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

PEDRO PROBST MININI

Discovering and Learning Preferred Operators for Classical Planning with Neural Networks

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

Advisor: Prof. Dr. Marcus Rolf Peter Ritt

Porto Alegre
July 3rd 2023

CIP — CATALOGING-IN-PUBLICATION

Minini, Pedro Probst

Discovering and Learning Preferred Operators for Classical Planning with Neural Networks / Pedro Probst Minini. – Porto Alegre: PPGC da UFRGS, 2023.

66 f.: il.

Thesis (Master) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2023. Advisor: Marcus Rolf Peter Ritt.

1. Classical planning. 2. Heuristic search. 3. Preferred operators. 4. Machine learning. I. Rolf Peter Ritt, Marcus. II. Título.

“All this from a slice of gabagool?”

— TONY SOPRANO

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to Marcus Ritt and André G. Pereira, for their support and guidance throughout the duration of this two-year journey. I consider myself incredibly fortunate to have had such dedicated mentors.

I would also like to extend my appreciation to my colleague, Rafael V. Bettker, whose tireless dedication and ability to program long hours into the night have greatly contributed to the extensibility and success of our research.

Lastly, I want to express my deep gratitude to my mother for being the only family I have.

ABSTRACT

In a planning task, an agent must choose the most efficient action from a potentially large set of actions at each step. During a heuristic search, logic-based planners use preferred operators to reduce the branching factor significantly. This work presents a method for sampling and learning preferred operators, aiming for their applicability across the entire state space of a planning task. We demonstrate that these learned preferred operators have competitive results compared to the current best logic-based approach. Our objective is to identify ideal preferred operators, situated along the shortest paths leading to some goal. However, due to the huge size of search state spaces, we introduce a novel sampling strategy tailored for extracting preferred operators that approximate the ideal ones. Our research shows we can obtain high-quality preferred operators from a sample set covering a fraction of the state space. To understand this new category of preferred operators, we conduct controlled experiments using planning tasks where we have access to the entire state space with perfect cost-to-goal estimates. We systematically compare the proposed approach to baselines, evaluate the effectiveness of learned preferred operators learned from several sample set sizes, and assess their performance when combined with different heuristic functions.
Keywords: Classical planning. Heuristic search. Preferred operators. Machine learning.

Descoberta e Aprendizado de Operadores Preferidos para Planejamento Clássico com Redes Neurais

RESUMO

Em uma tarefa de planejamento, um agente deve escolher a ação mais eficiente de um conjunto potencialmente grande de ações em cada passo. Durante uma busca heurística, planejadores lógicos usam operadores preferidos para reduzir significativamente o fator de ramificação. Este trabalho apresenta um método para amostragem e aprendizagem de operadores preferidos, visando sua aplicabilidade em todo o espaço de estados de uma tarefa de planejamento. Demonstramos que esses operadores preferidos aprendidos têm resultados próximos à melhor abordagem lógica atual. Nosso objetivo é identificar os operadores preferidos ideais, que estão situados ao longo dos caminhos mais curtos que levam a algum objetivo. No entanto, devido ao enorme tamanho dos espaços de estado, apresentamos uma nova estratégia de amostragem adaptada para extrair operadores preferidos que aproximam os ideais. Nossa pesquisa mostra que podemos obter operadores preferidos de alta qualidade a partir um conjunto de amostras que abrange uma fração do espaço de estados. Para obter uma compreensão mais aprofundada sobre essa nova categoria de operadores preferidos, realizamos experimentos controlados usando tarefas de planejamento sobre as quais temos acesso a todo o espaço de estados com estimativas perfeitas de custo para o objetivo. Nós comparamos sistematicamente a abordagem proposta com *baselines*, avaliamos a eficácia dos operadores preferidos aprendidos com variados tamanhos de conjuntos de amostras e avaliamos o desempenho quando combinados com diferentes funções heurísticas.

Palavras-chave: Planejamento clássico. Busca heurística. Operadores preferidos. Aprendizado de máquina.

