

Understanding Sample Generation Strategies for Learning Heuristic Functions in Classical Planning

Rafael V. Bettker
Pedro P. Minini
Andre G. Pereira
Marcus Ritt

RVBETTKER@INF.UFRGS.BR
PPMININI@INF.UFRGS.BR
AGPEREIRA@INF.UFRGS.BR
MARCUS.RITT@INF.UFRGS.BR

Universidade Federal do Rio Grande do Sul, Brazil

Abstract

We study the problem of learning good heuristic functions for classical planning tasks with neural networks based on samples represented by states with their cost-to-goal estimates. The heuristic function is learned for a state space and goal condition with the number of samples limited to a fraction of the size of the state space, and must generalize well for all states of the state space with the same goal condition. Our main goal is to better understand the influence of sample generation strategies on the performance of a greedy best-first heuristic search (GBFS) 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 algorithm used to generate the sample set and how close the sample estimates to the perfect cost-to-goal are. These two factors are dependent: having perfect cost-to-goal estimates is insufficient if the samples are not well distributed across the state space. We also study other effects, such as 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 practical strategies result in a learned heuristic that, when guiding a GBFS algorithm, increases by more than 30% the mean coverage compared to a baseline learned heuristic.

1. Introduction

A planning task is defined by the initial state that describes the starting conditions of the environment, the goal condition that specifies the conditions to be achieved, and the operators that can be applied to states of the planning task. Each operator has preconditions that must be satisfied to be applicable and effects that are the changes that occur in the state after the operator is applied. Planning is finding a sequence of operators that transforms an initial state into one that satisfies the goal condition. Applying an operator has a cost, and the cost of a plan is the total cost of its operators. A plan is optimal if its cost is minimal among all plans. One effective approach for solving planning tasks is to use algorithms from the best-first search family, which rely on a guiding function f that estimates the quality of a state s . Best-first search expands first the state with the best quality. A heuristic function $h(s)$ estimates the cost-to-goal from state s and may be used as a quality measure by f . One algorithm within the best-first search family is greedy best-first search (GBFS) (Doran and Michie 1966), which is only guided by the heuristic function $h(s)$. Another example is the A* search algorithm, guided by $f(s) = g(s) + h(s)$ (Hart et al. 1968), where $g(s)$ is the

current cost to reach the state s from the initial state. Generally, best-first search algorithms are more effective when the heuristic function h better estimates the optimal cost-to-goal, although this is not always the case (Holte 2010).

Some of the most successful heuristic functions h are based on methods that solve a simplified version of the planning task and use the cost to solve the simplified task to produce the cost-to-goal estimate. Examples of methods used to produce heuristic functions include delete relaxation (Bonet and Geffner 2001; Hoffmann and Nebel 2001), landmarks (Helmert and Domshlak 2009; Hoffmann et al. 2004; Karpas and Domshlak 2009), critical paths (Haslum and Geffner 2000), constraints-based (Bonet 2013; van Den Briel et al. 2007), and abstractions (Culberson and Schaeffer 1998; Edelkamp 2001; Helmert et al. 2007). Many of these heuristics come with additional properties, such as admissibility. A heuristic is admissible if it does not overestimate the optimal cost-to-goal. A* guided by admissible heuristic is guaranteed to find an optimal plan if one exists.

Increasing interest in learning heuristic functions with neural networks (NNs) has been driven by rapid progress in other application areas. Some works in this area include those by Agostinelli et al. (2019), Arfaee et al. (2011), Ferber et al. (2020), Ferber et al. (2022), O’Toole et al. (2022), Samadi et al. (2008), Shen et al. (2020), Toyer et al. (2020), and Yu et al. (2020). The basic approach is simple: generate a set of state pairs along with their estimated cost-to-goal values, then train a supervised model on these pairs. This model learns to predict cost-to-goal values for new, unseen states. Since learning heuristics that match the true cost-to-goal may not be optimal, an alternative method involves learning to rank states by optimizing the Kendall rank correlation coefficient during training, as in Garrett et al. (2016). Nonetheless, 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 (progression) or backward search (regression).
- C2) Estimates of cost-to-goal are typically hard to obtain and finding the perfect cost-to-goal amounts to solving the task on the samples.
- C3) Planning domains are very different, and logic-based heuristics, which leverage the logical relationships encoded in the description of the planning task, apply to any domain. This results in the problem of transferring a learned heuristic to new domains, tasks, or state spaces.
- C4) Planners, which are systems designed to find solutions for planning tasks, rely 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.

In this paper, we are interested in strategies for generating samples which we introduce in Section 3, and in particular, what characterizes samples that lead to heuristic functions that perform well. To this end, Section 4 presents a systematic study of the contributions of

each strategy when solving distinct initial states of a single state space. We learn state-space-specific heuristics using a feedforward neural network (FNN), focusing mainly on the quality of the learned heuristic and its influence on the number of expanded states and coverage. We aim to understand better how to learn high-quality heuristics. In experiments on small state spaces in Section 5, 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 6 we compare the best-proposed strategy with a baseline and traditional logic-based heuristics over large state spaces. Furthermore, we qualitatively compare existing methods in the literature in Section 6.1. Finally, we summarize the main findings in Section 7.

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. The 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 (Section 4).

2. Preliminaries

This section introduces the foundational concepts and notations necessary for the discussions that follow. We also provide an overview of the existing literature related to our work.

2.1 Background and Notation

A SAS⁺ planning task (Bäckström and Nebel 1995) is a tuple $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s^*, \text{cost} \rangle$, where \mathcal{V} is a set of variables, \mathcal{O} a set of operators, s_0 an initial state, s^* the goal condition, and $\text{cost} : \mathcal{O} \rightarrow \mathbb{R}_+$ a function mapping operators to costs. A variable $v \in \mathcal{V}$ has a finite domain $D(v)$, and a partial state s is a partial function $s : \mathcal{V} \rightarrow \mathcal{D}$, with $\mathcal{D} = \cup_{v \in \mathcal{V}} D(v)$ and such that $s(v) \in D(v)$, whenever $s(v)$ is defined. Otherwise, $s(v)$ is undefined, written as $s(v) = \perp$. The domain of s is $\text{dom}(s) = \{v \in \mathcal{V} \mid s(v) \neq \perp\}$. We also write $s|_U$ for the restriction of s to a subset $U \subseteq \mathcal{V}$, i.e. the state that agrees with s on U and is undefined otherwise. A (complete) state s is a partial state with $\text{dom}(s) = \mathcal{V}$. The initial state s_0 is a state, and the goal condition s^* is a partial state. A partial state s can also be interpreted as a set of facts $\{(v, s(v)) \mid v \in \text{dom}(s)\}$. We use these interpretations interchangeably, and

their use will be clear from the context. We further write $\mathcal{S}(s)$ for the set of all complete states that agree with partial state s , namely $\mathcal{S}(s) = \{s' \mid \text{dom}(s') = \mathcal{V} \wedge s'|_{\text{dom}(s)} = s\}$. An operator $o \in \mathcal{O}$ is a pair of preconditions and effects $(\text{pre}(o), \text{eff}(o))$, both partial states. Operator o can be applied to state s if $s \supseteq \text{pre}(o)$, and its application produces a successor state $s' = \text{succ}(s, o) := \text{eff}(o) \circ s$, where $s' = t \circ s$ is defined by $s'(v) = t(v)$ for all $v \in \text{dom}(t)$, and $s'(v) = s(v)$ otherwise. The set of all successor states of state s is $\text{succ}(s) = \{\text{succ}(s, o) \mid o \in \mathcal{O}, o \text{ applicable to } s\}$. A sequence of operators $\pi = (o_1, \dots, o_k)$ with $o_i \in \mathcal{O}$ is valid for initial state s_0 if for $i \in [k]$ operator o_i can be applied to s_{i-1} and produces $s_i = \text{succ}(s_{i-1}, o_i)$. A plan is a valid sequence π for s_0 such that $s_k \supseteq s^*$ and the cost of plan π is $\sum_{i \in [k]} \text{cost}(o_i)$.

The predecessor $\text{pred}(s, o)$ of a partial state s under operator o is obtained by regression. It is again a partial state, and all complete states in $\mathcal{S}(\text{pred}(s, o))$ can reach s by applying operator o . For regression, we consider an operator $o \in \mathcal{O}$ to be relevant for partial state s if $\text{eff}_r = \text{dom}(\text{eff}(o)) \cap \text{dom}(s) \neq \emptyset$; the operator is consistent if $s \subseteq \text{eff}(o)|_{\text{eff}_r} \circ \text{pre}(o)$. Relevance requires that at least one variable is defined both in the effect and in the partial state to be regressed, and consistency requires an agreement on such variables or preconditions that are maintained. An operator o then is backward applicable in partial state s if it is relevant and consistent with s and leads to predecessor $\text{pred}(s, o) = \text{pre}(o) \circ (s|_{\text{dom}(s) \setminus \text{eff}_r})$. Note that $\text{succ}(\text{pred}(s, o), o) \supseteq s$, but may differ from s . Similar to progression, a partial state s has predecessors $\text{pred}(s) = \{\text{pred}(s, o) \mid o \in \mathcal{O}, o \text{ backward applicable to } s\}$. A regression sequence from state s_0 then is valid if o_i can be applied to s_{i-1} and produces $s_i = \text{pred}(s_{i-1}, o_i)$. All partial states s_i can reach a partial state s sup. Since the goal is a partial state, a valid regression sequence $\rho = (o_1, \dots, o_k)$ will generate a partial state that can reach the goal in at most k steps and with a cost at most $\sum_{i \in [k]} \text{cost}(o_i)$.

