, 1966), another widely used heuristic search algorithm in classical planning, is an alternative to A* search that prioritizes expanding states with the lowest heuristic values without considering the cost of reaching those states, i.e., $f(s) = h(s)$. Both A* and GBFS are considered a "greedy" algorithm because it only considers the heuristic estimate and makes decisions solely based on that information. While GBFS can be highly efficient regarding expansion speed, it does not guarantee to find an optimal solution as A* does. Nonetheless, GBFS is a popular choice in planning, where quickly finding any feasible solution is more important than finding the optimal solution. Therefore, in our experiments, GBFS serves as the search algorithm.

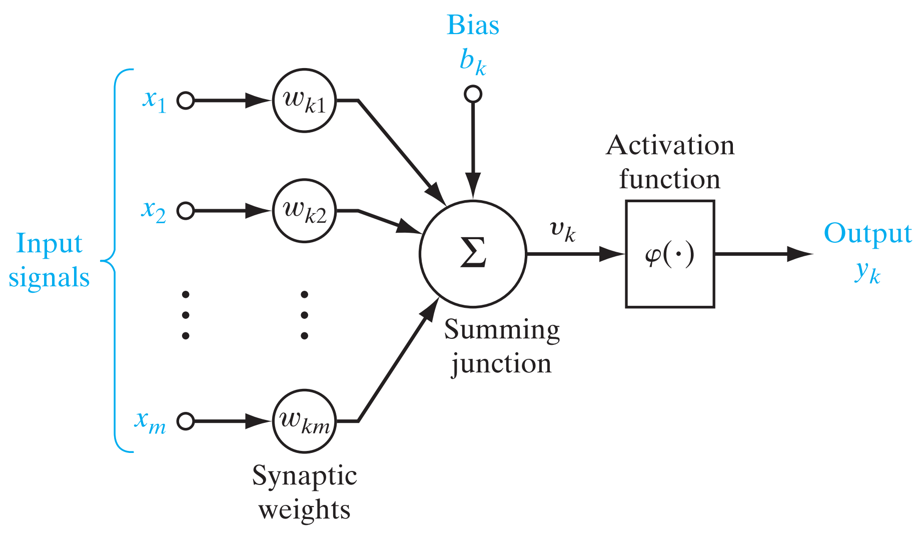
Heuristic search algorithms require the use of a heuristic function. A heuristic function $h : \mathcal{S} \to \mathbb{R}_+ \cup \{\infty\}$ maps each state in the state space $\mathcal{S}$ to a non-negative number or ∞. The number represents the cost-to-goal estimate (h-value) for the given state. An infinity value indicates a dead-end state, i.e., without any paths leading to the goal. The perfect heuristic h^* produces the cost of an optimal s-plan for each state $s \in \mathcal{S}$.

The heuristic function can have certain properties, such as admissibility, consistency, and goal-awareness. An admissible heuristic function never overestimates the actual goal distance. It guarantees optimality when combined with specific search algorithms such as A*. A heuristic function is consistent if, for every state $s \in \mathcal{S}$ and for every applicable operator o with $s' = \mathrm{succ}(s, o)$, it holds that $h(s) \leq cost(o) + h(s')$. Finally, a goal-aware heuristic guarantees $h(s) = 0$ for all goal states $s \subseteq s^*$.

Heuristics can be arbitrary functions, allowing for design flexibility based on domain knowledge and problem-specific insights. Therefore, a heuristic can be classified as logic-independent or logic-dependent. In a logic-independent setting, we interact with the planning task only by functions that allow accessing the initial state s_0, the goal condition s^*, and the successors $\mathrm{succ}(s)$ or predecessors $\mathrm{pred}(s)$ of a state s through black-box functions (STURTEVANT; HELMERT, 2019) – which could also be learned. We do not have access to the logical description of operators. In contrast, a logic-dependent heuristic can use the complete description of the model, which permits, for example, reasoning about operators and the computation of mutexes.

A logic-dependent heuristic is the FF (Fast-Forward) heuristic (HOFFMANN;

Figure 2.3 – Structure of the artificial neuron.



Source: Haykin (2009)

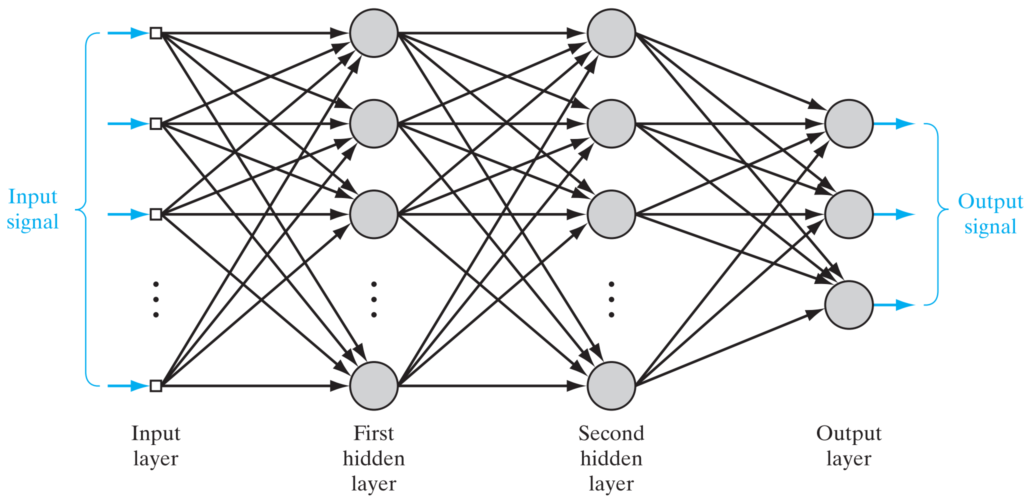
NEBEL, 2001), which computes its cost-to-goal estimates by considering a relaxed version of the planning problem. In a relaxed version, all effects that remove facts from a state (delete effects) of the operators are ignored, resulting in a simplified version of the problem. The FF heuristic extracts a solution from a relaxed version and computes the cost of operators in the plan, using it as the cost-to-goal estimate. As logic-independent, we have the goal-count heuristic, which does not rely on the planning model. Instead, it counts the number of unsatisfied goal conditions in the current state, assuming that each unsatisfied condition requires an additional unit cost operator to be satisfied. Both the FF and goal-count heuristics are addressed in our experiments (Chapter 4).

2.3 Neural Networks

Neural networks (NN) have gained popularity due to their ability to learn and generalize from datasets. In classical planning, various neural networks approaches have been proposed (see Section 2.4). These approaches range from simpler models that prioritize processing speed – and, consequently, expansion rate – to more complex architectures that exploit the relational structure of a domain.

Essentially, neural networks are composed of multiple artificial neurons, whose structure is illustrated in Figure 2.3. A neuron k consists of four components: input signals $x_1, \ldots, x_m$, weights $w_{k1}, \ldots, w_{km}$, bias term b_k, and an activation function φ. A neuron k produces an output $y_k = \varphi(v_k + b_k)$ where $v_k = \sum_{j\in[m]} w_{kj}x_j$. The activation

Figure 2.4 – Graph of a neural network. Vertices and edges represent the neurons and their connections, respectively.



Source: Haykin (2009)

function introduces non-linearity to the output. A common example is the Rectified Linear Unit (ReLU) activation $\varphi(v) = \max(0, v)$.

The NN comprises multiple layers of neurons, as illustrated in Figure 2.4, which transform the input data into an output. The input layer receives the input data and passes it to the subsequent layers. The hidden layers, located between the input and output layers, weigh the neuron weights to learn the features that transform the input data into the desired output. These hidden layers enable the network to capture complex patterns and relationships within the data. Finally, the output layer produces the prediction for the given input. A specific type of NN called Feedforward Neural Network (FNN) allows information to flow in one direction, from the input layer through the hidden layers to the output layer.

The learning process involves adjusting the network's parameters, such as weights and biases, to minimize the error between its predictions and the true values of the training samples. This adjustment is typically performed using an optimization algorithm to minimize a specified loss function. A commonly used loss function is the Mean Squared Error (MSE), which quantifies the error by computing the mean squared difference between the predicted and true values. Training is often performed on batches of data rather than individual samples to update the parameters efficiently. The batch size determines the number of samples processed together before updating the network's parameters. Batches enable training with larger sets of samples, which can be computationally infeasible to handle all at once. Haykin (2009) provides further details for a more comprehensive understanding of weight adjustment and training in neural networks.

By iteratively adjusting its parameters based on the training data, an NN gradually improves its ability to make accurate predictions. The quality of the NN depends on the quality of the sample set. Additionally, the network's capacity to learn and generalize from the training data is influenced by its architecture. Each one has its purpose and may excel in specific types of tasks.

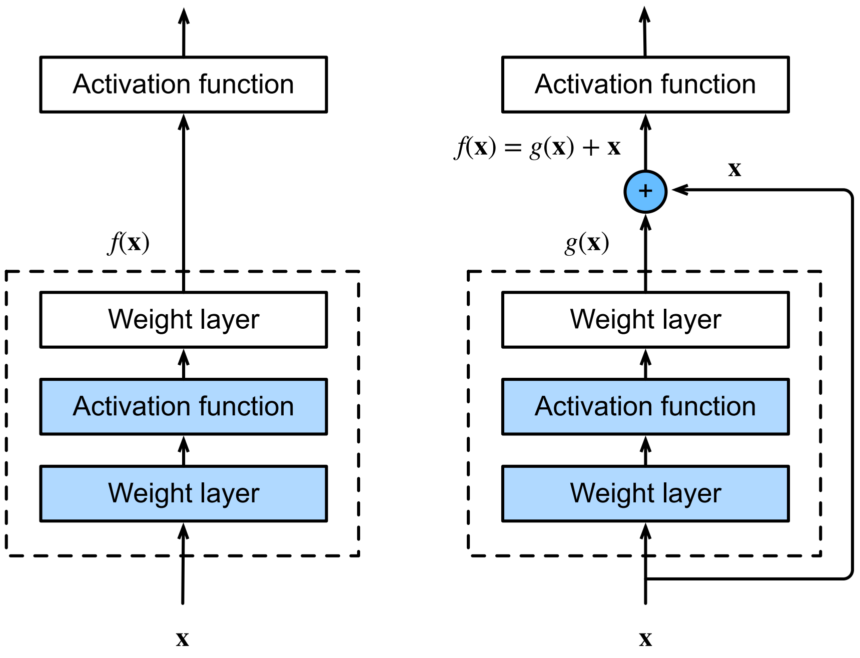
2.3.1 Residual Networks

ResNet, short for Residual Network, was proposed by He et al. (2016) and has gained attention due to its ability to address the vanishing gradient problem (HOCHREITER, 1998) and improve training performance for deep networks. While its initial success was in image recognition, ResNets has also been used in planning (AGOSTINELLI et al., 2019; FERBER et al., 2022) to address training deep neural networks.

ResNet uses the concept of residual connections or skip connections, which enable the network to learn residual mappings. These connections allow the network to bypass specific layers and pass the input directly to deeper layers. By doing so, ResNet mitigates the vanishing gradient problem in deep networks, where the gradient becomes increasingly small as they propagate backward through the network layers, causing the weights in the earlier layers to update slowly or not at all. The residual connections establish "highways" for information flow, helping maintain the signal propagation.

Figure 2.5 illustrates a schematic representation of both a regular block and a residual block – with a shortcut connection that skips two layers – in a ResNet architecture. In a regular block, the output of its layers is directly mapped to the activation function. In a residual block, the network needs to learn the residual mapping. Instead of forcing the network to learn all the information from the input, residual blocks allow it to focus only on the residual, i.e., the difference between the input and output, needed to achieve the desired output. He et al. (2016) provide a more detailed explanation of ResNets.

Figure 2.5 – A regular block (left) and a residual block (right).



Source: Zhang et al. (2021)

2.3.2 Learning Heuristic Functions

Many heuristics for classical planning are derived from a model of the task, such as SAS$^+$. An obvious alternative is to learn to map a state s to its heuristic value $h(s)$. We focus on learning with neural networks, although other supervised learning methods could be used. To learn a heuristic function, an NN is trained on pairs of states and cost-to-goal estimates. The learned heuristic functions are usually not admissible, so traditional optimality guarantees are lost.

A propositional representation of a state is more suitable for learning functions over states, as the variables in a planning task are categorical variables. To this end, consider a SAS$^+$ planning task $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s^*, \text{cost} \rangle$, and let $\mathcal{V} = \{v_1, \ldots, v_n\}$ and $D(v_i) = \{d_{i1}, \ldots, d_{i,s_i}\}$, $i \in [n]$ be some order of the variables and their domains. We represent any state s by a sequence of facts

$$\mathcal{F}(s) = (f_{11}, f_{12}, \ldots, f_{1,s_1}, \ldots, f_{n1}, f_{n2}, \ldots, f_{n,s_n}),$$

where each fact $f_{ij} = [s(v_i) = d_{ij}]$ indicates if variable v_i assumes value d_{ij} in state s. Note that facts $\mathcal{F}_i = \{f_{i1}, \ldots, f_{i,s_i}\}$ corresponding to variable v_i satisfy the consistency condition $\sum_{f \in \mathcal{F}_i} f \leq 1$ since each variable assumes at most one value, and $\sum_{f \in \mathcal{F}_i} f = 0$ only if v_i is undefined. More generally, for any set of facts $\mathcal{F}$ we write $\text{mutex}(\mathcal{F})$ if

$\sum_{f \in \mathcal{F}}[f] \leq 1$ must be satisfied in states of Π. Many planning systems can deduce mutexes from the description of the planning task Π (HELMERT, 2009); we will discuss and analyze their utility for sampling states later. Some architectures provide additional input to the neural network, e.g., the propositional representation of the goal condition. The target output for training may be the cost-to-goal estimates directly or some encoding of them.