LIST OF ABBREVIATIONS AND ACRONYMS

BCE	Binary Cross-Entropy
BFS	Breadth-First Search
BSP	Backward State Space
DFS	Depth-First Search
DQ	Dual-Queue
FF	Fast-Forward
FSM	Focused Sampling Method
FSP	Forward State Space
GBFS	Greedy Best-First Search
GPU	Graphics Processing Unit
IPC	International Planning Competition
MSE	Mean Squared Error
NN	Neural Network
ResNet	Residual Network
SAI	Sample Improvement
SAS^+	Simplified Action Structures Plus
SAT	Propositional Satisfiability Problem
STRIPS	Stanford Research Institute Problem Solver
SUI	Successor Improvement
XRS	Expansion from Random Successors

LIST OF SYMBOLS

d^*	Longest distance between the goal condition and any initial state
h	Heuristic
h^*	Perfect heuristic
$\hat{h}$	Learned heuristic
h^{add}	Add heuristic
h^{FF}	FF heuristic
h^{GC}	Goal-count heuristic
po^*	Oracle of ideal preferred operators
$\hat{po}^*$	Learned ideal preferred operators
po^{FF}	FF preferred operators
$\hat{po}^{\text{FSM}}$	FSM-based learned preferred operators
$\hat{po}^{\text{G}}$	Shortest-path-graph-based learned preferred operators
$\hat{po}^{\text{G}^*}$	Shortest-path-graph-based (with h^*-values) learned preferred operators

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

CONTENTS

1 INTRODUCTION

Planning tasks can be solved through heuristic search, which systematically expands potential states in a search space based on an informed guess, or heuristic, about which states are likely to lead to a solution. In planning, preferred operators act in conjunction with heuristic functions and help reduce the number of expanded states during the planning process by prioritizing states considered advantageous.

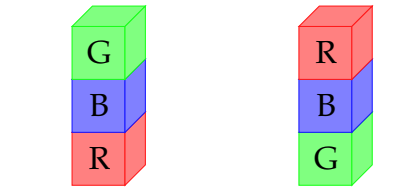
This study introduces a novel approach to deriving preferred operators in planning tasks. Unlike the traditional logic-based methods, we use a sample-based approach to discover preferred operators. By training a neural network (NN) with a sample set consisting of pairs of states and their preferred operators discovered during the sampling procedure, the NN learns to generalize preferred operators for the given planning task.

1.1 Planning

Planning involves determining a series of operators (or actions) that transform a given initial state to satisfy a goal condition. In a classical planning task, the agent acts in a fully-observable environment, i.e., with access to all relevant information of the current state of the world, such as the positions of objects. The agent starts in a initial state and needs to fulfill a specific goal condition. This is achieved by using deterministic operators to modify the current state of the world. A solution plan for the planning task is as a sequence of operators that successfully satisfy the goal condition when applied to the initial state. A state expansion involves the application of all relevant operators to a given state, thereby generating its successor states.

For example, in a Blocks World task, consider the initial state (left) and the goal state (right) shown in Figure 1.1. The agent needs to apply a sequence of operators to reach the goal state from the initial state. We can define operators such as “pick up block X”, “put block X on block Y”, and “put block X on the table.” In this example, the agent can find one of the possible plans by applying the following operators: pick up block G, put block G on the table, pick up block B, put block B on block G, and put block R on block B.

Figure 1.1 – Initial state and goal state of a Blocks World task.



(initial state) (goal state)

Planners are software systems specifically designed to find plans for planning tasks. Planners that do not rely on reasoning about SAT formulas may use several algorithms and techniques to explore the search space of possible operators and states in order to find an optimal plan, where the best possible solution is returned, or a satisfactory plan, where a suboptimal solution can be returned. Planning systems typically take as input a formal representation of the planning problem, including the initial state, goal condition, and a set of available operators. The formal representation of a planning task can be specified using various notations (Section 2.1). In this study, we focus on discovering a satisfactory plan for a given task using a best-first search algorithm.