Given a planning task, the set of all states over \mathcal{V} is the state space. The forward state space (FS) is the set of all states reachable from the initial state s_0 by applying a sequence of operators. Similarly, the backward state space (BS) is the set of all partial states reachable from the goal s^* by applying a sequence of backward applicable operators. Note that it is possible that the backward state space may contain partial states that cannot be reached by a forward search (also called spurious states, e.g. by Alcazar et al. (2013)). This problem is addressed in Section 3.

2.2 Representing and Generating Samples for Learning Heuristic Functions

A heuristic function $h : \mathcal{S} \rightarrow \mathbb{R}_+ \cup \{\infty\}$ maps each state in the state space \mathcal{S} to a non-negative cost-to-goal estimate (h -value), or to ∞ when the heuristic concludes that the state cannot reach the goal. For example, the goal-count heuristic for a state s and goal condition s^* computes the number of unsatisfied facts of the goal condition by the state i.e. $|\{v \in \text{dom}(s^*) \mid s(v) \neq s^*(v)\}|$ – the resulting count indicates how far s is from satisfying the goal condition.

Many heuristics for classical planning are derived from a model of the task, such as the SAS⁺ model introduced in the previous section. An obvious alternative is to learn to map a state s to its heuristic value $h(s)$. We focus on learning with NNs, although other supervised learning methods could be used. To learn a heuristic function, an NN is trained on pairs

of states s and cost-to-goal estimates c . 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 planning task $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s^*, \text{cost} \rangle$, and let $\mathcal{V} = \{v_1, \dots, v_n\}$ and $D(v_i) = \{d_{i,1}, \dots, d_{i,z_i}\}$, $i \in [n]$ be some order of the variables and their domains. We represent a state s by a sequence of facts

$$\mathcal{F}(s) = (f_{11}, f_{12}, \dots, f_{1,z_1}, \dots, f_{n1}, f_{n2}, \dots, f_{n,z_n}),$$

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 $\mathcal{F}_i = \{f_{i1}, \dots, f_{i,z_i}\}$ 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 propositions \mathcal{P} we write $\text{mutex}(\mathcal{P})$ if $\sum_{p \in \mathcal{P}} [p] \leq 1$ 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. The target output for training may be the cost-to-goal estimates directly or some encoding of them.

An important aspect of sample generation is the degree of dependency on the domain model or the planning task. We would like to learn in a black-box setting where we interact with the planning task only by functions that allow accessing the initial state s_0 , the goal condition s^* , and the applicable operators in a partial state. In this setting, we do not have access to the logical description of operators, but only to functions that return the successors or predecessors of a partial state (Sturtevant and Helmert 2019).

For several reasons, the black-box setting is interesting for studying the problem of learning heuristic functions. For example, it still can be applied if the domain model is unavailable because a simulator or learned models generate successors and predecessors. Also, a black-box setting can more easily be adapted than traditional logic-based heuristics to domain models with richer descriptions. Finally, we can address a question of Frances et al. (2017), asking if a planner that has only access to the state structure and goals can approach the performance of planners that also have access to the logical description of the domain, in the specific case of only learning heuristics functions.

Another aspect of sample generation is the cost of generating them. This depends on the number of samples and the cost to generate each. In particular, we have the problem of deciding how many samples are required since, generally, only a very small part of the state space can be sampled. Also, we have to decide which label to assign to each sample. The perfect heuristic h^* , that maps each state s to the cost of an optimal s -plan or ∞ if no such plan exists, would provide useful labels. However, even with samples labeled with perfect heuristic h^* , we expect the search performance to also depend on the region of the state space represented by the samples and how well the NN generalizes. In general, labeling samples with h^* is impractical. 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. The generation of samples to produce heuristic functions is not restricted to learning heuristics with NN. For example, Haslum et al. (2007) generates samples by random walks from the initial state to evaluate which patterns should be used to produce a pattern database heuristic.

Additionally, network architecture and sample generation depend on the range of tasks the learner intends to generalize. This may be a state space with a fixed goal condition, a planning domain, or an entire planning formalism. In the first case, the learned function has to generalize over a set of planning tasks defined by any pair of initial state s_0 part of a fixed FS and fixed goal s^* . In the second case, the learned function has to generalize over all tasks part of a domain. Finally, a learning-based heuristic that generalizes over a planning formalism is domain-independent.

2.3 Related Work

There have been two main approaches to define the architecture of neural network for learning heuristic functions. In the first, the architecture depends on the domain model (e.g. uses information about preconditions and effects of operators), in the second the architecture is independent of the model.

The usual setting in the first approach (Gehring et al. 2022; Shen et al. 2020; Ståhlberg et al. 2022; Toyer et al. 2018; Toyer et al. 2020) is to train a NN with samples of small tasks of a domain and evaluate it on larger tasks of the same domain. The networks trained can be general networks such as neural logic machines (Dong et al. 2019) and graph neural networks (Gori et al. 2005; Rivlin et al. 2020; Scarselli et al. 2008), or networks proposed explicitly in the context of planning such as Hypergraph Neural Networks (Shen et al. 2020) and Action Schema Networks (Toyer et al. 2018). These networks require the logical description of the domain and the task to be instantiated and can typically be generalized across different state spaces of a domain, and sometimes across domains, as is the case for Hypergraph Neural Networks. These approaches also help in understanding learning heuristics. For example, the main goal of Ståhlberg et al. (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.

In the second approach (Ferber et al. 2020; Ferber et al. 2022; O’Toole et al. 2022; Yu et al. 2020) typically trains an FNN and evaluates 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 et al. (2020) systematically study hyperparameters on the architecture of the FNN and found that their influence is secondary. They 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.

Furthermore, Yu et al. (2020) and O’Toole et al. (2022) perform backward searches from the goal, the former with depth-first search and the latter with random walks. Both use the lowest depth in which the state was generated as cost-to-goal estimates. Ferber et al. (2022) uses a combination of backward and forward searches (Arfaee et al. 2011). First, it generates new initial states with backward random walks and then solves them with a GBFS guided by a learned heuristic. The plans found provide the samples, and each sample is a state in the plan with the cost-to-goal estimate as its distance to the goal. O’Toole et al. (2022) also proposed a method to generate samples that do not use expansions. This method includes randomly generated states in the sample set with a cost-to-goal estimate equal to the maximum value in the sample set plus one. O’Toole et al. (2022) showed that this method substantially increases coverage. The methods from the second set of approaches

are highly independent of the domain model and planning task description and require low computational resources to generate samples and train the FNN. Also, despite having competitive results compared to logic-based heuristics, they are still unable to surpass the goal-count heuristic.

In particular, this work follows the second set of approaches since we aim to use minimal information from the task description. In Section 6.1, we compare the proposed approach to O’Toole et al. (2022) and Ferber et al. (2022) as they share the same NN architecture and dataset, with variations in the sampling and training procedures.

3. Sample Generation

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

We restrict this study to generalizing over planning tasks with initial states part of the same forward state space (FS) and a fixed goal condition. We study black-box 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 approaches that have access to mutexes derived from a SAS⁺ model. We address first, in Section 3.1, the generation of states, and then, in Section 3.2, the estimation of the cost-to-goal. In both sections, we discuss approaches from the literature and introduce new methods. The methods are a novel sampling strategy combining regression by breadth-first search with random walk, an adaptive regression limit, and two improvement methods for cost-to-goal estimates.

3.1 Generation of States

Unlike other domains in machine learning, where datasets of samples are often collected in real-world experiments and need to be manually annotated and curated, sample generation here is an algorithmic problem since we can generate the state space and compute cost-to-goal estimates. Approaches from the literature to generate the states include methods based on progression from one or more initial states, random sampling of the state space, or regression from the goal. In both progression and regression, one can apply different expansion strategies such as random walks, breadth- or depth-first searches, or combinations of more than one strategy such as bootstrapping (Arfaee et al. 2011). A problem in progression and random sampling is obtaining the cost-to-goal estimates. Without access to efficiently computable heuristic functions, or in black-box approaches, these values have to be obtained by search, which can have a cost exponential in the size of the task. To remain less dependent on models than logic-based methods and also more general, we focus on regression for which an upper bound on the cost-to-goal is readily available, as discussed in Sections 3.1.1 and 3.1.2. Since regression leads to partial states, the problem of generating complete states is addressed in Section 3.1.3. Random sampling is also discussed in Section 3.1.4.

3.1.1 SAMPLING BY REGRESSION

To generate samples, we expand partial states through regression from the backward state space. Expansion methods include breadth-first search (BFS), depth-first search (DFS), random walks (RW), as well as a combination of BFS with random walks. We discuss the completion of these partial states in Section 3.1.3. A regression rollout is defined as a series of partial state expansions, and it stops if the last expanded partial state has no predecessors or is at the depth limit L . The sampling generation process stops if the number of required samples N has been reached. Note that random walks can have multiple rollouts due to the depth limit L , while BFS and DFS only have one.