An important aspect of sample generation related to challenges C1 and C2 (Chapter 1) is the degree of dependency on the domain model or the planning task and the cost of generating the samples. The cost of sample generation depends on the number of samples and the cost to generate each. This generates the problem of deciding how many samples are required since, generally, only a very small part of the state space can be sampled. More importantly, ideal samples would be labeled with the perfect heuristic h^*. In general, ideal labeling is impractical since it requires solving the planning task on a large number of initial states. Therefore, we are mainly interested in good heuristic estimates that can be generated fast. We analyze the influence of sample size and quality experimentally later.

Additionally, and related to challenges C3 and C4, network architecture and sample generation depend on the range of tasks the learner intends to generalize. The range may be the state space of a planning task, a planning domain, or an entire planning formalism. In the first case, the set of planning tasks is defined over any pair of initial state s_0 and goal s^*. Often the set of planning tasks is restricted to select the initial state from the FSS of some given initial state and to a fixed goal. In the second case, the learned function has to generalize over all domain tasks. Finally, a learning-based heuristic that generalizes over a planning formalism is domain-independent. An important aspect of sample generation is the distribution of states which are part of the sample set. For example, the sample set can contain only states with a short distance to the goal or only states with a short distance to the initial states. We assume that the distribution of h^*-values, e.g., in a histogram, represents the distribution of states in the sample set.

2.4 Related Work

Two main research topics have been explored in learning heuristic functions: structured NN-based approaches and non-structured NN-based approaches. The structured NN-based approaches utilize model descriptions, such as information about operator preconditions and effects, to construct a neural network. These approaches aim to generalize across different domains or planning formalisms. On the other hand, non-structured NN-based approaches focus on sampling, using limited access to the model to identify mutexes and generate states that are close to those encountered during the search, aiming to generalize only over a state space.

2.4.1 Structured NN-based Approaches

The first set of approaches (TOYER et al., 2018; TOYER et al., 2020; SHEN; TREVIZAN; THIEBAUX, 2020; STÅHLBERG; BONET; GEFFNER, 2022; GEHRING et al., 2022) aims for heuristic functions that generalize beyond a specific task. These approaches rely on the model to build domain-specific or multi-domain heuristics. The usual setting is to train different architectures of neural networks with samples of small tasks of a domain generated with a logic-dependent method and evaluated on larger tasks of the same domain. These architectures can be general networks such as Neural Logic Machines (DONG et al., 2018) and Graph Neural Networks (SCARSELLI et al., 2008; GORI; MONFARDINI; SCARSELLI, 2005).

In the context of planning, Shen, Trevizan and Thiebaux (2020) use Hypergraph Neural Networks (HGN) as an extension of graph networks (BATTAGLIA et al., 2018). HGN aims to learn planning heuristics through training, focusing on developing domain-independent heuristics capable of generalizing across various domains, as well as domain-specific and multi-domain heuristics. The HGN encompasses vertices representing task propositions within a hypergraph structure, with edges denoting operators connecting the preconditions to their effects.

Another approach proposed explicitly for planning tasks is the Action Schema Network (ASNet) (TOYER et al., 2018). ASNets are composed of alternating proposition and action layers, with the first and last always being an action layer. Each action layer contains an action module for each operator in a specific task, while each propositional layer contains a proposition module for each fact in the task. Weight sharing is used to optimize efficiency, where action modules with operators derived from the same action schema share the same weight, and proposition modules with facts derived from the same predicate share the same weight. This weight sharing allows for the reuse of a single set of learned weights across all tasks within a class of planning problems.

These networks achieve competitive results compared to traditional heuristics and can generalize well, but require the logical description of the domain and the task to be instantiated. These approaches also help in understanding learning heuristics. For example, the main goal of Ståhlberg, Bonet and Geffner (2022) is to understand the expressive power and limitations of learning heuristics. The main limitation of these approaches is the strong dependence on the domain model and task description. They also require too much memory and are slow. Generally, they are too big to be instantiated for the size of tasks typically used. For example, HGN has $32.8\,\%$ coverages on the large planning tasks used in this thesis.

2.4.2 Non-structured NN-based Approaches

The second set of approaches (FERBER; HELMERT; HOFFMANN, 2020; YU; KUROIWA; FUKUNAGA, 2020; FERBER et al., 2022; O'TOOLE et al., 2022) accesses the model for computing mutexes and is typically task-specific. Specific to each task, they can perform better than the approaches in the first set. They typically train an FNN and evaluate the learned heuristic on a state space using tasks with the same goal and different initial states. These networks are trained with pairs of states and cost-to-goal estimates. Ferber, Helmert and Hoffmann (2020) systematically studied hyperparameters on the FNN and found that, for a fixed architecture, two aspects significantly influence how informed the heuristic is: the subset of selected samples and the size of the sample set.

Ferber et al. (2022) use a combination of backward and forward searches (ARFAEE; ZILLES; HOLTE, 2011). First, they generate new initial states with backward random walks and then solve them with a GBFS guided by a learned heuristic. The sampling is performed in parallel with the training. The number of samples per search varies throughout the process, starting with a random value ranging from 0 to 5 and doubling it as plans are found. The plans found provide the samples for the next training epoch, where each sample is a state in a plan with the cost-to-goal estimate as its distance to the goal through the plan. Their FNN architecture is a ResNet with two hidden layers and a residual block consisting of two more hidden layers.

O'Toole et al. (2022) use the same FNN architecture as Ferber et al. (2022), which is also applied in this thesis. They use random walks to perform five backward searches from the goal, with a depth of 500, where the depth at which the state is generated serves as the cost-to-goal estimate. Each sampled partial state is converted into 20 complete states and added to the sample set. Furthermore, they sample an additional $50\,\mathrm{K}$ randomly generated states with a cost-to-goal estimate equal to the maximum value in the sample set plus one, resulting in a total of $100\,\mathrm{K}$ samples. They showed that random sampling improves the performance of the heuristic function by including in the sample set states from regions of the state space not reached by the backward search.

Yu, Kuroiwa and Fukunaga (2020) used a backward search approach with the DFS algorithm. In their best configuration, they sample $100\,\mathrm{K}$ states in 500 searches, i.e., 200 states per search, with a cost-to-goal estimate equal to the depth at which the state was generated. In contrast to the previous approaches, they use a compact FNN consisting of only one hidden layer with 16 neurons.

The methods from the non-structured NN-based approaches are highly independent of the domain model and task description and require low computational resources to generate samples and train the FNN. However, despite having competitive results compared to logic-based heuristics, they can still not surpass the goal-count heuristic.

3 SAMPLE GENERATION

Our objective is to systematically investigate sample generation methods to understand the influence of each technique on the performance of sampling strategies. We explore the effects of each component in sample generation and propose new strategies to improve the performance of the learned heuristic. By systematically analyzing and comparing the outcomes of each strategy, we can gain insights into the strengths and weaknesses of different sample generation approaches. This understanding will allow us to identify how each technique contributes to enhancing performance and guide us to a better learned heuristic function.

Therefore, we focus on how two aspects of sample generation influence the performance of the learned heuristic to guide a search algorithm: the distribution of states s_i in the state space and the quality of the estimates h_i of samples concerning the h^*-value. Learning a heuristic function requires a set of samples $(s_1, h_1), \ldots, (s_N, h_N)$, where each sample $(s_i, h_i), i \in [N]$ consists of a state s_i and a cost-to-goal estimate h_i.

We restrict our study to generalizing over planning tasks with initial states part of the same FSS and a fixed goal condition. We study logic-independent approaches with access to predecessors and successors of partial states through a black-box function, to the goal condition, and to the domain of each variable. We also study mutex-based approaches with access to mutexes derived from Fast Downward (HELMERT, 2006). We address the generation of states in Section 3.1 and the estimation of the cost-to-goal in Section 3.2. In both sections, we explore techniques from the previous works and introduce novel methods. The methods are a sampling strategy combining regression by breadth-first search with random walk, an adaptive regression limit based on the input task parameters, and two improvement methods for the cost-to-goal estimates. Finally, Section 3.3 presents the workflow of all the techniques discussed and an example of their applications in the Blocksworld domain.

3.1 Generation of States

Unlike other domains in machine learning, such as image recognition or natural language processing, where datasets of samples are often collected in real-world experiments and subsequently need to be manually annotated and carefully curated, the task of sample generation in classical planning presents a particular problem. In contrast to the reliance on external data sources, classical planning uses the structure of the task domain. Thus, the generation of samples becomes an algorithmic problem that uses the availability of the state space and the ability to compute cost-to-goal estimates, allowing for a systematic and controlled investigation of various techniques using only computational resources.

Previous works typically address three fundamental approaches: progression from one or more initial states, random sampling of the state space, or regression from a goal state. Progression and regression involve starting from a state and systematically applying forward (progression) or backward (regression) operators to expand the state space. There are several expansion strategies, such as random walk (RW), BFS and DFS, or teacher searches which include methods using reinforcement learning or bootstrapping (ARFAEE; ZILLES; HOLTE, 2011). Each strategy influences the coverage and expansion patterns of the generated states, contributing to the overall diversity and representation of the sample set.

A problem in both random sampling and progression-based methods is obtaining the cost-to-goal estimates. Without access to efficiently computable heuristic functions, or in logic-independent approaches, obtaining these values is a challenging task. Typically, search algorithms must be used to generate these estimates, which can introduce significant computational costs in tasks with large state spaces.

To remain less dependent on models than logic-based methods and more general, we focus on regression for which an upper bound on the cost-to-goal is readily available, discussed in Sections 3.1.1 and 3.1.2. Regression leads to partial states, so the problem of generating complete states is addressed in Section 3.1.3. Using random sampling as an additional approach is discussed in Section 3.1.4.

3.1.1 Sampling by Regression

To generate samples, we expand states from the backward state space through regression using the main techniques in the literature, such as expansion by BFS, DFS, and RW. Regression sampling consists of one or more regression rollouts involving a series of state expansions. A rollout starts in an arbitrary state and continues until the last expanded state has no predecessors or a maximum regression limit is reached. Therefore, RW can have multiple rollouts, while BFS and DFS only have one. The sampling generation stops when the total number of samples N has been reached.

During expansion, we optionally use mutexes obtained from an analysis of the planning task – in our case, as computed by Fast Downward (HELMERT, 2006) – to avoid partial states which cannot be completed to complete states without violating a mutex. We also avoid repeated partial states for random walk rollouts, such that a single rollout never cycles. However, the same partial state can be sampled multiple times in different rollouts.

Regression starts from the goal s^* with the cost-to-goal estimate $h(s^*) = 0$. When applying a backward operator $o_r \in \mathcal{O}_r$ to a state s, the resulting state $s' = \text{pred}(s, o_r)$ receives a cost-to-goal estimate of $h(s') = h(s) + \text{cost}(o_r)$. For a state s generated during rollout that satisfies the goal condition, i.e., $s \subseteq s^*$, we set its cost-to-goal estimate to 0. States are added to the sample set when generated in the random walks and when expanded in BFS or DFS. In all methods, backward operators applicable to a state are applied randomly.

Using different expansion strategies and considering multiple rollouts, we aim to generate a diverse and representative sample set of the state space. Different expansion strategies generate sample sets with different distributions over the state space. In our experience, good coverage of states close to the goal, such as those obtained by BFS or random walks, is valuable, as is the greater depth obtained by DFS or random walks. However, random walks from a goal state often sample states close to the goal multiple times, while DFS can result in samples that are distant. Based on these observations, we propose a novel combination of BFS and random walks called Focused Sampling Method (FSM) that aims to have a good coverage close to the goal, from the BFS, and a diverse set of samples from the remaining state space, from the random walks. By integrating these techniques, FSM aims to achieve a balanced and comprehensive sampling of the state space. Algorithm 1 presents the approach.

FSM has two phases. In the first phase, a fixed percentage p of the desired N samples is generated using BFS. The percentage p also serves as a constraint on computational resources, and the BFS process terminates when either the desired number of samples or the resource limit is reached first. BFS expands a state from layer k and generates n states from layer $k + 1$ (line 11). Each generated state is inserted into the sample set S if the current total samples plus n are within pN; otherwise, no states are sampled, and BFS expands another state. When expanding an entire layer, the algorithm moves to the next layer.