During expansion, we optionally use mutexes obtained from an analysis of the planning task – in this case as computed by Fast Downward (Helmert 2006) – to discard partial states which cannot be completed to complete states without violating a mutex, as described in Section 3.1.3. We also discard repeated partial states for random walk rollouts, such that a single rollout never cycles, although the same partial state may be sampled several times in different rollouts. Starting from $h(s^*) = 0$, a partial state $s' \in \text{pred}(s, o)$ obtained by applying an operator o backwards to partial state s has a cost-to-goal estimate $h(s') = h(s) + \text{cost}(o)$. For samples s that satisfy the goal condition ($\mathcal{S}(s) \subseteq \mathcal{S}(s^*)$) we reset the cost-to-goal estimate to $h(s) = 0$. Partial states are added to the sample set when generated in the random walks and when expanded in BFS and DFS. In all methods, operators backward applicable to a partial state are applied in random order.

Different expansion strategies generate sample sets with varying frequencies of optimal distances from the goal. In our experience, good coverage of states close to the goal, such as those obtained by BFS or random walks, is useful, as is the greater depth obtained by DFS or random walks. However, random walks from the goal often sample states close to the goal multiple times, and DFS can lead to a concentration of distant samples from the goal. Based on these observations, we propose a novel combination of BFS and random walks called FSM that aims to have a good coverage close to the goal and a diverse set of samples from the remaining state space. FSM has two phases. In the first phase, a fixed percentage p_{FSM} of the N samples is generated by BFS. (The value of p_{FSM} 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.) The BFS expands a partial state from layer k that generates n partial states from layer $k + 1$, and these partial states are sampled only if the current total samples plus n are within $p_{FSM}N$ samples; otherwise, no partial states are sampled and BFS expands another partial state. Let Q be the partial states of the set of samples that have not been expanded. The second phase generates multiple random walk rollouts, each starting from a partial state in Q chosen randomly with a complete replacement only after all partial states have been selected once. This is repeated until reaching N samples in the sample set. During a random walk, partial states sampled in the BFS phase are not resampled.

3.1.2 MAXIMUM REGRESSION LIMIT

A simple strategy to limit the sample generation depth is to use a fixed maximum limit L . Yu et al. (2020) and O’Toole et al. (2022) used this strategy with respective limits of $L = 200$ and $L = 500$. However, since tasks have state spaces with different maximum distances to the

goal, a fixed limit is undesirable for generating a representative part of the state space. If the maximum regression limit L overestimates the maximum distance-to-goal, the corresponding cost-to-goal estimates could be considerably larger than their respective h^* -values because states could be generated through extremely non-optimal paths. If the maximum regression limit L underestimates the maximum distance-to-goal, some part of the state space may never be generated.

The ideal regression limit for a given algorithm should allow it to generate a representative part of the state space without reaching states by extremely non-optimal paths. Let d^* denote the distance from the goal to its farthest state part of the state space. For BFS, d^* is an upper bound on the regression limit because at depth d^* the whole state space has been explored; for DFS and random walks, higher limits are required since they do not follow optimal paths. Since d^* , in general, is unknown and hard to compute, we propose two practical methods to define state-space-dependent maximum regression limits that aim to estimate d^* .

The first method uses the number of facts $F = |\mathcal{F}(s_0)|$ to estimate d^* . Since a set of facts represents each state, if one assumes that all operators have no precondition and change exactly one fact, one can reach any state in the state space from any other state by applying at most F operators. However, operators can modify more than one fact at a time. We refine this approximation by considering the average number of facts changed by the task operators. The second method is defined as $\bar{F} = \lceil F/\bar{\text{eff}} \rceil$ where $\bar{\text{eff}} = \sum_{o \in \mathcal{O}} |\text{eff}(o)|/|\mathcal{O}|$, i.e., the number of facts per mean number of effects in the operators.

3.1.3 SAMPLE COMPLETION

Regression sampling generates a set of partial states, while the NN is trained on and receives as input complete states during the search. Therefore, we evaluate three approaches to complete the undefined variables $\mathcal{U} = \mathcal{V} \setminus \text{dom}(s)$ of a partial state s . The first approach is a random assignment. The method assigns to each undefined variable $v \in \mathcal{U}$ of a partial state a random value in $\text{dom}(v)$. This completion strategy can be applied in a black-box setting since it uses information about the structure of a state.

The second method is mutex-based and aims to avoid states that are impossible to reach during the search. For each partial state, the method processes the variables \mathcal{U} in a random order and assigns to each of the variables in \mathcal{U} a random value in $\text{dom}(v)$ that does not violate the mutexes. We repeat this method for each partial state at most 10K times. If after 10K times this method does not produce a complete state that respects the mutexes, we still include the partial state in the sample set. In this case, we maintain all variables $v \in \mathcal{U}$ undefined, which sets all facts \mathcal{F}_v for variables $v \in \mathcal{V}^u$ to false¹.

Since the set of mutexes we use is incomplete, the mutex-based method can still generate complete states that are impossible to reach during the forward search. Thus, we evaluate an ideal completion method to investigate the influence of generating only complete states that are reachable during the search. For each partial state s , this method samples a random complete state from in the forward state space. Only for this method, if during the sample generation regression, a partial state s is generated such that no full state is in the forward

1. Empirically this case is negligible since it occurs in about 0.1% of the samples, in four of nine domains.

state space, then s is considered an invalid predecessor. This ideal method can only be applied to small tasks where we can enumerate the complete FS of the initial state s_0 .

In all three methods, we do not check for repeated states after completion, i.e., two partial states can generate the same complete state. As a result, it is possible for two complete states to have different h -values, and we address this in Section 3.2.1.

3.1.4 RANDOMLY GENERATED SAMPLES

O’Toole et al. (2022) have shown that adding randomly generated samples to a set of samples generated by expansion improves the performance of the search algorithm guided by the learned heuristic. They propose to set the cost-to-goal estimate for randomly generated samples to $L + 1$ for a maximum regression limit of L . To study the effect of randomly generated samples, we include this method in this study. These samples are generated from fully undefined states using the mutex-based completion technique. If the generated state s is already part of the sample set, i.e., $s = s_i$ for some $i \in [N]$ it receives cost-to-goal estimate h_i , otherwise the cost estimate $1 + \max_{i \in [N]} h_i$ that is larger than all samples estimates.

3.2 Improving Cost-to-Goal Estimates

We start by observing that cost-to-goal estimates never underestimate the true cost-to-goal h^* , as follows.

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

Proof. This follows because each estimate is witnessed by a plan. As observed in Section 2, a valid regression sequence $\rho = (o_1, \dots, o_k)$ generates a sequence of partial states $s_i = \text{pred}(s_{i-1}, o_i)$, $i \in [k]$ starting from the goal s_0 , such that s_k can reach the goal in at most k steps and with cost at most $\sum_{i \in [k]} \text{cost}(o_i)$. Furthermore, if $r = \text{pred}(t, o)$ and $r' \in \mathcal{S}(r)$ is a complete state, we have $\text{succ}(r', o) \in \mathcal{S}(t)$. Therefore, for any complete state s sampled from s_k , o_k, o_{k-1}, \dots, o_1 is a valid plan. Thus, $h(s)$ cannot be lower than the optimal cost $h^*(s)$. \square

In general, we expect better h -value estimates to lead to better-learned heuristics, and in turn to less expanded states during a search, but as previously noted this is not necessarily the case (Holte 2010). Therefore, we apply two procedures that improve the cost-to-goal estimates but maintain Property 3.1. The first, dubbed SAI, minimizes estimates over repeated samples, and the second, SUI, over successors of samples.

3.2.1 IMPROVEMENT OF REPEATED SAMPLES

In most state spaces, it is common for a state to be sampled in more than one random walk rollout, ending up with multiple duplicates with different estimates. Thus, 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 *exact* partial states as well as complete states. We call this procedure *sample improvement* (SAI). Choosing the minimum h -value is clearly 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.

3.2.2 IMPROVEMENT OVER SUCCESSORS

Besides sampling the same states, it is common to sample states that are neighbors in the state space, particularly for states close to the goal. Information from neighboring states can be used to improve the cost-to-goal estimates, as follows. First, for fast subset testing, we create an empty trie T , and for each partial state s_i , $i \in [N]$, we insert s_i into T keyed by its facts $(s(v))_{v \in \mathcal{V}}$. Next, we build a graph $G = (V, A)$ with all sampled partial states $V = \{s_i \mid i \in [N]\}$, and $A = \emptyset$. For every pair of partial states $s, t \in V$ such that for some operator $o \in \mathcal{O}$ applicable to s we have $\mathcal{S}(\text{succ}(s, o)) \subseteq \mathcal{S}(t)$, we add an arc (s, t) of weight $w(s, t) = \text{cost}(o)$ to A . (Unlike in regression, if $\text{pre}(o)$ mentions an undefined variable in s then o is not applicable.) We use the trie T to search for each successor in the partial states that are supersets. For partial states generated by regression, by construction, at least one such successor exists, except for the goal s^* . Using 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 $h(s) = \min(h(s), h(t) + w(s, t))$. The process continues as long as there are updates. We call this procedure *successor improvement* (SUI). As for SAI, we only add valid transitions among partial states, so all distances are still witnessed by plans, and Property 3.1 is maintained.

3.3 Training Set Generation Workflow

The process described in this paper follows the workflow illustrated in Figure 1. It begins with the generation of samples using a sampling algorithm such as FSM. The next steps improve the sample estimates using SAI, followed by SUI, both of which are applied on partial states. The second and third steps are optional and can be applied independently. However, if applied, they are applied in this order. The fourth step uses a state completion technique to produce complete states from partial states. Then, optionally, we add randomly generated samples to the sample set. Finally, if SAI was used previously, then it will be applied to the complete states.

4. Experimental Setup

In this section, we present the experimental settings used throughout the experiments in small state spaces (Section 5), where we can enumerate the complete forward state space with associated perfect cost-to-goal estimates h^* , and in large state spaces (Section 6), in order to validate our findings in a practical setting with large planning tasks.

Common 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 propositional fact is set to 1 if it is true in the state and 0 otherwise, as explained in Section 2.2, 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 output of the NN uses ReLU activation during training and evaluation. The training

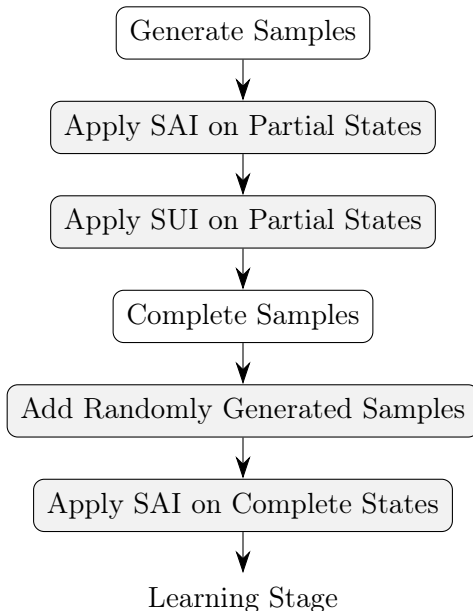


Figure 1: Training set generation workflow, shaded steps are optional.

uses the Adam optimizer (Kingma and Ba 2015), a learning rate of 10^{-4} , an early-stop patience of 100, and a mean squared error 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. The number of samples N depends on the type of experiment and domain, and is given in the sections below. In the experiments, different learned heuristics are denoted as \hat{h}_B^A , where A and B indicate different algorithmic choices.

We select the domains and tasks from Ferber et al. (2022), namely: Blocks World, 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 Downward² planning system with PyTorch 1.9.0 (Ferber et al. 2020; Paszke et al. 2019). The source code, planning tasks, and experiments are available³. All experiments were run on a PC with an AMD Ryzen 9 3900X 12-core processor, running at 4.2 GHz with 32 GB of main memory, using a single core per process distributed among 12 (for small planning tasks) and 10 cores (for large planning tasks). We solve all tasks with GBFS guided by the learned heuristic \hat{h} . When multiple states have the same heuristic value, the state that was generated earlier (i.e., has a lower generation order) is selected for expansion first (FIFO).

Initialization. An NN may fail to train if, after initialization, it outputs zero for all training samples (this is called “born dead” in Lu et al. 2020). This condition arises when the weights and biases of the NN are initialized in a way that consistently maps the ReLU activation region to negative values, resulting in a zero gradient and no weight updates during training.

2. Available at <https://github.com/PatrickFerber/NeuralFastDownward>.

3. Available at <https://github.com/yaaig-ufrgs/NeuralFastDownward-FSM>.

Among *all* small state space experiments using neural networks, about 5 % of the networks were born dead, where the Blocks World domain represented 84 % of the total of born deads, followed by VisitAll (15 %) and Transport (about 1 %). In the large state space experiments, less than 1 % of the networks were born dead. Pipesworld-NoTankage represented 98 % of born deads, and Storage 2 %. In the case of a born dead NN, we reinitialize it with a different seed until the NN outputs a non-zero value for some sample. In our experiments, no NN requires more than one reinitialization.

Baseline. We compare distinct configurations to a baseline \hat{h}_0 similar to previous approaches from the literature. For the baseline, the NN is trained by a sampling method that uses random walks with a regression limit of 200 backward steps. In addition, mutexes are used during regression and for sample completion, but resetting the h -value to 0 in goal states and the improvement strategies SAI and SUI are turned off.

BFS budget in FSM. To determine a value for p_{FSM} , preliminary experiments in small state spaces were performed using $p_{\text{FSM}} \in \{0.01, 0.05, 0.1, 0.2, \dots, 0.9\}$. We use the same baseline configuration but with the FSM sampling algorithm. The corresponding geometric mean expansions obtained were 64.35, 57.89, 56.31, 60.43, 57.6, 57.3, 62.39, 69.22, 77.49, 78.09, and 76.04. respectively. Thus, we fix $p_{\text{FSM}} = 10\%$ in the experiments.

5. Experiments in Small State Spaces

In this section, we study the behavior of different sampling methods on small state spaces. In particular, we analyze the effects of the following factors on the number of expanded states during search: the algorithms used to generate samples (partial states and their respective cost-to-goal estimates) (Section 5.1), the partial state completion methods (Section 5.2), and the accuracy of cost-to-goal estimates and methods to improve them (Section 5.3). Finally, we compare the proposed learning-based approaches to commonly used heuristics during search (Section 5.4).

For each domain, we select the task with the largest size state space that can be enumerated completely to obtain h^* -values. We only select tasks with state spaces with 30 K states or more, and fewer than 1 M states. Table 1 shows the tasks and their state space sizes. For domains Grid, Rovers, Scanalyzer, and Transport, the best task found had fewer than 30 K states, and VisitAll more than 1 M, so we manually modified these tasks. We could not find a non-trivial task within the limits for Depot, Pipesworld-NoTankage and Storage, so they were excluded from the 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. 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 conducted tests using a total of 25 combinations of sample seeds and network seeds (5 sample seeds \times 5 network seeds). Specifically, we trained five networks for each sample seed, ranging from sample seed 1 to sample seed 5, and repeated the process for each network seed. Additionally, we evaluated each network over 50 different initial

Table 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			

states for each domain. In other words, when we compare the number of expanded states, each cell in the tables represents the geometric mean of 25 networks over 50 test tasks, i.e., 1250 searches with GBFS. The training time has been limited to 30 minutes. If not stated otherwise, methods BFS, DFS, RW and FSM use mutexes, the improvement strategies SAI and SUI, and the number of samples is equal to 1% of the state space size. Under these conditions, more than 98.5 % of the NNs converge.

Furthermore, if not stated otherwise, for the fixed parameters we use the baseline setting defined in Section 4 (regression limit $L = 200$, mutexes, no cost-to-goal improvement methods).

5.1 Sample Generation

Our first set of experiments aims to analyze the influence of the sample generation methods on the quality of the learned heuristics. Section 5.1.1 compares the different sampling algorithms, Section 5.1.2 compares the adaptive regression limits, Section 5.1.3 investigates the effect of randomly generated samples.

In all the experiments below when comparing several treatments, we apply the non-parametric Mack-Skillings test for blocked designs with an equal number of replications in each experimental cell, with a confidence level $\alpha = 0.01$ to see if the treatments are significantly different⁴, such . If not specified otherwise, the test is applied to each domain separately, with initial states as blocking factors. If treatments are significantly different we apply a corresponding post-hoc test to find the different treatments, again with a family-wise error rate of $\alpha = 0.01$. For details about the tests see Hollander et al. (2014, ch. 7.9 and 7.10). In all following tables, the statistically best values are highlighted in bold.

5.1.1 GENERATION ALGORITHMS

This experiment compares the four sample generation algorithms BFS, DFS, RW, and FSM and shows that a sample set including diverse regions of the forward state space (such as those generated by RW and FSM) yields fewer expansions in the geometric mean.

To control the effect of the cost-to-goal estimates on the quality of the learned heuristic, we replace sample estimates with optimal values h^* before training. The left side of Table 2 shows the geometric mean of the number of expanded states of a GBFS guided by the learned heuristics. The right side in column FS shows the mean h^* -value of the whole forward state

4. The Mack-Skillings test is here preferable over, e.g., the more common Friedman test, since we have several replications per experimental cell.

Table 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 entire forward state space (FS) and the generated sample sets.

Domain	Expanded states				Mean h^* -values				
	\hat{h}_{bfs}	\hat{h}_{dfs}	\hat{h}_{rw}	\hat{h}_{fsm}	FS	BFS	DFS	RW	FSM
Blocks	1881.33	63.24	34.46	30.76	18.77	10.77	17.68	11.93	14.42
Grid	83.03	91.03	66.67	65.14	16.59	5.32	17.11	7.22	8.86
N-Puzzle	929.73	132.37	85.02	83.95	21.97	10.42	20.17	20.02	19.78
Rovers	11.63	12.20	11.08	11.09	6.45	2.34	5.19	4.95	4.99
Scanalyzer	100.81	21.44	20.34	19.89	8.34	2.91	7.91	7.07	6.51
Transport	94.69	39.96	18.81	18.42	12.23	2.94	11.30	10.05	9.53
VisitAll	45.25	20.99	18.46	18.04	8.97	2.00	9.10	6.80	6.57
Geo. mean	132.84	40.12	28.51	27.64	12.21	4.22	11.45	8.81	9.09

space, and columns BFS, DFS, RW, and FSM show the mean h^* -values of the sample sets representing 1% of the forward state space FS. We see that heuristic \hat{h}_{bfs} leads to more expanded states than \hat{h}_{dfs} , which in turn expands about 40% more states than \hat{h}_{rw} and \hat{h}_{fsm} , which perform similarly. All algorithms are significantly different for $\alpha = 0.01$. Using \hat{h}_{bfs} is significantly worse, and leads in all domains the highest or close to the highest number of expansions. Heuristic \hat{h}_{dfs} has a high number of expansions in Blocks World, 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 geometric mean estimates in all domains. Although the mean h^* -value of DFS is closest to that of the whole forward state space FS, 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 a single rollout with BFS or DFS.