Let Q be the states in S that did not generate successors for the sample set,

Algorithm 1: FSM algorithm

```
Data: N: number of samples; p: fraction of samples obtained by BFS; s0:
      initial state; L: random walk rollout limit
Result: S: the sample set
Function FSM(p, s0, L):
    $S \leftarrow \{s_0\}$
    $Q \leftarrow \{s_0\}$
    $D_k \leftarrow \{s_0\}$ // current layer
    $D_{k+1} \leftarrow \emptyset$ // next layer
    $N_{BFS} \leftarrow N \times p$
    // BFS phase
    while $|S| < N_{BFS}$ and $D_k \neq \emptyset$ do
        Shuffles $D_k$ with a uniform distribution
        foreach $s \in D_k$ do
            $t \leftarrow \{s' \mid s' \in \mathrm{pred}(s)$ and $s' \notin S\}$
            if $|S| + |t| \leq N_{BFS}$ then
                $D_{k+1} \leftarrow D_{k+1} \cup t$
                $S \leftarrow S \cup t$
                $Q \leftarrow (Q \cup t) \setminus \{s\}$
        $D_k \leftarrow D_{k+1}$
    // RW phase
    while $|S| < N$ do
        Shuffles Q with a uniform distribution
        foreach $s \in Q$ do
            $S_{RW} \leftarrow$ set of $\min(L - h^*(s), N - |S|)$ states sampled via RW
              starting from s
            $S \leftarrow S \cup S_{RW}$
            if $|S| = N$ then break;
    return S
```

i.e., they were not expanded, or their set of successors did not fit. The second phase generates multiple random walk rollouts, each starting from a state $s \in Q$ chosen randomly with a complete replacement after all states have been selected once. Each rollout samples $L - h^*(s)$ states, where L is the maximum regression limit and $h^*(s)$ is the distance already traversed by BFS. During a random walk, states sampled in the BFS phase are avoided. The random walks are repeated until reaching N samples in the sample set.

3.1.2 Maximum Regression Limit

The maximum regression limit controls the depth of the rollout during sample generation. This limit determines how far the sampling can deviate from the goal during regression. In random walk, the regression limit is represented by the maximum length (number of random steps) the algorithm can take in a rollout. In BFS or DFS, the maximum regression limit corresponds to the maximum depth (distance from the goal) that can be reached.

This limit serves two main objectives. First, particularly in regression-based methods, it helps maintain the precision of the cost-to-goal estimate, which is determined by the distance from the goal state that the sample was generated and tends to degrade during sampling due to the random nature of algorithms such as random walk or DFS. Second, it regulates the distribution of samples by periodically restarting the sampling, effectively distributing them across the different distances from the goal.

By establishing an appropriate maximum regression limit, we can balance the expansion of the state space and precision in the cost-to-goal estimates. If the limit is too low, the sampling may fail to cover areas far from the goal, typically where the initial states of a search reside, resulting in a sample set lacking information from these regions. On the other hand, if the limit is too large, the sampling may extend too far from the goal, increasing the error on the cost-to-goal estimates.

A simple strategy is to define some maximum limit L. Previous work has used this, e.g., Yu, Kuroiwa and Fukunaga (2020) and O'Toole et al. (2022) with $L = 200$ and $L = 500$, respectively. A fixed limit is not the optimal choice for tasks with state spaces of different maximum distances to a fixed goal when we aim for a representative sample of the state space. Therefore, using adaptive strategies is important to dynamically adjust the maximum rollout limit, ensuring the generation of a representative sample set that captures the characteristics of the state space.

The ideal regression limit is the one that stops regression when reaching the furthest state from the goal. Let d^* denote the distance from the goal to its farthest state. For BFS, d^* would be the ideal estimate; for DFS and random walks, higher limits are required since they do not follow the shortest paths. Since d^*, in general, is unknown, a potential solution is to estimate this value using available or domain-specific information.

We propose two adaptive and approximate methods to define a suitable maximum regression limit based on the input task parameters. The first method uses the number of facts $F = |\mathcal{F}(s_0)|$ to estimate the d^*. Since each state is represented by a set of facts $\mathcal{F}$, we assume that each operator changes exactly one fact in the worst-case scenario. Therefore, starting from a particular state, it is possible to modify all the facts by applying F operators and thus reaching any other state in the state space.

However, operators can modify more than one fact at a time, enabling a more precise estimation by considering the average number of facts changed by the task's operators. Therefore, we propose the second method, denoted as

$$\bar{F} = \left\lceil \frac{F}{\sum_{o \in \mathcal{O}} \frac{|\operatorname{eff}(o)|}{|\mathcal{O}|}} \right\rceil,$$

i.e., the number of facts per mean number of effects in the operators. This technique generates a more refined estimate of the d^*.

Using these adaptive and approximated methods, we can define the maximum regression limit based on the characteristics of each planning task. The regression limit F offers a simple yet effective estimation, while the $\bar{F}$ provides a more refined approximation.

3.1.3 State Completion

The goal condition is represented as a partial state, consisting of a subset of facts of a complete state that need to be satisfied to reach the desired solution. Therefore, sampling by regression generates a set of partial states. Since neural networks use complete states as input during the search, it becomes necessary to perform state completion, which involves assigning values to the undefined variables in the partial states. By using additional information and making more informed decisions during the state completion, we can investigate the gain it can have on the neural network's learning.

We introduce three approaches to complete a partial state. The first one is a random assignment, where each partial state can be completed by assigning a value $s(v) \in \operatorname{dom}(v)$ to all fact pairs $(v, s(v))$ where $s(v) = \perp$. This completion technique is logic-independent since it relies solely on the available domain values without making additional assumptions.

The second method is mutex-based and aims to avoid states that are impossible to reach during the search. It also assigns a random value $s(v) \in \operatorname{dom}(v)$ to each undefined variable v but excludes states that do not satisfy mutexes. This filtering is achieved by rejection sampling, where the undefined variables are processed in random order and set to a random value that does not violate the mutexes. If the state cannot be completed, we leave the facts undefined, i.e., set to false. By using the mutex information, we generate states more consistent with those in the forward state space, enhancing the quality of the training data.

While this mutex-based solution provides enhanced state completion, generating invalid states or states that cannot be reached during the search is still possible. We introduce an ideal state completion method to investigate the influence of generating only those states that are reachable from the initial states of interest. This method aims to produce a sample set that is highly relevant to the specific task. To complete a partial state s, we sample a random state from $s \cap$ FSS. Since we sample by regression, for some states, $s \cap$ FSS may be empty; such states are avoided during regression. However, it is important to note that this method applies only to small tasks where we can enumerate the complete forward state space of the initial state s_0.

3.1.4 Randomly Generated Samples

We explore random sampling, as proposed by O'Toole et al. (2022), as an additional approach to generate samples. They have shown that adding randomly generated samples to a set of samples generated by expansion improves the performance of the learned heuristic. In their work, they propose to set the cost-to-goal estimate for these samples to $L + 1$ for a maximum regression limit of L.

We use this method to investigate the influence of randomly generated samples on the sample set. These samples are generated from a fully undefined state s, i.e., $s(v) = \perp$ for all $v \in \mathcal{V}$, using the mutex-based state completion described in the previous section. When generating a state s by random sampling, if s is already present in the sample set, it receives the same cost-to-goal estimate; otherwise, it receives a cost estimate of $1 + \max_{i \in [N]} h_i$ that is larger than all sample estimates. By using random samples, we aim to investigate their influence on the overall performance of the sampling approach.

3.2 Improving Cost-to-Goal Estimates

The cost-to-goal estimate's accuracy also influences the quality of the sample set. We aim to reduce the difference from the actual goal distance h^* to improve the cost-to-goal estimate. Our sampling approach ensures that the cost-to-goal estimates assigned to the samples are based on the operators used to reach that particular state. Thus, we guarantee that a sample never underestimates h^*.

Property 3.1. *The cost-to-goal estimate $h(s)$ of a sample s obtained by regression satisfies $h(s) \geq h^*(s)$.*

Proof. Each estimate is witnessed by a plan. As observed in Chapter 2, a valid regression sequence $\rho = (o_1, \ldots, o_k)$ generates a sequence of partial states that can reach the goal in at most k steps and with a cost at most $\sum_{o \in \rho} \text{cost}(o)$, which cannot be lower than the optimal cost. □

Therefore, we apply two procedures that improve the cost-to-goal estimates but maintain Property 3.1. The first minimizes estimates over repeated samples, while the second over successors of samples. Both are logic-independent techniques.

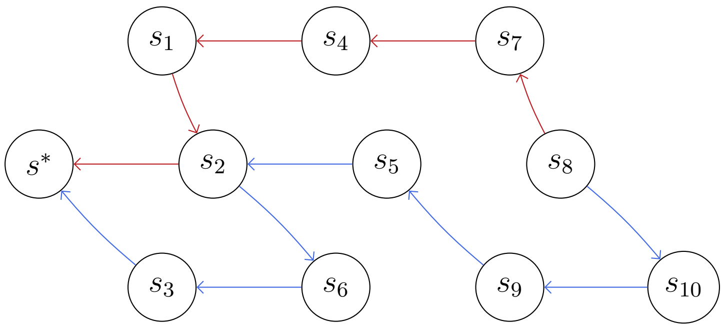
3.2.1 Improvement of Repeated Samples

When generating samples through multiple rollouts, there is a possibility of overlap, where the same state could appear at different distances across different rollouts, leading to varying cost-to-goal estimates. When training the NN, different labels for the same sample can result in inconsistency and affect learning. To address this issue and improve the quality of cost-to-goal estimates, we propose the Sample Improvement (SAI).

SAI aims to improve the consistency of the training data by selecting a single cost-to-goal estimate for each unique state. Property 3.1 guarantees that no sampled cost-to-goal estimate underestimates the actual goal distance h^*, so the lower the value, the closer it is to h^*. Therefore, we choose the sample with the lowest cost-to-goal estimate to represent each particular state.

For all sampled states s we update each cost-to-goal estimate to the best estimate $h(s) = \min\{h_i \mid s = s_i, i \in [N]\}$. Since different partial states can generate identical complete states, the improvement is applied to partial and complete states. Choosing

Figure 3.1 – SAI technique applied on samples of two random walk rollouts. Each node represents a state, and each arc is an applicable operator. Each rollout has a color, and its applied operators are colored accordingly.



the minimum h-value is sound since, in all cases, we have valid plans from a regression that witness these distances; for the same reason, Property 3.1 still holds.

Figure 3.1 presents the technique applied to sampling from random walks. Both rollouts generate the state s_2 but through different paths. The red rollout reaches it with a distance of one from the goal state s^*, and the blue rollout generates s_2 after three hops. Consequently, we have two samples $(s_2, 1)$ and $(s_2, 3)$. Applying the SAI technique, we update the cost-to-goal estimate $h(s_2)$ of both samples to $\min(1, 3) = 1$. The same process is applied to samples of state s_8, which receives $h(s_8) = \min(5, 7) = 5$.

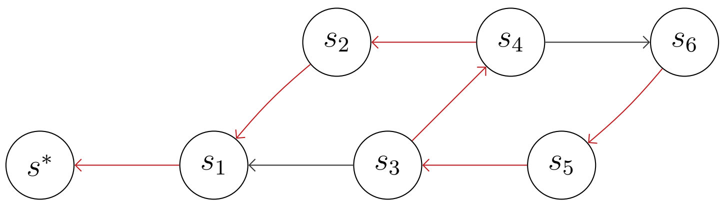
3.2.2 Improvement over Successors

In addition to sampling the same states in different rollouts, sampling states neighbors in the state space is common – more frequent in states at the beginning of the rollout. By sampling neighboring states, we can use the local information to enhance the accuracy of the cost-to-goal estimates. For this, we propose a technique called Successor Improvement (SUI).

SUI uses the fact that neighboring samples, distant by an operator, can be connected by this operator to form a new path to the goal and, consequently, new cost-to-goal estimates. We can update the corresponding cost-to-goal estimate to approximate h^* if it produces a shorter path than the current one. Note that although it is possible to expand the technique to neighbors of two or more distance operators, the computational cost grows exponentially with the average number of successors per sampled state.

Consider a directed, weighted, and labeled graph $G = (V, A)$ where each vertex in V corresponds to a sampled state and is labeled by its lowest cost-to-goal estimate. We

Figure 3.2 – SUI technique applied on samples of a random walk rollout (red arches). Each node represents a state, and each arc is an applicable operator.

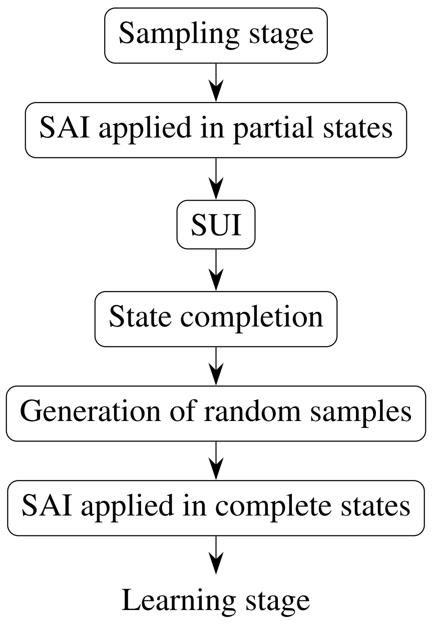


insert an arc (s, t) into A for every pair of states $s, t \in V$ such that $t \in \mathrm{succ}(s)$. When generating a successor of a partial state, we typically generate another partial state. Since a partial state represents a set of complete states, to determine if a successor $s' \in \mathrm{succ}(s)$ matches any sampled state t, we need to compute the subsets of t. For this, we maintain a trie data structure, where each sampled state is inserted with its variable values as key. When generating a successor s', we search the trie for sampled states t supersets of s'; that is, s generates t from applying an operator o and assigning zero or more undefined variables. Thus, we can include an arc (s, t) in A with a weight $c_{s,t} = \text{cost}(o)$.