We now compare these results to results shown in Table 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 2. This 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}_{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. Again, all algorithms have a different performance for $\alpha = 0.01$. The increase in the number of expanded states is highest for \hat{h}_{dfs} , which expands a mean of 177.02 states using h -values instead of 40.12 when using h^* , i.e. about four 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.

Comparing the mean h^* -values from Table 2 to the mean h -values Table 3, we can see that the methods generate samples that overestimate h^* -values and that DFS overestimates much more than the others. Also, when contrasting the two extreme sample sets generated

Table 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 entire forward state space (FS) and the mean estimated h -values over the generated sample sets.

Domain	Expanded states				Mean h -values				
	\hat{h}_{bfs}	\hat{h}_{dfs}	\hat{h}_{rw}	\hat{h}_{fsm}	FS	BFS	DFS	RW	FSM
Blocks	1881.33	145.51	67.41	60.56	18.77	10.77	166.11	28.09	38.43
Grid	271.15	1115.51	142.69	148.27	16.59	6.77	184.59	21.06	22.46
N-Puzzle	929.73	707.33	173.90	146.11	21.97	10.42	187.51	96.29	90.65
Rovers	49.45	22.27	18.63	16.62	6.45	4.65	27.11	25.90	24.91
Scanalyzer	166.06	93.20	52.00	38.86	8.34	2.98	146.17	90.30	87.86
Transport	166.52	567.07	89.84	103.13	12.23	3.27	196.12	95.20	88.59
VisitAll	115.65	40.31	20.95	20.54	8.97	2.74	28.48	22.48	22.37
Geo. mean	257.47	177.02	60.74	56.31	12.21	5.14	103.50	43.30	44.38

by BFS and DFS with RW and FSM, we find that in both Tables, BFS generates samples closer to the goal and DFS more distant from it.

In an additional experiment we verified that not only the absolute predicted h -values, but their ranking agrees well with h^* -values. The mean correlation between these values, over all domains, are 0.302, 0.237, 0.464, 0.521, for methods BFS, DFS, RW, and FSM, respectively. Except for BFS, this agrees with the ranking of the methods according to the number of expanded states, and shows that a better sample distribution leads to predicted h -values that guide better.

Although BFS has an estimation quality close to the h^* -values, and a better correlation, it expands more states than the other methods. These results suggest that sampling more states in localized regions of the state space is not sufficient to achieve good results during search with GBFS. Furthermore, \hat{h}_{fsm} expands fewer states than \hat{h}_{rw} and is the best in five of seven domains. Because \hat{h}_{fsm} had a lower increase in expansions compared to \hat{h}_{rw} , we focus on FSM in the remaining experiments.

5.1.2 REGRESSION LIMIT

In this experiment, we analyze the influence of the regression limit on the number of expanded states with sample generation strategy FSM. The experiments show that values of regression limits slightly larger than d^* , which is the maximum distance to the goal in the state space, tend to yield fewer expansions in the geometric mean.

We compare the fixed regression limits $L = 40$ (a good fixed limited for the small state spaces chosen by observing d^*) and $L = 200$ (used by Yu et al. (2020)) with two state-space-dependent strategies that aim to estimate d^* : setting the regression limit to the number of facts F or to the number of facts divided by the mean number of effects \bar{F} . We also present results for $d^*/2$, d^* and $2d^*$. The mean values for F , \bar{F} , and d^* , are shown on the left-hand side in Table 4. Both F and \bar{F} overestimate the mean largest distance d^* (except for \bar{F} in

Table 4: State space information and expanded states of GBFS guided by \hat{h} trained on FSM samples with different regression limits L and no h -value improvements.

Domain	Mean Limits			Expanded states						
	d^*	F	\bar{F}	$d^*/2$	d^*	$2d^*$	40	200	F	\bar{F}
Blocks	24	64	17	1279.70	43.48	45.29	49.37	60.56	46.81	89.89
Grid	32	76	44	95.07	122.18	163.26	134.02	148.27	158.29	119.93
N-Puzzle	31	81	41	112.78	66.37	77.42	75.29	146.11	84.25	66.52
Rovers	19	32	27	32.05	17.93	15.73	15.75	16.62	15.69	14.76
Scanalyzer	15	42	20	92.50	63.21	49.50	38.51	38.86	40.92	52.11
Transport	17	66	35	48.91	47.95	51.16	49.63	103.13	65.24	49.39
VisitAll	15	31	17	46.30	29.17	23.69	21.35	20.54	24.03	27.80
Mean	22	56	29	98.84	47.66	47.43	44.02	56.31	48.47	49.78

Blocks), which is desirable since random walks do not follow the optimal paths. Finally, \bar{F} estimates d^* better than F for all domains.

The right-hand side of Table 4 gives the number of expanded states for GBFS guided by \hat{h} trained on FSM samples with different regression limits L and no h -value improvements. We see that limits d^* , $2d^*$, $L = 40$, F , and \bar{F} perform better than $d^*/2$ and the fixed limit 200. The results with $d^*/2$ show that underestimating d^* generally degrades performance. Also, note that \bar{F} underestimates d^* in Blocks, and the performance substantially degrades. The proposed strategies F and \bar{F} yield similar results to d^* , $2d^*$, and $L = 40$. However, we do not have access to d^* in general. Therefore, the proposed strategies provide good limits in practice and perform better than the previously used fixed limit of $L = 200$.

5.1.3 RANDOM SAMPLES

In this experiment, we evaluate the effect of adding randomly generated samples (as explained in Section 3.1.4) to a sample set generated using the sampling strategy FSM and regression limit \bar{F} . Unlike the previous experiments, we additionally apply the improvement strategies SAI and SUI to the states sampled by regression (SAI and SUI are discussed in Section 5.3.1). The following experiments show that randomly generated samples are beneficial up to about 60% of the total sample set, provided their cost-to-goal estimates are larger than the existing samples. We generate samples $S = \{(s_1, h_1), \dots, (s_N, h_N)\}$ where 10%, 20%, ..., 100% are random samples (to which SAI and SUI are not applied) and the rest is sampled with FSM and a regression limit \bar{F} . Random samples get an h -value 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 states). 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 5 shows the performance up to 70% random samples. We have omitted 80%, 90%, and 100% since expansions are higher (respectively 45.12, 56.85, and 12081.97). The

Table 5: Expanded states of GBFS with a learned heuristic over samples generated by FSM with regression limit \bar{F} , all cost-to-goal improvement strategies, and a varying percentage of randomly generated samples.

Domain	Percentage of random samples							
	0	10	20	30	40	50	60	70
Blocks	79.61	42.80	43.36	46.99	46.20	39.74	47.60	55.88
Grid	70.14	50.98	49.37	59.90	53.27	56.63	59.28	86.81
N-Puzzle	70.33	67.51	68.05	71.88	73.01	70.27	76.47	82.78
Rovers	14.98	12.97	12.37	12.45	12.40	12.99	12.89	14.41
Scanalyzer	37.00	24.47	22.40	24.57	26.13	30.00	24.04	26.61
Transport	21.39	22.81	24.56	25.27	28.74	31.16	37.46	44.13
VisitAll	28.54	21.57	20.76	21.05	21.02	21.96	22.75	25.04
Geo. mean	38.82	30.22	29.74	31.81	32.11	32.95	34.40	40.21

number of expansions is considerably reduced when using random samples, with 20% random samples performing slightly better than other percentages.

To better understand the effect of random samples, we have performed four additional experiments with 20% of random samples. The first focuses on cost-to-goal estimates. We keep the samples but replace $H + 1$ by small values, namely either a random h -value from the sample set S , or a random value drawn from $U[1, 5]$. This leads to overall geometric means of 173.91 and 3231.64 expanded states, respectively. The second experiment forces the random samples to be part of the FS. This leads to a geometric mean of 31.11 expanded states. The third experiment does not apply mutexes and leads to a geometric mean of 30.27 expanded states. The fourth experiment generates random samples with true noise (each Boolean fact of a state has a 50% chance of being 0 or 1), yielding a geometric mean of 45.01 expanded states. From these additional experiments, it is clear that the most relevant factor is a high h -value, although generating random state samples with true noise yields more expansions than not using random samples. The most probable explanation for the effect of random samples seems to be that they enhance the likelihood of guiding the search toward samples where the network has learned good cost-to-go estimates, but that would require further experiments.

5.2 Partial State Completion

Now we focus on how sampled partial states are converted to complete states. In this experiment, we use FSM, regression limit \bar{F} , and all the samples have optimal cost-to-goal estimates h^* . We compare three different sample completion strategies for a partial state s . All of them select a random state from the set of states $\mathcal{S}(s)$ 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 (perfect baseline). Note that even without restrictions, state completion is subject to the mutexes implicit in the SAS⁺ encoding.

Table 6: Expanded states of GBFS with \hat{h} trained with FSM, \bar{F} , h^* cost-to-goal estimates, and different state completion approaches, and the probability $P(s \in FS)$ of a completed state being in the forward state space.

Domain	Expanded states			$P(s \in FS)$	
	None	Mutex	FS	None	Mutex
Blocks	163.14	88.19	75.56	0.00	99.85
Grid	64.55	61.35	55.33	57.61	76.53
N-Puzzle	154.74	63.99	71.38	8.38	100.00
Rovers	10.77	10.98	10.10	24.26	23.27
Scanalyzer	30.34	31.10	33.64	100.00	100.00
Transport	17.23	17.20	16.12	7.30	7.50
VisitAll	20.93	21.18	21.89	64.41	63.83
Geo. mean	40.92	33.05	32.18		

Table 6 presents the expanded states and the probability of a completed state being in the FS for these approaches. We can see that applying mutexes leads to a moderate reduction of the number of expanded states, and is very close to an ideal completion of the states. However, completing randomly also presents competitive results, except for N-Puzzle and Blocks World. In precisely these two domains completing using mutexes is much better than randomly, since generating a valid combination of facts “clear” in Blocks World, or assigning the empty tile to the correct position is improbable. For Rovers, Scanalyzer, Transport, and VisitAll, there is no distinction between the “None” and “Mutex” approaches because either there are no mutexes or the mutexes rarely apply to the sample states.

5.3 Estimates of Cost-to-Goal

This set of experiments aims to analyze how techniques used to improve cost-to-goal estimates influence sample quality. To this end, in Section 5.3.1, we directly compare the difference of the improved estimates (with SAI and SUI) to h^* -values. Then, to see how well the learned heuristics generalize, in Section 5.3.2 we evaluate them over the complete forward state spaces from Table 1.

5.3.1 QUALITY OF ESTIMATES

First, we compare the quality of the cost-to-goal estimates to h^* with and without h -value improvement techniques and distinct regression limits. We generate the samples with FSM and varying regression depth limits, and complete the partial states with mutexes. We demonstrate that using adaptive regression limits and cost-to-goal improvements (SAI and SUI) consistently brings the estimates closer to h^* . The results in Table 7 show the mean absolute difference between the sample estimates and h^* , so smaller means indicate better approximations. Note that we are not evaluating the output of an NN, but the cost-to-goal estimates of the sample sets.

Table 7: Mean difference of estimates of samples of the sample set to h^* .

Domain	No improvements			With SAI and SUI		
	200	F	\bar{F}	200	F	\bar{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

The improvement strategies SAI and SUI substantially reduce the estimates for all regression limiting methods. For regression limits L of 200, F and \bar{F} , using only SAI reduces the estimates to 31.28, 13.95, 4.92 respectively, and using only SUI reduces the estimates to 11.10, 5.48, 1.94 respectively. Thus, SUI has the most effect on improving the cost-to-goal estimates compared to SAI. Also, the adaptive regression limiting methods are better than the fixed default 200, and \bar{F} has the best results. When comparing 200 to \bar{F} without h -value improvements, the cost-to-goal estimate difference to h^* decreases by about 6 times in the geometric mean, with Blocks World having the best performance, improving more than 25 times. Finally, using both h -value improvements and a regression limit \bar{F} reduces the difference to h^* from 33.45 to only 1.60.

5.3.2 EVALUATION OVER THE FORWARD STATE SPACES

We now analyze the quality of the proposed learned heuristics and the logic-based heuristics fast-forward h^{FF} (Hoffmann and Nebel 2001) and goal-count h^{GC} over all states from the forward state space FS of each task. In summary, h^{FF} better approximates the h^* estimates over the whole forward state space, but the proposed approaches are close. Table 8 shows the results. Except for the baseline \hat{h}_0 , the samples are generated with FSM limited by an L of 200, F or \bar{F} , and improved with SAI and SUI. The learned heuristic $\hat{h}_{\bar{F}}^{20\%}$ is the same as $\hat{h}_{\bar{F}}$, but 20% of the samples are randomly generated. We see that $\hat{h}_{\bar{F}}$ reduces the difference of the predicted result to the real one by about 11 times when compared to \hat{h}_0 , and when compared to h^{GC} it has the smallest difference in all but two domains. Also, $\hat{h}_{\bar{F}}$ presents a similar mean difference to h^{FF} . Note that due to the randomly generated samples in the sample set, $\hat{h}_{\bar{F}}^{20\%}$ doubles the difference compared to $\hat{h}_{\bar{F}}$. Blocks World is the only domain with a lower value, due to around two-thirds of the FS states having an h^* -value within a range of two or less the value assigned for random samples, thus improving the average.

Comparing Tables 7 and 8, we observe that relative order between 200, F and \bar{F} is preserved. The mean difference of the samples' estimates to h^* for \bar{F} is 1.60 in Table 7, and when the corresponding $\hat{h}_{\bar{F}}$ is required to generalize over the entire FS, the mean difference is 3.57.

Table 8: Mean difference of h^{FF} , h^{GC} and \hat{h} , to h^* when evaluated over the FS.

Domain	h^{FF}	h^{GC}	\hat{h}_0	\hat{h}_{200}	\hat{h}_F	$\hat{h}_{\bar{F}}$	$\hat{h}_{\bar{F}}^{20\%}$
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

5.4 Comparison to Logic-Based Heuristic Functions

We now compare the NN-based heuristics with logic-based heuristics and show that the learned heuristics are competitive with commonly used heuristics. The number of expanded states of GBFS guided by different heuristic functions is shown in Table 9. The NNs are trained with samples obtained with FSM, all cost-to-goal improvement strategies and regression depth limited by \bar{F} .

First, we see that \hat{h}_0 expands fewer states than h^{GC} in most domains except Scanalyzer and VisitAll, but it is far worse than h^{FF} except in Blocks World and VisitAll, where the learned heuristic has particularly good results. We also see that $\hat{h}_{\bar{F}}$ expands fewer states than \hat{h}_0 in five domains. In turn, h^{FF} has better results than $\hat{h}_{\bar{F}}$ in five domains. However, $\hat{h}_{\bar{F}}$ surpasses h^{FF} if 20% of the samples are random, or if we increase the budget of $\hat{h}_{\bar{F}}$ to 5% (instead of 1%) of the number of states in the forward state space as shown in Table 10. This table also indicates that after increasing the budget to 50% of the number of states in the forward state space, the gains in quality of the learned heuristic are negligible.

Comparing Tables 8 and 9, we see that for the NN-based heuristics, the order of approaches in terms of $h-h^*$ difference remains consistent for \hat{h}_0 and $\hat{h}_{\bar{F}}$, but not for $\hat{h}_{\bar{F}}^{20\%}$, which has a higher mean difference than $\hat{h}_{\bar{F}}$ but presents the least state expansions, even when compared to h^{FF} . 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 FS, random samples are obtained after the regression procedure and can be helpful.

6. Experiments in Large State Spaces

The main goal of the following experiments is to verify the findings from the previous sections on large state spaces, so we compare different configurations of the improved methods with logic-based heuristics and a baseline. In summary, the following experiments confirm that the previous findings also apply to large state spaces. Then, in Section 6.1, we compare the proposed approach with other learning-based approaches from the literature. We report

Table 9: Expanded states of GBFS with different heuristic functions.

Domain	h^*	h^{FF}	h^{GC}	\hat{h}_0	$\hat{h}_{\overline{F}}$	$\hat{h}_{\overline{F}}^{20\%}$
Blocks	19.25	136.35	248.06	67.41	79.61	43.36
Grid	20.52	32.24	197.12	169.98	70.14	49.37
N-Puzzle	22.40	121.17	607.62	173.82	70.33	68.05
Rovers	9.80	10.54	44.61	18.63	14.98	12.37
Scanalyzer	8.97	23.14	28.05	52.00	37.00	22.40
Transport	13.17	17.12	184.37	89.57	21.39	24.56
VisitAll	11.84	25.86	16.09	20.95	28.54	20.76
Geo. mean	14.28	34.44	101.42	62.25	38.82	29.74

Table 10: Expanded states of GBFS with $\hat{h}_{\overline{F}}$ trained with a number of samples corresponding to some percentage of the number of states in the FS of each task.

Domain	5 %	25 %	50 %	100 %
Blocks	25.40	20.35	20.03	19.68
Grid	64.91	50.65	39.42	30.09
N-Puzzle	34.72	26.44	25.07	23.73
Rovers	14.78	13.36	13.26	17.43
Scanalyzer	15.78	11.80	11.86	12.26
Transport	19.19	17.49	16.20	15.32
VisitAll	20.92	18.16	19.26	20.45
Geo. mean	24.54	20.18	19.23	19.14

results over 9 combinations of sample seeds and network seeds (3 sample seeds and 3 network seeds).

We use the benchmark defined by Ferber et al. (2020) and Ferber et al. (2022). Ferber et al. (2020) selects for each domain, IPC planning task with their original initial states, that are solved within 1 and 900 seconds by GBFS with h^{FF} . Each domain has the following number of selected tasks: Blocks World, 5; Depot, 6; Grid, 2; N-Puzzle, 8; Pipesworld-NoTankage, 10; Rovers, 8; Scanalyzer, 6; Storage, 4; Transport, 8; VisitAll, 6. For each selected task, Ferber et al. (2020) generates new initial states with random walks from the original initial state. These new initial states with the original goal conditions of the selected tasks define the test planning tasks. For selected task Ferber et al. (2022) experiments with 50 moderate tasks resulting in a benchmark with 3,150 planning tasks. We use the same benchmark used by Ferber et al. (2022).