Using the graph G, we propagate the cost-to-goal estimate from each sampled state to its sampled predecessors. We iterate over each arc $(s, t) \in A$ and update the cost-to-goal estimate $h(s) = \min(h(s), h(t) + c_{s,t})$. The process continues as long as there are updates. For partial states generated by regression, by construction, at least one successor exists, except for the goal s^*. Therefore, in the worst case, the algorithm will make L iterations, propagating the cost-to-goal estimate of a goal state to the most distant state. As for SAI, all distances are witnessed by plans, so Property 3.1 is maintained.

Figure 3.2 shows an example. The rollout does not overlap but intersects at a distance of one operator between states s_1 and s_3. This information is discovered by inserting the black arcs into the graph G, which are operators not previously used by the sampling. From there, we can connect these states to generate a new path from s_3 to the goal that is shorter than the one produced by the rollout, updating its cost-to-goal estimate from 4 to $h(s_1) + 1 = 1 + 1 = 2$, according to the new path. Now, s_3 can update the estimate of its neighbors in the next iteration. Thus, all states generated in the rollout after s_3 will be updated. Therefore, although distant states from the goal have fewer neighbors than those near the goal, their cost-to-goal estimates are updated through the rollout propagation.

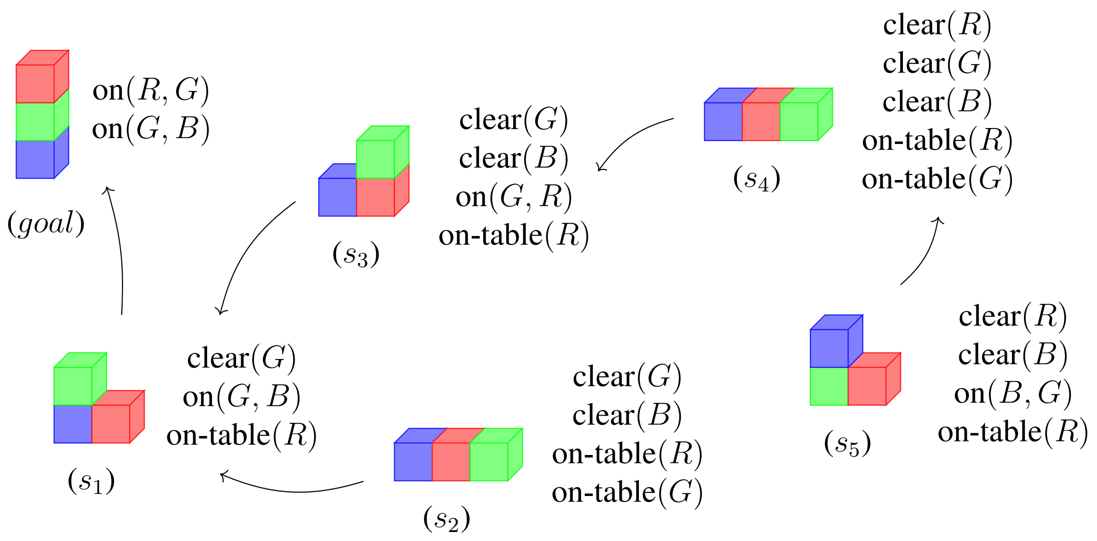
Figure 3.3 – Sample generation workflow.


Sampling stage
SAI applied in partial states
SUI
State completion
Generation of random samples
SAI applied in complete states
Learning stage

3.3 Workflow

Our approach follows the workflow shown in Figure 3.3. The workflow begins with the sampling stage, where a set of samples is generated using algorithms such as random walk or FSM. The sampling must be performed through regression to enable the SAI and SUI. After obtaining the sample set, the first step of sample processing is to apply SAI to the samples, represented as partial states at this stage. Then, the SUI technique is performed, which achieves the same outcome even if SAI is not applied to the partial states. However, applying SAI beforehand is computationally advantageous – except in cases where the effect of the SAI is minimal – as it reduces the number of iterations required to update the cost-to-goal estimates in the SUI stage. The SUI is the last step, where the samples are treated as partial states. Next, the undefined variables of each sample are assigned according to the chosen state completion technique, resulting in a sample set of complete states. Then, we use random sampling to generate additional states for the sample set. By generating the random samples before applying SAI to complete states, there is no need to check if each randomly generated sample is already in the sample set to copy its cost-to-goal estimate since a high cost-to-goal estimate $L + 1$ is reduced by the SAI in this case. This step is handled by SAI, which updates the value if necessary. Applying SAI to the complete states concludes the preprocessing of the sample set before it proceeds to the supervised learning stage. Overall, this workflow applies a series of transformations to the sample set. Each step contributes to adjusting

Figure 3.4 – Example of sampling in Blocksworld. Each set of blocks corresponds to a state, with its facts described on the side and its label below.



the distribution of samples over the state space or refining their cost-to-goal estimates, both key factors for enhancing the quality of the sample set.

Examples of all techniques can be extracted from Figure 3.4. The subset of the state space presented consists of six states from the Blocksworld with three blocks. Each state has a set of facts that describe its configuration. These facts include clear(A) to indicate that block A has no block above it, on(A, B) to denote that block A is placed on top of block B, and on-table(A) to signify that block A is on the table. The block labels are abbreviated based on their colors, with red, blue, and green named as R, B, and G, respectively. In this context, the goal state is defined as a stack of all blocks, with the blue block on the table and the red block on top. The described facts are based on applying backward operators from the goal state.

We can illustrate a regression using the FSM algorithm. Suppose we want to sample six states, i.e., $N = 6$, with $p = 0.67$ and a maximum rollout limit of $L = 4$. The goal state is initially sampled and expanded, adding the state s_1 to the open queue Q (see Algorithm 1). In the second iteration, s_1 is expanded, adding s_2 and s_3 to Q, which are also expanded and sampled in the next iteration. The BFS phase ends with four samples, and the random walk phase begins, where s_2 and s_3 are the starting states for rollouts. Consider s_3 for the first rollout, which generates s_4 and then s_5. The starting state s_3 was sampled in the BFS and is not sampled again, so only s_4 and s_5 are inserted in the sample set. Upon reaching s_5, the rollout ends due to the maximum regression limit $L = 4$. The sample budget is reached, and the sample generation ends. If the sample budget were larger, a new rollout would start from a predecessor of s_2 or s_3 excluding s_4. Assuming unit costs, the cost-to-goal estimate of a sample is the number of hops to the goal, e.g., $h(s_4) = 3$.

Originally, a random walk rollout cannot visit states sampled by BFS. However, even though s_4 has the same configuration as s_2, they have different facts, making them distinct states according to duplicate detection. Note that $h(s_2) = 2$ and $h(s_4) = 4$, i.e., the cost-to-goal estimate of s_4 overestimates h^*. When using the mutex-based approach to complete states, both s_2 and s_4 become the same complete state $s = \{$clear(R), clear(G), clear(B), on-table(R), on-table(G), on-table$(B)\}$, and the SAI updates the cost-to-goal estimate $h(s_4) = \min(4, 2) = 2$. On the other hand, SUI also adjusts the value. When creating the graph G, a new arc is inserted from s_4 to s_1 motivated by the operator that moves the green block onto the blue block. Thus, we can update the estimate of s_4 to $h(s_4) = h(s_1) + 1 = 1 + 1 = 2$, reaching h^* in the same way. In the next iteration, the cost-to-goal estimate of s_5 is updated to $h(s_5) = h(s_4) + 1 = 2 + 1 = 3$, also reaching h^*.

4 EXPERIMENTS

In this section, we present two sets of experiments. In the first set, we analyze the behavior of sampling methods on planning tasks for which we can enumerate the complete state space with associated perfect cost-to-goal estimates h^*. We study how different techniques can influence the number of state expansions by a search algorithm. In the second set, we evaluate how our findings generalize to a practical setting with large planning tasks. Our methods are then compared to traditional heuristics and previous works.

4.1 Settings

We use a residual neural network (HE et al., 2016) to learn a heuristic for a state space. The network's input is a Boolean representation of the states, where a fact is set to 1 if it is true in the state and 0 otherwise, as explained in Section 2.3.2, and its output is a single neuron with the predicted h-value. The network has two hidden layers followed by a residual block with two hidden layers. Each hidden layer has 250 neurons that use ReLU activation and are initialized as proposed by He et al. (2015). The training uses the Adam optimizer (KINGMA; BA, 2015), learning rate of 10^{-4}, early-stop patience of 100, and MSE loss function. Due to better results in preliminary experiments, we use batch sizes of 64 for small and 512 for large state spaces. We use $90\,\%$ of the sampled data as the training set, with the remaining $10\,\%$ as the validation set. Different learned heuristics are denoted as $\hat{h}_X$, where X indicates different algorithmic choices.

We select the domains and tasks from Ferber et al. (2022): Blocksworld, Depot, Grid, N-Puzzle, Pipesworld-NoTankage, Rovers, Scanalyzer, Storage, Transport, and VisitAll. All domains have unit costs except for Scanalyzer and Transport, for which we consider the variant with unit costs. All methods are implemented on the Neural Fast Downward planning system with PyTorch 1.9.0 (FERBER; HELMERT; HOFFMANN, 2020; PASZKE et al., 2019). Our source code, planning tasks, and experiments are available[1]. All experiments were run on a PC with an AMD Ryzen 9 3900X processor, using a single core with $4\,$GB RAM per process. The GPU provides a subtle speedup, so it was not used. We solve all tasks with GBFS guided by the heuristic h with the lowest generation order as the tie-breaking strategy.

[1]Available at <https://github.com/bettker/NeuralFastDownward>

We observe that an NN may fail to train if, after initialization, it outputs zero for all training samples – referred to as “born dead” in Lu et al. (2020). This condition arises when the weights and biases of the NN are initialized to consistently map the ReLU activation region to negative values, resulting in a zero gradient and no weight updates during training. This phenomenon occurs with a non-negligible frequency in experiments on smaller state spaces with fewer samples, mainly in the Blocksworld and VisitAll domains. A smaller sample set has limited input variability, reducing the chances of at least one sample correctly activating the ReLU units. In that case, we reinitialize with a different seed until the network outputs a non-zero value for some sample.

To establish a point of comparison for evaluating the performance of the approaches, we propose a baseline $\hat{h}_0$. The baseline refers to a neural network configured similarly to previous non-structured NN-based methods described in Section 2.4.2. The $\hat{h}_0$ is trained using random walks with $L = 200$. Mutexes are applied during the regression and for state completion, but resetting the h-value to zero in goal states and the improvement strategies SAI and SUI are turned off. By comparing the results of our approaches with the baseline, we can assess the potential improvements over existing methods.

To determine a value for p, preliminary experiments in small state spaces were performed using $p \in \{0.01, 0.05, 0.1, 0.2, \ldots, 0.9\}$. We use the same baseline configuration but with the FSM sampling algorithm. The corresponding geometric mean expansions obtained were 84.92, 79.51, 74.05, 79.46, 80.39, 80.79, 96.17, 120.42, 133.99, 167.14, and 175.58, respectively. As a result, we set a fixed value of $p = 0.1$ for the experiments.

4.2 Small State Spaces

In this section, we study the behavior of different sampling methods on small state spaces. For each domain, we select the task from the IPC benchmarks with the largest state space between $30\,\mathrm{K}$ and $1\,\mathrm{M}$ states that can be enumerated completely to obtain h^*-values. Table 4.1 shows the tasks and their state space sizes. For domains Grid, Rovers, Scanalyzer, and Transport, the best task found had fewer than $30\,\mathrm{K}$ states, and VisitAll more than $1\,\mathrm{M}$, so we manually modified these tasks. We could not find a non-trivial task within our limits for Depot, Pipesworld-NoTankage, and Storage, so they were excluded from our experiments. We generate the initial states for the small state spaces by performing a random walk of length 200 from the original initial state of a task.

Table 4.1 – Size of the forward state spaces for the selected small tasks in seven domains. Tasks marked with $*$ were modified.

Domain	Task	#States	Domain	Task	#States
Blocks	blocks-7-0	65990	Scanalyzer	p03*	46080
Grid	prob01*	452353	Transport	p02*	637632
N-Puzzle	prob-n3-1	181440	VisitAll	p-1-4*	79931
Rovers	p03*	565824			

Rovers, Scanalyzer, and VisitAll, had duplicated initial states or states that satisfied the goal condition. Thus, we generate the initial states for these domains with random walks of length 25, 50, and 8, respectively.

In the small state space experiments, the coverage for all methods is $100\,\%$. Therefore we use the number of expanded states to evaluate the quality of the heuristic function. In these experiments, we report means over experiments in 25 models (five different network seeds and five different sample seeds) and 50 initial states. The training time has been limited to 30 minutes. After initialization, the percentage of NN that output 0 for all samples was at most $40\,\%$ (in Blocksworld), and all networks successfully passed the initialization test after the second initialization. Under these conditions, less than $1.5\,\%$ of the NNs did not converge within the time limit.