We generate samples within 1 hour and set 1 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. We fix the number of samples at $N = 16M/|\mathcal{V}|$, such that domains with more variables \mathcal{V} receive proportionally less samples. This results in a mean of 500 MB RAM during sampling and 2 GB during the h -value improvement.

Table 11: Mean coverages and expanded states of the learned heuristics with both regression limit 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. Geometric mean is used for the overall mean of expanded states.

Domain	Coverage (%)				Expanded states			
	\hat{h}_F	$\hat{h}_{\bar{F}}$	\hat{h}'_F	$\hat{h}'_{\bar{F}}$	\hat{h}_F	$\hat{h}_{\bar{F}}$	\hat{h}'_F	$\hat{h}'_{\bar{F}}$
Blocks	93.33	96.31	100.00	100.00	19452	2722	4241	5406
Depot	81.26	84.67	85.96	86.41	11869	13440	15528	14557
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	3157	7628	5414	5345
Rovers	13.92	13.33	13.67	13.53	15	17	16	15
Scanalyzer	66.44	67.78	66.15	66.67	146	35	188	38
Storage	3.94	9.11	1.67	8.33	-	-	-	-
Transport	75.53	85.97	81.42	87.50	64985	32603	36064	29671
VisitAll	92.81	80.52	92.70	79.00	1931	2978	1324	4575
Mean	54.32	63.21	50.81	53.32	2710	1849	2152	2020

First, we reassess the previous results using the regression limits F and \bar{F} on large state spaces since the previous experiments (Section 5.1) produced similar results. Table 11 shows the mean coverage and number of expanded states for the methods using F or \bar{F} , with all h -value improvements, and with and without mutexes (denoted by \hat{h}').

When comparing the learned heuristic $\hat{h}_{\bar{F}}$ over \hat{h}_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 F is better – this is also observed in the small state space experiments. Without mutexes, the coverage improvements from $\hat{h}'_{\bar{F}}$ over \hat{h}'_F are minor. The smaller number of expanded states in approaches with \bar{F} indicate samples of higher quality. With or without mutexes, using \bar{F} has the highest positive effect in N-Puzzle, increasing its coverage by about four times. Also, we see that not using mutexes improves the results in Blocks World, Depot and Transport, while having a minimal effect in Pipesworld-NoTankage, Rovers, Scanalyzer and VisitAll. From these results, we conclude that \bar{F} has better performance than F for large state spaces. Therefore, the following experiments will use \bar{F} .

Next, we compare the logic-based heuristics h^{FF} and h^{GC} , the baseline \hat{h}_0 and the best approach $\hat{h}_{\bar{F}}^{20\%}$. The results are presented in Table 12. We see that h^{FF} dominates in most domains, achieving twice the mean coverage of the baseline \hat{h}_0 . However, $\hat{h}_{\bar{F}}^{20\%}$ has only 12% less mean coverage than h^{FF} , improving \hat{h}_0 by about 31%, with competitive coverage in most domains. Note that $\hat{h}_{\bar{F}}^{20\%}$ achieves better mean coverage than h^{GC} , with higher or equal coverage in 6 out of 10 domains. Also, in all domains except Pipesworld-NoTankage and Transport, the best-learned heuristic expands fewer states on the same initial states when compared to h^{FF} , indicating that the learned heuristic is more informed and that the inferior

Table 12: Mean coverages and expanded states of the logic-based heuristics h^{FF} and h^{GC} compared to the baseline learned heuristic \hat{h}_0 and the best learned heuristic $\hat{h}_{\bar{F}}^{20\%}$, obtained via training over samples with FSM, \bar{F} , 20 % of random samples, and all h -value improvement strategies. Expanded states consider only the initial states solved by all heuristics; N-Puzzle and Storage had no common solved initial state. For expanded states we use the geometric mean.

Domain	Coverage (%)				Expanded states			
	h^{FF}	h^{GC}	\hat{h}_0	$\hat{h}_{\bar{F}}^{20\%}$	h^{FF}	h^{GC}	\hat{h}_0	$\hat{h}_{\bar{F}}^{20\%}$
Blocks	100.00	100.00	100.00	100.00	12499	32115	2771	2099
Depot	94.33	80.00	57.19	89.26	44811	197999	13139	6756
Grid	94.00	51.00	38.11	60.33	4579	23901	2313	4332
N-Puzzle	92.50	4.00	13.75	86.81	-	-	-	-
Pipes-NT	63.40	89.40	13.51	79.84	100	1348	7653	756
Rovers	85.50	66.00	13.53	15.39	12	38	36	11
Scanalyzer	100.00	100.00	59.70	73.67	50	81	1085	18
Storage	33.00	13.50	1.94	27.67	-	-	-	-
Transport	100.00	100.00	48.89	100.00	3505	244299	76457	7476
VisitAll	92.00	100.00	74.19	98.85	31560	259	3787	1656
Mean	85.47	70.39	42.08	73.18	1810	5140	3582	881

coverage is an effect of the slower expansion speed of the NN-based heuristics. Note, however, since only tasks solved by all methods are considered, the evaluation is biased towards easier tasks. 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 was generated from 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 11 and 12, we notice that all NN-based heuristics have similarly poor results in Rovers, independent of configuration. Considering only the learned heuristics, when using 20 % of random samples ($\hat{h}_{\bar{F}}^{20\%}$) instead of 0 % ($\hat{h}_{\bar{F}}$), 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.

6.1 Comparison to Other Sample-Based Methods

In this section we compare the learned heuristics to those of Ferber et al. (2022) and O’Toole et al. (2022). All methods share the same benchmark and NN architecture, use a Boolean representation for the samples, and mutexes to complete unassigned state variables. They differ, however, in batch size, patience value, NN initialization functions, and percentages of data split into training and validation sets. Furthermore, machine configurations, libraries, and resource limits are different, and existing results cannot be easily reproduced. Thus, we limit ourselves to a qualitative comparison. Even within these constraints, the following experiments indicate that the proposed approach achieves better coverage in less time.

Ferber et al. (2022) learns using bootstrapping (Arfae et al. 2011), which iteratively improves the learned heuristic by training with samples with cost-to-goal estimates of increasing distances from the goal. The Bootstrap method generates new initial states with backward random walks from the goal for a given state space. For each generated initial state, the method attempts to find a plan with GBFS guided by the current learned heuristic. The plans found with their respective states and cost-to-goal estimates are used as training samples. If the method finds a plan for more than 95% of the initial states, it generates initial states with longer random walks. 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) generates samples using random walks from the goal. They perform 5 rollouts with a regression limit of $L = 500$ and use the current depth as cost-to-goal estimates; they use the Tarski framework (Francés et al. 2018) to perform the regression procedure and develop a cost-to-goal improvement strategy equivalent to SAI, and 50 % of the samples are randomly generated as described in Section 3.1.4. 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.

We now compare the coverage results between all methods over the same tasks. In the comparison we consider only the best configurations of both Ferber et al. (2022) and O’Toole et al. (2022). We also use 100 K samples to make them more comparable to O’Toole et al. (2022). In these experiments, our sampling and training procedures can take a combined time of up to 2 hours, and we use a search time limit of 5 minutes.

Table 13 shows the coverage results of our best method using 0 %, 20 % and 50 % of random samples, together with the results as reported by Ferber et al. (2022) (\hat{h}^{Boot}) and O’Toole et al. (2022) ($\hat{h}^{\text{N-RSL}}$). Considering coverages, we notice that the heuristics $\hat{h}_{\bar{F}}^{\%}$ have results more similar to $\hat{h}^{\text{N-RSL}}$ than to \hat{h}^{Boot} , and higher coverage, except for Grid, Rovers, Scanalyzer, and Storage. Heuristic \hat{h}^{Boot} is best in Grid, Rovers, and Storage. Particularly in Storage, \hat{h}^{Boot} has 89 % coverage, while coverage of the other approaches is close to 20 %. Heuristics $\hat{h}_{\bar{F}}^{\%}$ have the best coverage in six out of ten domains, namely Blocks World, Depot, N-Puzzle, Pipesworld-NoTankage, Transport, and VisitAll. All approaches have comparatively low coverage in Rovers, and \hat{h}^{Boot} and $\hat{h}^{\text{N-RSL}}$ have low coverage in N-Puzzle when compared to the proposed approaches.

Generating samples only through regression (i.e., without solving states) and training afterward is faster than bootstrapping, as the states generated by the backward random walk must be solved with the currently learned heuristic to produce plans used as samples. Both $\hat{h}^{\text{N-RSL}}$ and the proposed methods suggest that sampling using regression with improvement strategies (such as SAI, SUI, and random samples) gives competitive results on most domains.

According to O’Toole et al. (2022), the proportion of random samples in the final 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 13, we also notice an improvement from using random samples, although smaller. The 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 %. This means that despite having the highest coverage with 20 %, using 50 % of random samples can maintain

Table 13: Mean coverage results of Ferber et al. (2022) (\hat{h}^{Boot}) and O’Toole et al. (2022) ($\hat{h}^{\text{N-RSL}}$), with results obtained from their respective papers, and our best learned heuristic trained with 100 K samples, from which 0 %, 20 % and 50 % were randomly generated.

Domain	\hat{h}^{Boot}	$\hat{h}^{\text{N-RSL}}$	$\hat{h}_{\overline{F}}$	$\hat{h}_{\overline{F}}^{20\%}$	$\hat{h}_{\overline{F}}^{50\%}$
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

similar coverage while improving sampling time, as random samples are generated much faster than in regression.

7. Conclusion

We have presented a study of sample generation and improvement strategies for training NNs to learn heuristic functions for classical planning. We have revised existing approaches to sample generation and proposed a new strategy that uses regression with breadth-first search and random walks, as well as several techniques that improve cost-to-goal estimates. A systematic analysis using complete information of small state spaces indicates that: for the samples obtained through regression with states covering diverse portions of the state space without repeated samples close to the goal works best, and enough samples of good quality translate to good search performance that can be compared to logic-based heuristics. Also, we confirm the results from O’Toole et al. (2022) showing that having randomly generated samples up to a limit in the final sample set is positive. Among the contributions, the h -value improvement strategy SUI and the adaptive regression limit \overline{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.

Acknowledgments

R. V. B. and P. P. M have contributed equally to this work. We thank the Fast Downward planning system team, and especially Patrick Ferber, Malte Helmert, and Jörg Hoffmann for sharing the source code of the Neural Fast Downward planning system, along with the dataset used in Section 6. This study was financed in part by the *Coordenação de Aperfeiçoamento*

de Pessoal de Nível Superior – Brasil (CAPES) – Finance Code 001. We acknowledge support from FAPERGS with project 21/2551-0000741-9. M. R. acknowledges support from CNPq (grants 437859/2018-5 and 310259/2022-3).

References

- Agostinelli, F., McAleer, S., Shmakov, A., & Baldi, P. (2019). Solving the Rubik’s cube with deep reinforcement learning and search. *Nature Machine Intelligence*, 1(8), 356–363.
- Alcazar, V., Borrajo, D., Fernandez, S., & Fuentetaja, R. (2013). Revisiting regression in planning. *International Joint Conference on Artificial Intelligence*, 2254–2260.
- Arfaee, S. J., Zilles, S., & Holte, R. C. (2011). Learning heuristic functions for large state spaces. *Artificial Intelligence*, 175(16-17), 2075–2098.
- Bäckström, C., & Nebel, B. (1995). Complexity results for SAS⁺ planning. *Computational Intelligence*, 11(4), 625–655.
- Bonet, B. (2013). An Admissible Heuristic for SAS⁺ Planning Obtained from the State Equation. *International Joint Conference on Artificial Intelligence*, 2268–2274.
- Bonet, B., & Geffner, H. (2001). Planning as heuristic search. *Artificial Intelligence*, 129(1-2), 5–33.
- Culberson, J. C., & Schaeffer, J. (1998). Pattern databases. *Computational Intelligence*, 14(3), 318–334.
- Dong, H., Mao, J., Lin, T., Wang, C., Li, L., & Zhou, D. (2019). Neural logic machines [Poster]. *International Conference on Learning Representations*.
- Doran, J. E., & Michie, D. (1966). Experiments with the Graph Traverser Program. *Proceedings of the Royal Society A*, 294, 235–259.
- Edelkamp, S. (2001). Planning with pattern databases. *European Conference on Planning*, 13–24.
- Ferber, P., Geißer, F., Trevizan, F., Helmert, M., & Hoffmann, J. (2022). Neural network heuristic functions for classical planning: Bootstrapping and comparison to other methods. *International Conference on Automated Planning and Scheduling*, 583–587.
- Ferber, P., Helmert, M., & Hoffmann, J. (2020). Neural network heuristics for classical planning: A study of hyperparameter space. *European Conference on Artificial Intelligence*, 325, 2346–2353.
- Frances, G., Ramírez Jávega, M., Lipovetzky, N., & Geffner, H. (2017). Purely declarative action descriptions are overrated: Classical planning with simulators. *International Joint Conference on Artificial Intelligence*, 4294–4301.
- Francés, G., Ramirez, M., & Collaborators. (2018). Tarski: An AI planning modeling framework [GitHub: <https://github.com/aig-upf/tarskio>].
- Garrett, C. R., Kaelbling, L. P., & Lozano-Pérez, T. (2016). Learning to rank for synthesizing planning heuristics. *International Joint Conferences on Artificial Intelligence*, 3089–3095.
- Gehring, C., Asai, M., Chitnis, R., Silver, T., Kaelbling, L. P., Sohrabi, S., & Katz, M. (2022). Reinforcement learning for classical planning: Viewing heuristics as dense reward generators. *International Conference on Automated Planning and Scheduling*, 32, 588–596.

- Gori, M., Monfardini, G., & Scarselli, F. (2005). A new model for learning in graph domains. *IEEE International Joint Conference on Neural Networks*, 2(2005), 729–734.
- Hart, P. E., Nilsson, N. J., & Raphael, B. (1968). A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2), 100–107.
- Haslum, P., Botea, A., Helmert, M., Bonet, B., Koenig, S., et al. (2007). Domain-independent construction of pattern database heuristics for cost-optimal planning. *AAAI Conference on Artificial Intelligence*, 1007–1012.
- Haslum, P., & Geffner, H. (2000). Admissible Heuristics for Optimal Planning. *International Conference on Artificial Intelligence Planning Systems*, 140–149.
- He, K., Zhang, X., Ren, S., & Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification. *International Conference on Computer Vision*, 1026–1034.
- He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. *IEEE Conference on Computer Vision and Pattern Recognition*, 770–778.
- Helmert, M. (2006). The Fast Downward planning system. *Journal of Artificial Intelligence Research*, 26, 191–246.
- Helmert, M. (2009). Concise finite-domain representations for PDDL planning tasks. *Artificial Intelligence*, 173(5-6), 503–535.
- Helmert, M., & Domshlak, C. (2009). Landmarks, critical paths and abstractions: What’s the difference anyway? *International Conference on Automated Planning and Scheduling*, 162–169.
- Helmert, M., Haslum, P., Hoffmann, J., et al. (2007). Flexible abstraction heuristics for optimal sequential planning. *International Conference on Automated Planning and Scheduling*, 176–183.
- Hoffmann, J., & Nebel, B. (2001). The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14, 253–302.
- Hoffmann, J., Porteous, J., & Sebastia, L. (2004). Ordered Landmarks in Planning. *Journal of Artificial Intelligence Research*, 22, 215–278.
- Hollander, M., Wolfe, D. A., & Chicken, E. (2014). *Nonparametric statistical methods* (3rd ed.). Wiley.
- Holte, R. C. (2010). Common misconceptions concerning heuristic search. *Symposium on Combinatorial Search*, 46–51.
- Karpas, E., & Domshlak, C. (2009). Cost-optimal planning with landmarks. *International Joint Conference on Artificial Intelligence*, 1728–1733.
- Kingma, D., & Ba, J. (2015). Adam: A method for stochastic optimization. *International Conference on Learning Representations*.
- Lu, L., Shin, Y., Su, Y., & Karniadakis, G. E. (2020). Dying ReLU and initialization: Theory and numerical examples. *Communications in Computational Physics*, 28(5), 1671–1706.
- O’Toole, S., Ramirez, M., Lipovetzky, N., & Pearce, A. R. (2022). Sampling from pre-images to learn heuristic functions for classical planning. *Symposium on Combinatorial Search*, 15(1), 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., . . . Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. *Advances In Neural Information Processing Systems*, 32, 8024–8035.
- Rivlin, O., Hazan, T., & Karpas, E. (2020). *Generalized planning with deep reinforcement learning*. arXiv: 2005.02305 [cs.AI].
- Samadi, M., Felner, A., & Schaeffer, J. (2008). Learning from Multiple Heuristics. *AAAI Conference on Artificial Intelligence*, 357–362.
- Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., & Monfardini, G. (2008). The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1), 61–80.
- Shen, W., Trevizan, F., & Thiebaux, S. (2020). Learning domain-independent planning heuristics with hypergraph networks. *International Conference on Automated Planning and Scheduling*, 574–584.
- Ståhlberg, S., Bonet, B., & Geffner, H. (2022). Learning general optimal policies with graph neural networks: Expressive power, transparency, and limits. *International Conference on Automated Planning and Scheduling*, 32, 629–637.
- Sturtevant, N., & Helmert, M. (2019). *Exponential-binary state-space search*. arXiv: 1906.02912 [cs.AI].
- Toyer, S., Thiebaux, S., Trevizan, F., & Xie, L. (2020). ASNets: Deep learning for generalised planning. *Journal of Artificial Intelligence Research*, 68, 1–68.
- Toyer, S., Trevizan, F., Thiebaux, S., & Xie, L. (2018). Action schema networks: Generalised policies with deep learning. *AAAI Conference on Artificial Intelligence*, 6294–6301.
- van Den Briel, M., Benton, J., Kambhampati, S., & Vossen, T. (2007). An LP-based heuristic for optimal planning. *Principles and Practice of Constraint Programming*, 651–665.
- Yu, L., Kuroiwa, R., & Fukunaga, A. (2020). Learning search-space specific heuristics using neural network. *ICAPS Workshop on Heuristics and Search for Domain-independent Planning*.