We aim to analyze the influence of the distribution of sampled states in the state space on the quality of the learned heuristics and how techniques used to improve cost-to-goal estimates influence sample quality. If not stated otherwise, we use the number of samples equal to 1% of the state space size, regression limit $L = 200$, mutex-based completion, and no cost-to-goal improvement methods.

4.2.1 Sample Generation Algorithms

This experiment compares four sample generation algorithms: BFS, DFS, RW, and FSM. To control the cost-to-goal estimates' effect on the learned heuristic's quality, we replace sample estimates with perfect values h^* before training. Table 4.2 shows the number of expanded states of a GBFS guided by the learned heuristics and the mean h^*-values over the sampled states. We see that heuristic $\hat{h}_{\text{BFS}}$ leads to more expanded states than $\hat{h}_{\text{DFS}}$, which in turn expands about $50\,\%$ more states than $\hat{h}_{\text{RW}}$ and $\hat{h}_{\text{FSM}}$, which perform similarly. Using $\hat{h}_{\text{BFS}}$ is significantly worse and leads to the highest or close to the highest number of expansions in all domains. Heuristic $\hat{h}_{\text{DFS}}$ has a high number of

Table 4.2 – Comparison of sampling strategies BFS, DFS, RW, and FSM on h^*-values. Expanded states of GBFS with learned heuristics and mean h^*-values over the sample sets.

	Expanded states				Mean h^*-values				
Domain	$\hat{h}_{\text{BFS}}$	$\hat{h}_{\text{DFS}}$	$\hat{h}_{\text{RW}}$	$\hat{h}_{\text{FSM}}$	FSS	BFS	DFS	RW	FSM
Blocks	5047.1	85.9	44.4	**37.3**	18.8	10.8	17.7	11.9	14.4
Grid	112.7	122.4	**99.2**	110.3	16.6	5.3	17.1	7.2	8.9
N-Puzzle	1477.2	176.8	109.5	**109.2**	22.0	10.4	20.2	20.0	19.8
Rovers	12.9	13.1	**11.8**	11.8	6.4	2.3	5.2	5.0	5.0
Scanalyzer	175.9	25.5	**24.5**	24.9	8.3	2.9	7.9	7.1	6.5
Transport	135.1	45.0	19.6	**19.2**	12.2	2.9	11.3	10.0	9.5
VisitAll	53.3	22.0	19.2	**18.7**	9.0	2.0	9.1	6.8	6.6
Geo. mean	201.9	48.3	34.0	33.5	12.2	4.2	11.4	8.8	9.1

expansions in Blocksworld, N-Puzzle, and Transport. Looking at the mean h^*-values, we see that samples generated by BFS have the lowest, and those by DFS the highest mean estimates in all domains. Although the distribution of DFS is closest to that of the whole state space (inferred from the mean h^*-values), the resulting heuristic expands more states than RW and FSM, which generate states closer to the goal. Therefore, multiple random walk rollouts seem better than one with BFS or DFS due to increased sample diversity in different portions of the state space, covering states more likely to be visited during the search.

We now compare these results to results shown in Table 4.3, obtained on exactly the same states but using the cost-to-goal estimates obtained during sampling for training the NN. Note that the results for BFS with estimated costs to the goal differ from those with exact values in Table 4.2. This difference happens because, during regression with BFS, the cost-to-goal estimates are only exact on partial states; when turning them to complete states, the estimates can be larger than h^*. Thus $\hat{h}_{\text{BFS}}$ with the estimates obtained during regression is less informed.

We can see that the relative order of the methods concerning the number of expanded states remains the same, although all methods expand more states. The increase in the number of expanded states is highest for $\hat{h}_{\text{DFS}}$, which expands about seven times more states. In contrast, the other methods expand about twice as much, meaning that the estimates produced by DFS during regression are inferior to those produced by the other methods. The mean h-values confirm this: we can see that DFS significantly overestimates the true distances. Although BFS has an estimation quality close to h^*-value, its expanded states also degrade. These results suggest that sampling more states in localized

Table 4.3 – Comparison of sampling strategies BFS, DFS, RW, and FSM on estimated h-values. Expanded states of GBFS with learned heuristics and mean h-values over the sample sets.

	Expanded states				Mean h-values				
Domain	$\hat{h}_{\text{BFS}}$	$\hat{h}_{\text{DFS}}$	$\hat{h}_{\text{RW}}$	$\hat{h}_{\text{FSM}}$	FSS	BFS	DFS	RW	FSM
Blocks	5047.1	205.6	87.0	**80.8**	18.8	10.8	166.1	28.1	38.4
Grid	431.2	4102.4	**263.6**	276.2	16.6	6.8	184.6	21.1	22.5
N-Puzzle	1477.2	1092.5	237.6	**177.1**	22.0	10.4	187.5	96.3	90.6
Rovers	91.8	27.8	21.9	**18.8**	6.4	4.6	27.1	25.9	24.9
Scanalyzer	328.0	263.5	70.6	**53.6**	8.3	3.0	146.2	90.3	87.9
Transport	215.1	1321.1	**111.0**	130.1	12.2	3.3	196.1	95.2	88.6
VisitAll	170.4	44.6	22.0	**21.5**	9.0	2.7	28.5	22.5	22.4
Geo. mean	446.7	326.7	79.8	73.1	12.2	5.1	103.5	43.3	44.4

regions of the state space (BFS closer to the goal and DFS more distant) is insufficient to achieve good results during the search with GBFS. Furthermore, $\hat{h}_{\text{FSM}}$ expands fewer states than $\hat{h}_{\text{RW}}$ and is the best in five of seven domains. Because $\hat{h}_{\text{FSM}}$ had a lower increase in expansions compared to $\hat{h}_{\text{RW}}$, we use FSM in the remaining experiments.

4.2.2 Maximum Regression Limit

In this experiment, we analyze the influence of the regression limit on the number of expanded states of the sample generation technique FSM. We compare a fixed regression limit of $L = 200$ with the adaptive rollout limits number of facts F and number of facts divided by the mean number of effects $\bar{F}$. Values F and $\bar{F}$ for the selected tasks and the largest distance of any state from a goal state d^* are shown in Table 4.4. Both F and $\bar{F}$ overestimate the largest distance d^*, except for $\bar{F}$ in three domains (Blocksworld, Scanalyzer, and VisitAll). As discussed in Section 3.1.2, this is desirable since random walk rollouts do not follow the shortest paths.

The right-hand side of Table 4.4 gives the number of expanded states for the three settings of L. We see that limits F and $\bar{F}$ perform better than the fixed limit 200, with F best on one, $\bar{F}$ on four, and 200 on two tasks. Also, when a limit of 200 is best, F presents the closest results, but when F or $\bar{F}$ are best, a limit 200 can be much worse. Note that F is the best only in the domain where $\bar{F}$ underestimates d^*. To validate this, we set $L = \lceil c\bar{F} \rceil$ for $c \in \{1.25, 1.5, 2, 2.5, 3, 3.5\}$ in an additional experiment on domain Blocksworld. The number of expanded states decreases to $c = 3$ with a mean of 52.31 expansions. Overall, the adaptive limits F and $\bar{F}$ are better estimates of the best regression limit.

Table 4.4 – State space information and expanded states of GBFS guided by $\hat{h}$ trained on FSM samples with different regression limits and no cost-to-goal improvements. The value d^* is the distance of the state most distant from a goal state.

Domain	d^*	F	$\bar{F}$	L_{200}	L_F	$L_{\overline{F}}$
Blocks	24	64	17	80.76	**58.06**	185.00
Grid	32	76	44	276.23	316.51	**204.89**
N-Puzzle	31	81	41	177.12	104.52	**80.83**
Rovers	19	32	27	18.85	17.74	**16.50**
Scanalyzer	15	42	20	**53.59**	57.02	88.76
Transport	17	66	35	130.14	82.35	**58.13**
VisitAll	15	31	17	**21.51**	25.61	29.96
Geo. mean				73.12	63.36	69.48

4.2.3 Randomly Generated Samples

In this experiment, we evaluate the effect of adding randomly generated samples to the sample set, as explained in Section 3.1.4. We generate sample sets $S = \{(s_1, h_1), \ldots, (s_N, h_N)\}$ where $10\,\%, 20\,\%, \ldots, 100\,\%$ are random samples and the rest is sampled with FSM and a regression limit $\bar{F}$. No cost-to-goal improvement is applied. Random samples get a cost-to-goal estimate of $H + 1$ where $H = \max_{i \in [N]} h_i$ is the largest h-value in samples S, except when they are part of the samples, in which case they receive the corresponding estimate (this happens in fewer than $1\,\%$ of the samples). Note that when using $100\,\%$ of random samples, each has the cost-to-goal estimate equal to the regression limit $L + 1$ instead of $H + 1$, as we do not have samples in S.

Table 4.5 shows the performance up to $70\,\%$ random samples. We have omitted $80\,\%$, $90\,\%$, and $100\,\%$ since expansions are higher (respectively 56.10, 73.94, and 11397.79). The number of expansions is considerably reduced when using random samples, with $20\,\%$ random samples performing slightly better than other percentages. This phenomenon also holds for individual domains, except Transport which expands on average a few states more, and N-Puzzle and Rovers, which expand a few states less. The results only degrade significantly from $60\,\%$, which makes $50\,\%$ a good choice when considering computational resources, as random samples are computationally cheaper to generate than regression samples.

To better understand the effect of random samples, we have performed three additional experiments with $20\,\%$ of random samples. The first focuses on cost-to-goal estimates. We keep the samples but replace $H + 1$ with small values: a random h-value from the sample set S or a random value drawn from $U[1, 5]$. This modification leads to

Table 4.5 – Expanded states of GBFS guided by $\hat{h}$ trained on FSM samples with regression limit $\bar{F}$, both cost-to-goal improvement strategies, and a varying percentage of randomly generated samples.

	Percentage of random samples							
Domain	0	10	20	30	40	50	60	70
Blocks	177.88	59.23	57.00	63.75	63.25	**48.99**	65.26	74.47
Grid	124.89	**60.77**	66.52	79.81	64.33	66.87	75.92	115.44
N-Puzzle	89.47	87.60	**80.93**	88.76	86.64	81.62	89.38	96.26
Rovers	17.03	14.16	**13.45**	13.63	13.60	14.34	14.23	16.38
Scanalyzer	55.29	37.88	**28.34**	33.74	32.37	42.94	28.85	35.83
Transport	**22.90**	24.58	25.95	27.44	33.02	34.53	43.36	52.36
VisitAll	30.90	22.70	**21.78**	22.05	22.11	23.11	24.45	25.75
Geo. mean	53.91	36.97	35.13	38.51	37.95	38.76	40.94	48.75

overall means of 295.65 and 3832.14 expanded states, respectively. The second experiment changes the distribution of the random samples: we force them to be part of the FSS. This approach leads to a mean of 36.90 expanded states. Finally, the third experiment does not apply mutexes, leading to a mean of 36.11 expanded states. From these additional experiments, it is clear that the most relevant factor is a high h-value, and the distribution and quality of the states seem to matter less. Overall, the most probable explanation for the effect of random samples is that they help to increase the probability that the search is guided towards samples for which the network has learned good estimates, i.e., the samples obtained through regression.

4.2.4 State Completion

Here we focus on how sampled partial states are converted to complete states. In this experiment, all the samples have perfect cost-to-goal estimates h^*. We compare three different state completion strategies for a partial state s. All of them select a random state from the set of states represented by s or a restriction of it: the set equals either to all states in s (no restrictions), only those states that satisfy mutexes, or only states from the forward state space (ideal baseline).

Table 4.6 presents the expanded states for these approaches. Applying mutexes has a moderate effect and is very close to an ideal completion of the states. However, completing randomly also presents competitive results, except for N-Puzzle. Also, N-Puzzle with both restrictions should have similar results, as for this particular domain, all

Table 4.6 – Expanded states of GBFS guided by $\hat{h}$ trained on FSM samples with $L_{\overline{F}}$, h^* cost-to-goal estimates, and different state completion techniques.

	Restriction		
Domain	None	Mutex	FSS
Blocks	212.30	207.18	**190.96**
Grid	104.34	96.02	**87.80**
N-Puzzle	225.62	**79.42**	98.71
Rovers	11.50	11.74	**10.65**
Scanalyzer	**41.20**	43.98	44.61
Transport	17.98	17.86	**16.65**
VisitAll	**22.17**	22.44	22.78
Geo. mean	51.37	44.15	43.57

partial states completed respecting the mutexes are part of the FSS, but this is not the case due to noisy training (detailed in Section 4.5.2). To confirm this, we ran 900 experiments, with 30 sample seeds and 30 network seeds, for N-Puzzle “Mutex” and “FSS”, and we achieved similar mean expansions of 84.4 and 88.66, respectively.

4.2.5 Quality of Estimates

Now, we compare the quality of the cost-to-goal estimates to h^* with and without cost-to-goal improvement techniques and distinct regression limits, as shown in Table 4.7. This table shows the mean absolute difference between the sample estimates and h^*, so smaller means indicate better approximations. Note that we are not evaluating an NN's output but the sample set's cost-to-goal estimates.

The improvement strategies SAI and SUI substantially reduce the estimates for all regression limiting methods. For L_{200}, L_F, and $L_{\overline{F}}$, using only SAI reduces the estimates to 31.28, 13.95, and 4.93, respectively, and using only SUI reduces the estimates to 11.1, 5.48, and 1.93 respectively. Thus, SUI has the most effect on improving the cost-to-goal estimates compared to SAI.

The adaptive regression limiting methods are superior to the fixed default L_{200}, and $L_{\overline{F}}$ has the best results. When comparing L_{200} to $L_{\overline{F}}$ without cost-to-goal improvements, the estimate difference to h^* decreases by about six times on geometric mean. Blocksworld has the best performance, improving more than 25 times. Finally, using both cost-to-goal improvements and rollout limit $L_{\overline{F}}$ reduces the difference to h^* from 33.45 to only 1.60.

Table 4.7 – Mean difference of the cost-to-goal estimates of samples of the sample set to h^*.

	No improvements			With SAI and SUI		
Domain	L_{200}	L_F	$L_{\overline{F}}$	L_{200}	L_F	$L_{\overline{F}}$
Blocks	24.01	12.90	0.91	12.56	6.90	**0.18**
Grid	13.60	13.29	9.84	1.32	1.32	**0.61**
N-Puzzle	70.87	21.80	6.10	60.79	19.18	**5.11**
Rovers	19.92	11.58	9.74	6.70	5.29	**4.88**
Scanalyzer	81.35	15.07	6.09	20.16	5.59	**1.89**
Transport	79.06	23.42	10.98	24.59	5.98	**2.44**
VisitAll	15.80	9.07	4.58	5.64	4.06	**2.15**
Geo. mean	33.45	14.56	5.56	10.95	5.35	1.60

Table 4.8 – Mean difference of h^{FF}, h^{GC} and $\hat{h}$, to h^* when evaluated over the forward state space.

Domain	h^{FF}	h^{GC}	$\hat{h}_0$	$\hat{h}_{L_{200}}$	$\hat{h}_{L_F}$	$\hat{h}_{L_{\overline{F}}}$	$\hat{h}^{20\%}_{L_{\overline{F}}}$
Blocks	6.76	13.37	26.46	16.58	9.84	2.91	**2.42**
Grid	3.72	13.78	26.85	4.21	4.10	**2.73**	9.78
N-Puzzle	**4.19**	14.86	79.84	65.37	23.90	6.75	12.73
Rovers	**0.17**	3.31	11.08	3.18	3.04	2.98	6.35
Scanalyzer	2.78	**1.08**	106.37	27.60	11.45	2.99	9.01
Transport	**1.13**	8.63	109.77	33.53	12.54	7.05	14.89
VisitAll	**1.31**	3.03	21.55	7.50	5.56	2.21	4.74
Geo. mean	1.84	5.92	39.80	13.91	8.13	3.57	7.40

4.2.6 Evaluation over the Forward State Spaces

We now analyze the quality of our learned heuristics and the traditional heuristics FF h^{FF} (HOFFMANN; NEBEL, 2001) and goal-count h^{GC} over all states from the forward state space of each task. Table 4.8 shows the results. Except for the baseline $\hat{h}_0$, the samples are generated with FSM limited by L_{200}, L_F, or $L_{\overline{F}}$ and improved with SAI and SUI. The learned heuristic $\hat{h}^{20\%}_{L_{\overline{F}}}$ is the same as $\hat{h}_{L_{\overline{F}}}$, but $20\,\%$ of the samples are randomly generated. We see that $\hat{h}_{L_{\overline{F}}}$ reduces the difference of the predicted h-value to the real one by about 11 times when compared to $\hat{h}_0$. When compared to h^{GC}, it has the smallest difference in all domains except Scanalyzer. Also, the heuristic $\hat{h}_{L_{\overline{F}}}$ presents a similar mean difference to h^{FF}. Due to the randomly generated samples in the sample set, $\hat{h}^{20\%}_{L_{\overline{F}}}$ doubles the difference compared to $\hat{h}_{L_{\overline{F}}}$. The only domain with a lower value is Blocksworld, driven by approximately two-thirds of the FSS states having an h^*-value within a range of two or less the value assigned for random samples, thus improving the average.

Table 4.9 – Expanded states of GBFS with different heuristic functions. The "h^*" column is ideal and only used for comparison.

Domain	h^*	h^{FF}	h^{GC}	$\hat{h}_0$	$\hat{h}_{L_{\overline{F}}}$	$\hat{h}^{20\%}_{L_{\overline{F}}}$	$\hat{h}^{20\%}_{L_{\overline{F}}\text{ / RW}}$
Blocks	19.44	183.00	332.66	87.00	177.88	**57.00**	106.55
Grid	20.80	**33.62**	265.58	304.22	124.89	66.52	56.76
N-Puzzle	22.56	139.86	818.70	246.77	89.47	**80.93**	83.50
Rovers	10.32	**11.46**	61.48	21.89	17.03	13.45	14.01
Scanalyzer	9.16	28.54	31.94	70.62	55.29	**28.34**	33.05
Transport	13.30	**17.82**	200.50	110.89	22.90	25.95	26.68
VisitAll	11.90	27.26	**16.70**	21.99	30.90	21.78	21.68
Geo. mean	14.53	38.98	124.95	81.86	53.91	35.13	38.92

Table 4.10 – Expanded states of GBFS with $\hat{h}_{L_{\overline{F}}}$ trained with the number of samples corresponding to some percentage of the number of states in the FSS of each task.

Domain	5 %	25 %	50 %	100 %
Blocks	30.21	20.90	20.40	**20.02**
Grid	132.45	99.87	68.85	**43.66**
N-Puzzle	37.31	27.17	25.58	**24.04**
Rovers	16.23	14.47	**14.45**	19.58
Scanalyzer	18.23	**12.86**	12.89	13.33
Transport	20.39	18.42	16.87	**15.81**
VisitAll	22.18	**19.05**	20.23	21.46
Geo. mean	29.62	23.28	21.72	21.10

Comparing Tables 4.7 and 4.8, we observe that relative order between L_{200}, L_F, and $L_{\overline{F}}$ is preserved. The mean difference of the samples' estimates to h^* for $L_{\overline{F}}$ is 1.60 in Table 4.7, and when the corresponding $\hat{h}_{L_{\overline{F}}}$ is required to generalize over the entire FSS, the mean difference is 3.57.

4.2.7 Comparison to Traditional Heuristic Functions

We now compare the learned heuristics with traditional ones. The number of expanded states of GBFS guided by different heuristic functions is shown in Table 4.9. The NNs are trained with samples obtained with FSM (except $\hat{h}^{20\%}_{L_{\overline{F}}\text{ / RW}}$ which was sampled by random walk), both cost-to-goal improvement strategies, and regression limit $\bar{F}$.

First, we see that the baseline $\hat{h}_0$ expands fewer states than h^{GC} in most domains except Grid, Scanalyzer, and VisitAll, but it is far worse than h^{FF} except in Blocksworld and VisitAll, where the learned heuristic has particularly good results. We also see that

$\hat{h}_{L_{\overline{F}}}$ expands less than $\hat{h}_0$ in five domains. h^{FF} has better results than $\hat{h}_{L_{\overline{F}}}$; however, $\hat{h}_{L_{\overline{F}}}$ surpasses h^{FF} if $20\,\%$ of the samples are randomly generated, or if we increase the budget of $\hat{h}_{L_{\overline{F}}}$ to $5\,\%$ (instead of $1\,\%$) of the number of states in the FSS as shown in Table 4.10. This table also indicates that after increasing the budget to $50\,\%$ of the number of states in the FSS, the gains in quality of the learned heuristic are negligible. Additionally, by comparing Tables 4.9 and 4.10, we see that having fewer samples but perfect estimates h^* has better results than having more samples but h^* estimates, meaning that improving the cost-to-goal estimates is more important than having more samples.

We see that for the learned heuristics, the order of approaches in terms of h-h^* difference (Table 4.8) and expanded states (Table 4.9) remains consistent for $\hat{h}_0$ and $\hat{h}_{L_{\overline{F}}}$, but not for $\hat{h}^{20\%}_{L_{\overline{F}}}$, which has a higher mean difference than $\hat{h}_{L_{\overline{F}}}$ but presents the least state expansions, even when compared to h^{FF}. We experimented with the learned heuristic using all our techniques, except for the RW sampling algorithm, to observe the performance gain of our proposed FSM algorithm ($\hat{h}^{20\%}_{L_{\overline{F}}}$) over the RW algorithm ($\hat{h}^{20\%}_{L_{\overline{F}}/\text{RW}}$). We see that RW's NN outperforms FSM's by ten expansions in Grid but achieves approximately double the expanded states in Blocksworld. Overall, sampling with FSM slightly outperforms RW in other domains, with a geometric mean of $10\,\%$ fewer expansions.

With these results, we conclude that a better generalization over the forward state space is good for the samples obtained during regression. In contrast, despite worsening the mean difference to the FSS, random samples are obtained after the regression procedure and can be helpful due to the reasons discussed in Section 4.2.3. Thus, a smaller h-h^* difference is not a definitive indication of good search quality.

4.3 Large State Spaces

The main goal of the following experiments is to validate our findings from the previous sections on large state spaces, so we compare different configurations of the improved methods with traditional heuristics and a baseline. We report results over 9 seeds (3 network seeds and 3 sample seeds), and using 50 initial states from each of Ferber et al. (2022) moderate tasks, which are the IPC tasks from ten selected domains that according to their results are solvable by GBFS with h^{FF} within 1 to 900 seconds. Each domain has the following number of tasks: Blocksworld, 5; Depot, 6; Grid, 2; N-Puzzle, 8; Pipesworld-NoTankage, 10; Rovers, 8; Scanalyzer, 6; Storage, 4; Transport, 8; VisitAll, 6.

Table 4.11 – Mean coverages and expanded states of the learned heuristics with regression limits and their respective approaches not using mutexes ($\hat{h}'$). Expanded states consider only the initial states solved by all heuristics; Grid, N-Puzzle, and Storage had no common solved initial state. The geometric mean is used for the overall mean of expanded states.

	Coverage (%)				Expanded states			
Domain	$\hat{h}_{L_F}$	$\hat{h}_{L_{\overline{F}}}$	$\hat{h}'_{L_F}$	$\hat{h}'_{L_{\overline{F}}}$	$\hat{h}_{L_F}$	$\hat{h}_{L_{\overline{F}}}$	$\hat{h}'_{L_F}$	$\hat{h}'_{L_{\overline{F}}}$
Blocks	93.33	96.31	**100.00**	**100.00**	459205	15279	8927	**8400**
Depot	81.26	84.67	85.96	**86.41**	**45002**	49629	55136	69342
Grid	76.56	**81.11**	41.33	38.33	-	-	-	-
N-Puzzle	21.64	**95.50**	7.42	35.33	-	-	-	-
Pipes-NT	17.80	17.78	17.82	**18.07**	**193039**	211291	209366	205928
Rovers	**13.92**	13.33	13.67	13.53	203	115	106	**99**
Scanalyzer	66.44	**67.78**	66.15	66.67	12793	415	17475	**367**
Storage	3.94	**9.11**	1.67	8.33	-	-	-	-
Transport	75.53	85.97	81.42	**87.50**	96642	**46120**	76756	65656
VisitAll	**92.81**	80.52	92.70	79.00	**11515**	20910	12972	47916
Mean	54.32	63.21	50.81	53.32	27378	9574	15230	10461

We generate samples within one hour and set one hour as the maximum training time. Each of the 50 initial states must be solved separately with GBFS within 5 minutes and $2\,$GB RAM. Generally, more samples yield better results; however, because we do not know how much time will be spent on the cost-to-goal improvement SUI stage as it is done after regression, we fix the number of samples at $N = 16\text{M}/|\mathcal{V}|$, which results in a mean of $500\,$MB RAM during sampling and $2\,$GB during SUI.

First, we reassess our previous results using the regression limits L_F and $L_{\overline{F}}$ on large state spaces since our previous experiments (Section 4.2.2) produced similar results. Table 4.11 shows the mean coverage and number of expanded states for the methods using L_F or $L_{\overline{F}}$, with both cost-to-goal improvements. In addition, we explore logic-independent variants (denoted by $\hat{h}'$) that use the state completion technique without mutexes since we want to assess the performance of learning over samples generated with or without information from the task.

When comparing the learned heuristic $\hat{h}_{L_{\overline{F}}}$ over $\hat{h}_{L_F}$, we see a mean coverage improvement of about $9\,\%$. All domains are improved or have very similar results, except VisitAll, where limiting the regression limit by L_F is better – this is also observed in the small state space experiment. Without mutexes, the coverage improvements from $L_{\overline{F}}$ over L_F are minor. However, the smaller number of expanded states in $\hat{h}'_{L_{\overline{F}}}$ indicates samples of higher quality, which achieves expansions close to when using mutexes. With or without mutexes, using $L_{\overline{F}}$ has the highest positive effect in N-Puzzle, increasing its coverage by about four times. Also, not using mutexes improves Blocksworld, Depot, and Transport results while having a minimal effect on Pipesworld-NoTankage, Rovers, Scanalyzer, and VisitAll. This result suggests that logic-independent approaches show potential in these domains. Based on the results, we conclude that $L_{\overline{F}}$ performs better than L_F for large state spaces. Therefore, the following experiments will use $L_{\overline{F}}$.

Next, we compare the traditional heuristics h^{FF} and h^{GC}, the baseline $\hat{h}_0$, and our best approach $\hat{h}^{20\%}_{L_{\overline{F}}}$ in Table 4.12. We see that h^{FF} dominates in most domains, achieving twice the mean coverage of the baseline $\hat{h}_0$. However, $\hat{h}^{20\%}_{L_{\overline{F}}}$ has only $12\,\%$ less mean coverage than h^{FF}, with competitive coverage in most domains and improving $\hat{h}_0$ by about $31\,\%$. Note that $\hat{h}^{20\%}_{L_{\overline{F}}}$ achieves better mean coverage than h^{GC}, with higher or equal coverage in 6 out of 10 domains. Also, in all domains except Transport, the best-learned heuristic $\hat{h}^{20\%}_{L_{\overline{F}}}$ expands fewer states when compared to h^{FF}, indicating that the learned heuristic is more informed and that the inferior coverage is an effect of the slower expansion speed of the learned heuristics. However, the expanded states are biased towards easier tasks, as they refer to commonly solved initial states across the approaches. Furthermore, when limiting h^{FF} by the same number of expansions as the learned heuristic, h^{FF} achieves coverage of 81.20, meaning that it still excels in most states. Because the dataset used contains only tasks that are solvable by h^{FF} within 900 seconds, the results are also biased towards better performance with a search guided by h^{FF}.

When comparing Tables 4.11 and 4.12, we notice that all learned heuristics have similarly poor results in Rovers, independent of configuration. Considering only the learned heuristics, when using $20\,\%$ of random samples ($\hat{h}^{20\%}_{L_{\overline{F}}}$) instead of $0\,\%$ ($\hat{h}_{L_{\overline{F}}}$), there are intermediate improvements of about $15\,\%$ in Storage, Transport, and VisitAll, and a significant improvement in Pipesworld-NoTankage, from approximately $18\,\%$ to $80\,\%$ coverage.

4.4 Comparison to Other Approaches

Although we do not aim to systematically compare other methods due to distinct machine configurations or libraries, we try to qualitatively compare our approach with Ferber et al. (2022) and O'Toole et al. (2022). All methods share the same dataset and NN configuration – differences include batch size, patience value, NN initialization functions, and percentages of data split into training and validation sets.

Regarding our method, the sampling and training procedures can take a combined

Table 4.12 – Mean coverages and expanded states of the traditional heuristics h^{FF} and h^{GC} compared to the baseline learned heuristic $\hat{h}_0$ and the best learned heuristic $\hat{h}^{20\%}_{L_{\overline{F}}}$, obtained via training over samples with FSM, $L_{\overline{F}}$, 20 % of random samples, and both cost-to-goal improvement strategies. Expanded states consider only the initial states solved by all heuristics; N-Puzzle and Storage had no common solved initial state. The geometric mean is used for the overall mean of expanded states.

	Coverage (%)				Expanded states			
Domain	h^{FF}	h^{GC}	$\hat{h}_0$	$\hat{h}^{20\%}_{L_{\overline{F}}}$	h^{FF}	h^{GC}	$\hat{h}_0$	$\hat{h}^{20\%}_{L_{\overline{F}}}$
Blocks	**100.00**	**100.00**	**100.00**	**100.00**	36482	87026	**5349**	7335
Depot	**94.33**	80.00	57.19	89.26	137053	874333	35972	**29984**
Grid	**94.00**	51.00	38.11	60.33	45117	29990	**15040**	27009
N-Puzzle	**92.50**	4.00	13.75	86.81	-	-	-	-
Pipes-NT	63.40	89.40	13.51	**79.84**	83825	**10416**	304764	17389
Rovers	**85.50**	66.00	13.53	15.39	38	466	596	**30**
Scanalyzer	**100.00**	**100.00**	59.70	73.67	408	3914	27570	**300**
Storage	**33.00**	13.50	1.94	27.67	-	-	-	-
Transport	**100.00**	**100.00**	48.89	**100.00**	**8342**	253501	139311	12149
VisitAll	92.00	**100.00**	74.19	98.85	248269	**337**	80155	5110
Mean	85.47	70.39	42.08	73.18	12529	15706	25184	3937

time of up to 2 hours, and we use a search time limit of 5 minutes. Ferber et al. (2022) perform sampling and training for up to 28 hours, with a search time limit of 10 hours. O'Toole et al. (2022) spend an unreported amount of time to generate 100 K samples and an average of 23 minutes in training, with a search time-limit of 6 minutes. Ferber et al. (2022) use a validation method that consists in retraining the network up to three times if the learned heuristic is not able to solve with GBFS more than 80 % of generated validation states with a search time-limit of 30 minutes. O'Toole et al. (2022) train 10 networks for each state space and select the one with the best-performing heuristic according to the same validation method as Ferber et al. (2022).

Considering only the best configurations, Ferber et al. (2022) and O'Toole et al. (2022) perform backward search-based sampling using random walks while we combine breadth-first search with random walk. Ferber et al. (2022) obtain states through regression and try to solve them using GBFS with the current heuristic to obtain the plans used as training data. They use bootstrapping, which consists of improving the learned heuristics by training with samples of increasing difficulty, starting with a random walk limit of 5 and doubling it (up to 8 times) whenever GBFS finds a plan for at least 95 % of the states obtained through regression. O'Toole et al. (2022) perform 5 rollouts with a regression limit of $L = 500$ and use the current depth as cost-to-goal estimates; they use

Table 4.13 – Mean coverage results of $\hat{h}^{\text{Boot}}$ (FERBER et al., 2022) and $\hat{h}^{\text{N-RSL}}$ (O'TOOLE et al., 2022), with results obtained from their respective papers, and our best learned heuristic trained with 100 K samples, from which 0 %, 20 % and 50 % are randomly generated.

Domain	$\hat{h}^{\text{Boot}}$	$\hat{h}^{\text{N-RSL}}$	$\hat{h}_{L_{\overline{F}}}$	$\hat{h}^{20\%}_{L_{\overline{F}}}$	$\hat{h}^{50\%}_{L_{\overline{F}}}$
Blocks	0.00	91.50	99.96	**100.00**	**100.00**
Depot	32.00	58.80	80.56	**91.52**	81.78
Grid	**100.00**	60.30	59.89	57.56	54.44
N-Puzzle	27.00	18.90	70.94	85.92	**86.36**
Pipes-NT	36.00	69.60	19.00	91.51	**96.62**
Rovers	**36.00**	12.50	13.47	14.33	14.50
Scanalyzer	33.00	**94.10**	67.00	66.67	66.15
Storage	**89.00**	16.40	19.67	22.00	19.17
Transport	84.00	70.80	**100.00**	99.67	86.89
VisitAll	17.00	95.40	84.52	99.04	**99.70**
Mean	45.40	58.80	61.50	72.82	70.56

the Tarski planning framework (FRANCÉS; RAMIREZ, 2018) to perform the regression procedure as opposed to Fast Downward and developed a cost-to-goal improvement techniques equivalent to SAI, and 50 % of the samples are randomly generated as described in Section 3.1.4. All methods have a Boolean representation for the samples and use mutexes (obtained from Fast Downward) to complete undefined variables.

We now compare the coverage results between all methods over the same tasks. We perform an additional set of experiments to show in Table 4.13. We present the results of our best method using 0 %, 20 % and 50 % of random samples, obtained from training over 100 K samples – same quantity as O'Toole et al. (2022). We also show results reported by Ferber et al. (2022) $\hat{h}^{\text{Boot}}$ and O'Toole et al. (2022) $\hat{h}^{\text{N-RSL}}$, both trained without validation. Note that although the dataset is the same, the results are not fully comparable due to different machine configurations and time dedicated to sampling, training, and testing. Considering coverages, we notice that our methods have results more similar to $\hat{h}^{\text{N-RSL}}$ than to $\hat{h}^{\text{Boot}}$, and higher coverage, except for Grid, Rovers, Scanalyzer, and Storage. Generating samples only through regression (i.e., without solving states) and training afterward is faster when compared to bootstrapping. A significant limitation of Ferber, Helmert and Hoffmann (2020) is the high cost of generating samples, as the states generated by the backward random walk must be solved with the currently learned heuristic to produce plans used as samples. (An alternative to computationally cheaper bootstrap is proposed by Lelis (2013).) Both $\hat{h}^{\text{N-RSL}}$ and our methods suggest that sampling using regression with improvement strategies (such as SAI, SUI, and random samples) gives competitive results in most domains.

According to O'Toole et al. (2022), the proportion of random samples in the sample set has the most positive effect on coverage – approximately doubling it when going from $0\,\%$ to $50\,\%$ of random samples (34.7 vs. 59.9, from their supplementary material). As seen in Table 4.13, we also notice an improvement from using random samples, although smaller. This improvement is more prominent in O'Toole et al. (2022), most likely due to their small quantity of rollouts (only five), negatively influencing sample diversity, which is compensated by adding random samples. Our experiments show that all domains either improve or have similar results, and the mean coverage improves by about $10\,\%$. Also, except for Pipesworld-NoTankage, we saw no improvements above $5\,\%$ using $50\,\%$ of random samples compared to $20\,\%$.

4.5 Limitations

This chapter discusses the problems and limitations of methods based on learning heuristics with neural networks. Understanding and addressing these limitations can help advance the field and develop more robust and effective learned heuristic functions.

4.5.1 Validation Loss

A significant limitation is the unreliability of the validation loss as an indicator of performance during the training. We observe that two NNs trained with identical configuration and regime, varying only in the training seed, produce distinct validation losses, where the network with a larger loss may outperform the one with a smaller loss in terms of coverage or number of states expanded during the search. For instance, when training with h^*-values over FSM samples, with one sample seed and 100 network seeds for each of the small state spaces, $36\,\%$ of the models with the least amount of expanded states had a higher validation loss than other models with more expanded states; if the states do not have h^*-values, the percentage increases to $49\,\%$. Regarding coverage, in the large state space experiments with the heuristic $\hat{h}^{20\%}_{L_F^-}$ and 9 runs (3 sample seeds and 3 network seeds) for each state space, $15\,\%$ of the models with the highest coverage had a higher validation loss than the other models with lower coverage. This phenomenon happens because the sampling procedure is imperfect – an NN learns a limited portion of the state

Table 4.14 – Expanded states with GBFS and their standard deviations in small state space experiments using the baseline $\hat{h}_0$ and the best heuristic $\hat{h}^{20\%}_{L_{\overline{F}}}$.

Domain	$\hat{h}_0$	$\hat{h}^{20\%}_{L_{\overline{F}}}$
Blocks	87.00 ± 26.24	57.00 ± 14.51
Grid	304.22 ± 177.52	66.52 ± 32.47
N-Puzzle	246.77 ± 133.26	80.93 ± 11.73
Rovers	21.89 ± 4.18	13.45 ± 0.78
Scanalyzer	70.62 ± 38.00	28.34 ± 7.86
Transport	110.89 ± 24.30	25.95 ± 2.33
VisitAll	21.99 ± 3.16	21.78 ± 2.30
Geo. mean	81.86 ± 25.78	35.13 ± 5.64

space, which varies per sample seed, so even if the validation loss is small, it provides no guarantees of search quality.

This phenomenon challenges model validation methods and clarifies that a lower validation loss does not necessarily translate to a better heuristic function in the search. Consequently, relying solely on validation loss to measure model quality can lead to misleading conclusions and suboptimal decisions about when to stop training or which NN is most effective for the search. Using multiple seeds can work around the problem. However, alternative evaluation metrics and techniques must be explored to more accurately assess the performance and effectiveness of learned heuristic models before the search.

4.5.2 Noisy Training

Another limitation to consider is the influence of seed initialization on the training. As noted by other methods that use some form of model validation (FERBER; HELMERT; HOFFMANN, 2020; SHEN; TREVIZAN; THIEBAUX, 2020; FERBER et al., 2022; O'TOOLE et al., 2022), training can be noisy, with multiple runs leading to very different results. A consequence of this is observed in our experiments comparing expanded states, where even a single state that ends up expanded due to an inaccurate heuristic can lead to numerous extra expansions. In contrast, other states can lead the search to a more direct path to the goal. This problem is common for all approaches that use GBFS, but this is further aggravated in NNs that learn from approximated values.

Table 4.14 compare the mean number of expansions of the baseline $\hat{h}_0$ and $\hat{h}^{20\%}_{L_{\overline{F}}}$. We see that standard deviation varies per domain, i.e., some domains are noisier than others, and training with better-quality samples typically helps. Regarding coverage in large state spaces, the standard deviation was smaller, as it does not vary at the same rate as expanded states. For example, $\hat{h}^{20\%}_{L_F^-}$ has a Grid coverage of 60 with a standard deviation of 16, while the mean standard deviation considering all the other domains is less than 3.

4.5.3 State Representation

This work and others (FERBER; HELMERT; HOFFMANN, 2020; FERBER et al., 2022; O'TOOLE et al., 2022) use the same STRIPS representation, and the NN receives as input a vector of Boolean values representing the set of facts of a complete state, where each input neuron corresponds to a fact. However, this can become inefficient when training over large tasks as the input size of the NN grows linearly with the number of facts in the task. Furthermore, sampling methods via regression generate partial states. With the assignment of undefined variables, part of the sampling information is lost.

Yu, Kuroiwa and Fukunaga (2020) use the same STRIPS representation with a Boolean input for NN but does not complete the undefined variables, assigning false to all $|D(v)| = n$ facts of each undefined variable v. Since their undefined variables are represented in SAS$^+$, we can infer that at least one of the n facts is true in a complete state. Consequently, by assigning false to all undefined facts, the NN is trained using states that are unreachable during the search process.

On the other hand, Yu, Kuroiwa and Fukunaga (2020) and Geissmann (2015) address the SAS$^+$ representation. They use a multivalued vector to represent a state, which aligns with the internal state representation in Fast Downward. This choice allows the representation of undefined values and increases the speed of the NN by reducing the dimensionality of the input layer. They discuss that the choice may depend on the domain, and the performance can vary across different domains. However, on average, this approach has shown worse performance than the Boolean representation in experiments on 46 domains (YU; KUROIWA; FUKUNAGA, 2020).

5 CONCLUSION

We have presented a study of sample generation and correction strategies for training FNNs to learn heuristic functions for classical planning. We have revised existing approaches to sample generation and proposed a new strategy that uses regression with BFS and random walks and several techniques that improve cost-to-goal estimates. By revising and refining existing sample generation methods, we have successfully enhanced the overall performance of the learned heuristic functions, achieving nearly double the coverage compared to our baseline.

Among our contributions, the cost-to-goal improvement technique SUI and the adaptive regression limit $\bar{F}$ have the most positive effects on sampling quality. The former improves the accuracy of cost-to-goal estimates by analyzing the successors of a state, while the latter avoids overestimates by limiting the maximum regression limit so that we have a sampling distribution within a reasonable cost-to-goal range. Also, one of our main findings is that having fewer samples with more accurate h-values is better than having more samples with inaccurate h-values. Furthermore, having multiple random walk rollouts – especially when combined with BFS – generates samples with better quality when compared to using only BFS or DFS.

Finally, a systematic analysis of small state spaces against ideal baselines seems to indicate that: a) for the samples obtained through regression, a distribution covering various portions of the state space without repeated samples close to the goal works best, b) both the sample size and reasonable cost-to-goal estimates contribute to search performance, with the latter being more important, c) enough samples of good quality translate to good search performance that can be compared to traditional heuristics, although logic-independent approaches (e.g., without mutexes) are currently not as good as logic-dependent ones.

Future works can be conducted to investigate the scalability of the proposed techniques, as well as approaches that address the limitations described in Section 4.5 can be explored. Additionally, compared to traditional heuristics, some domains have poor results with learned heuristics, such as Rovers, which, as far as we know, has a low coverage among all NN-based methods. It needs to be clarified why some domains perform well and others poorly among different learned heuristics.

We validate the findings of O'Toole et al. (2022) that including randomly generated samples in the sample set has a consistently positive influence. However, the underlying reasons for this effect remain unexplained. A study on random sampling can be conducted to understand its effect when combined with other sampling methods. Still, exploring new approaches to determine the cost-to-goal estimate in random samples as an alternative to the current arbitrary value could improve performance.

Furthermore, sampling approaches involving novelty can be promising. O'Toole et al. (2022) proposed generating states that maximize the number of undiscovered facts, i.e., facts that have not been observed in any state of the current rollout, during the random walk steps. While their approach does not yield significant gains, other novelty or information gain methods can be explored.

Finally, heuristic functions with logic-independent approaches seem promising. The experiments demonstrate competitive results compared to the mutex-based one, with a coverage difference of less than $4\,\%$ in 8 out of 10 domains addressed. These findings indicate the potential of logic-independent approaches, such as those with black-box interfaces.

REFERENCES

AGOSTINELLI, F.; MCALEER, S.; SHMAKOV, A.; BALDI, P. Solving the Rubik's Cube with Deep Reinforcement Learning and Search. **Nature Machine Intelligence**, v. 1, n. 8, p. 356–363, 2019.

ARFAEE, S. J.; ZILLES, S.; HOLTE, R. C. Learning Heuristic Functions for Large State Spaces. **Artificial Intelligence**, v. 175, n. 16-17, p. 2075–2098, 2011.

BÄCKSTRÖM, C.; NEBEL, B. Complexity Results for SAS$^+$ Planning. **Computational Intelligence**, v. 11, n. 4, p. 625–655, 1995.

BATTAGLIA, P. W.; HAMRICK, J. B.; BAPST, V.; SANCHEZ-GONZALEZ, A.; ZAMBALDI, V.; MALINOWSKI, M.; TACCHETTI, A.; RAPOSO, D.; SANTORO, A.; FAULKNER, R.; GULCEHRE, C.; SONG, F.; BALLARD, A.; GILMER, J.; DAHL, G.; VASWANI, A.; ALLEN, K.; NASH, C.; LANGSTON, V.; DYER, C.; HEESS, N.; WIERSTRA, D.; KOHLI, P.; BOTVINICK, M.; VINYALS, O.; LI, Y.; PASCANU, R. **Relational inductive biases, deep learning, and graph networks**. 2018.

BONET, B. An Admissible Heuristic for SAS$^+$ Planning Obtained from the State Equation. In: **Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence**. Beijing, China: AAAI Press, 2013. p. 2268–2274.

BYLANDER, T. The computational complexity of propositional STRIPS planning. **Artificial Intelligence**, v. 69, n. 1-2, p. 165–204, 1994.

DONG, H.; MAO, J.; LIN, T.; WANG, C.; LI, L.; ZHOU, D. Neural Logic Machines. In: **6th International Conference on Learning Representations**. Vancouver, Canada: OpenReview.net, 2018.

DORAN, J. E.; MICHIE, D. Experiments with the Graph Traverser program. **Proceedings of the Royal Society A**, v. 294, p. 235–259, 1966.

EDELKAMP, S.; SCHRÖDL, S. **Heuristic Search: Theory and Applications**. [S.l.]: Academic Press, 2012. ISBN 978-0-12-372512-7.

FERBER, P.; GEIßER, F.; TREVIZAN, F.; HELMERT, M.; HOFFMANN, J. Neural Network Heuristic Functions for Classical Planning: Bootstrapping and Comparison to Other Methods. In: **Proceedings of the Thirty-Second International Conference on Automated Planning and Scheduling**. Online: AAAI Press, 2022.

FERBER, P.; HELMERT, M.; HOFFMANN, J. Neural Network Heuristics for Classical Planning: A Study of Hyperparameter Space. In: **24th European Conference on Artificial Intelligence**. Santiago de Compostela, Spain: IOS Press, 2020. p. 2346–2353.

FIKES, R. E.; NILSSON, N. J. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. **Artificial Intelligence**, Elsevier, v. 2, n. 3-4, p. 189–208, 1971.

FRANCÉS, G.; RAMIREZ, M. **Tarski: An AI Planning Modeling Framework**. [S.l.]: GitHub, 2018. <https://github.com/aig-upf/tarski>.

GEHRING, C.; ASAI, M.; CHITNIS, R.; SILVER, T.; KAELBLING, L. P.; SOHRABI, S.; KATZ, M. Reinforcement Learning for Classical Planning: Viewing Heuristics as Dense Reward Generators. In: **Proceedings of the Thirty-Second International Conference on Automated Planning and Scheduling**. Online: AAAI Press, 2022. v. 32, p. 588–596.

GEISSMANN, C. **Learning Heuristic Functions in Classical Planning**. Dissertation (Master) — University of Basel, 2015.

GORI, M.; MONFARDINI, G.; SCARSELLI, F. A New Model for Learning in Graph Domains. In: **Proceedings. 2005 IEEE International Joint Conference on Neural Networks**. Montreal, Canada: IEEE, 2005. v. 2, n. 2005, p. 729–734.

HART, P. E.; NILSSON, N. J.; RAPHAEL, B. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. **IEEE Transactions on Systems Science and Cybernetics**, v. 4, n. 2, p. 100–107, 1968.

HASLUM, P.; GEFFNER, H. Admissible Heuristics for Optimal Planning. In: **Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling**. Whistler, Canada: AAAI Press, 2004. p. 140–149.

HAYKIN, S. **Neural Networks and Learning Machines**. 3. ed. Hamilton, Canada: Pearson, 2009.

HE, K.; ZHANG, X.; REN, S.; SUN, J. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. In: **Proceedings of the 2015 IEEE International Conference on Computer Vision**. Santiago, Chile: IEEE Computer Society, 2015.

HE, K.; ZHANG, X.; REN, S.; SUN, J. Deep Residual Learning for Image Recognition. In: **2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)**. Las Vegas, USA: IEEE Computer Society, 2016. p. 770–778.

HELMERT, M. The Fast Downward Planning System. **Journal of Artificial Intelligence Research**, v. 26, p. 191–246, jul. 2006.

HELMERT, M. Concise Finite-Domain Representations For PDDL Planning Tasks. **Artificial Intelligence**, v. 173, n. 5-6, p. 503–535, 2009.

HOCHREITER, S. The Vanishing Gradient Problem During Learning Recurrent Neural Nets and Problem Solutions. **International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems**, World Scientific, v. 6, n. 02, p. 107–116, 1998.

HOFFMANN, J.; NEBEL, B. The FF Planning System: Fast Plan Generation Through Heuristic Search. **Journal of Artificial Intelligence Research**, v. 14, p. 253–302, 2001.

HOFFMANN, J.; PORTEOUS, J.; SEBASTIA, L. Ordered Landmarks in Planning. **Journal of Artificial Intelligence Research**, v. 22, p. 215–278, 2004.

KARPAS, E.; DOMSHLAK, C. Cost-Optimal Planning with Landmarks. In: **Proceedings of the Twenty-First International Joint Conference on Artificial Intelligence**. Hainan Island, China: AAAI Press, 2009. p. 1728–1733.

KINGMA, D.; BA, J. Adam: A Method for Stochastic Optimization. In: **3rd International Conference on Learning Representations**. San Diego, USA: ICLR, 2015.

LELIS, L. H. S. de. **Cluster-and-Conquer: a Paradigm for Solving State-Space Problems**. Thesis (PhD) — University of Alberta, 2013.

LU, L.; SHIN, Y.; SU, Y.; KARNIADAKIS, G. E. Dying ReLU and Initialization: Theory and Numerical Examples. **Communications in Computational Physics**, v. 28, n. 5, p. 1671–1706, jun. 2020.

O'TOOLE, S.; RAMIREZ, M.; LIPOVETZKY, N.; PEARCE, A. R. Sampling from Pre-Images to Learn Heuristic Functions for Classical Planning. In: **Fifteenth International Symposium on Combinatorial Search**. Vienna, Austria: AAAI Press, 2022. v. 15, n. 1, p. 308–310.

PASZKE, A.; GROSS, S.; MASSA, F.; LERER, A.; BRADBURY, J.; CHANAN, G.; KILLEEN, T.; LIN, Z.; GIMELSHEIN, N.; ANTIGA, L.; DESMAISON, A.; KöPF, A.; YANG, E.; DEVITO, Z.; RAISON, M.; TEJANI, A.; CHILAMKURTHY, S.; STEINER, B.; FANG, L.; BAI, J.; CHINTALA, S. PyTorch: An Imperative Style, High-Performance Deep Learning Library. **Advances In Neural Information Processing Systems**, v. 32, 2019.

SAMADI, M.; FELNER, A.; SCHAEFFER, J. Learning from Multiple Heuristics. In: **Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence**. Chicago, USA: AAAI Press, 2008. p. 357–362.

SCARSELLI, F.; GORI, M.; TSOI, A. C.; HAGENBUCHNER, M.; MONFARDINI, G. The Graph Neural Network Model. **IEEE Transactions on Neural Networks**, v. 20, n. 1, p. 61–80, 2008.

SHEN, W.; TREVIZAN, F.; THIEBAUX, S. Learning Domain-Independent Planning Heuristics with Hypergraph Networks. In: **Proceedings of the Thirty International Conference on Automated Planning and Scheduling**. Online: AAAI Press, 2020. p. 574–584.

STÅHLBERG, S.; BONET, B.; GEFFNER, H. Learning General Optimal Policies with Graph Neural Networks: Expressive Power, Transparency, and Limits. In: **Proceedings of the Thirty-Second International Conference on Automated Planning and Scheduling**. Online: AAAI Press, 2022. v. 32, p. 629–637.

STURTEVANT, N.; HELMERT, M. **Exponential-Binary State-Space Search**. 2019.

TOYER, S.; THIEBAUX, S.; TREVIZAN, F.; XIE, L. ASNets: Deep Learning for Generalised Planning. **Journal of Artificial Intelligence Research**, v. 68, p. 1–68, may 2020.

TOYER, S.; TREVIZAN, F.; THIEBAUX, S.; XIE, L. Action Schema Networks: Generalised Policies With Deep Learning. In: **Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence**. New Orleans, USA: AAAI Press, 2018.

YU, L.; KUROIWA, R.; FUKUNAGA, A. Learning Search-Space Specific Heuristics Using Neural Network. In: **Proceedings of the 12th Workshop on Heuristics and Search for Domain-independent Planning**. Online: ICAPS, 2020.

ZHANG, A.; LIPTON, Z. C.; LI, M.; SMOLA, A. J. **Dive into Deep Learning**. 2021.