1.2 Heuristic Search

Planners commonly use a best-first search algorithm such as greedy best-first search (GBFS) (DORAN; MICHIE, 1966). GBFS arranges states in a priority queue based on their cost-to-goal estimate (also known as heuristic value or h-value) provided by the heuristic function. It first expands states with the lowest cost-to-goal estimate to find a solution. Various domain-independent heuristics effectively compute the cost-to-goal estimate of a state by using domain logic, which has information that permits reasoning about operators and other rules, such as mutual exclusive relations (mutexes) that indicate infeasible states. These heuristics are based mainly on techniques such as abstractions (CULBERSON; SCHAEFFER, 1998), delete relaxation (HOFFMANN; NEBEL, 2001), and landmarks (HOFFMANN et al., 2004; HELMERT; DOMSHLAK, 2009). The heuristic function is the most important component of a planner since it guides the search.

1.3 Preferred Operators

Operators considered advantageous for specific reasons, such as generating states closer to the goal, are referred to as preferred operators (HELMERT, 2006; RICHTER; HELMERT, 2009). These operators are used alongside heuristic functions to assist planners in minimizing the number of expanded states when solving a planning task. By prioritizing the expansion of states generated by preferred operators, planners benefit from additional guidance, often resulting in a higher success rate for solving tasks than relying only on the heuristic function. Learning preferred operators shares similarities with learning policies, as both involve selecting actions more likely to result in desirable outcomes. The existing methods for identifying preferred operators are currently limited to logic-based approaches. The current most effective method uses the preferred operators computed alongside the Fast-Forward (FF) heuristic (HOFFMANN; NEBEL, 2001), as implemented in the Fast Downward planning system (HELMERT, 2006). Planners incorporating preferred operators emerged as winners in the satisficing track of the International Planning Competition (IPC) in the years 2004 (HELMERT, 2006), 2008 (RICHTER; WESTPHAL, 2010), 2011 (RICHTER et al., 2011), and 2018 (SEIPP; RÖGER, 2018).

1.4 Learning with Neural Networks

Learning with NNs refers to a machine learning approach that involves designing and training interconnected layers of artificial neurons, known as neural networks, inspired by the biological brain. It involves learning hierarchical representations of data, where each layer in the network progressively extracts more complex and abstract features. Through the usage of NNs, learning algorithms can automatically discover and capture patterns from large-scale datasets in various tasks, including image classification, speech recognition, and natural language processing.

Learning models based on NNs can learn in different ways. This study focuses on supervised learning, which uses labeled datasets to classify or make predictions. In this case, a dataset used to train an NN can be represented as multiple pairs in the format (y, x). The target y refers to the desired output or

label associated with a specific input instance x. With supervised learning, an NN can effectively learn and generalize from the provided labeled data to make accurate predictions or classify new, unseen inputs.

1.5 Learning in Planning

In recent years, interest in using NNs to learn heuristic functions (FERBER et al., 2020; YU et al., 2020; SHEN et al., 2020; FERBER et al., 2022; O'TOOLE et al., 2022; BETTKER et al., 2022) or policies (TOYER et al., 2018; TOYER et al., 2020; STÅHLBERG et al., 2022) to solve planning tasks has increased. The general approach for supervised methods is to generate a set of samples as pairs of states and cost-to-goal values and use them for training an NN. However, it is challenging to learn effective heuristic functions since state spaces tend to grow exponentially in size as the amount of information needed to describe them increases, but the portion of the state space that we can actually sample is relatively small. Logic-based heuristics can be applied to any domain, while learned heuristics depend on the learning model, and even with domain-independent models, transfer learning can be difficult (SHEN et al., 2020). Moreover, learned heuristics are generally slow to compute, thus they need to be more informed, i.e., expand fewer states when compared to logic-based heuristics to reach the goal. These characteristics also apply to learning preferred operators.

1.6 Contributions

This study represents the first attempt to discover preferred operators from a sample set and use an NN to learn them. We present a new sampling method and a novel sample-based technique for identifying preferred operators. The technique involves backward search from the goal condition (also known as regression), constructing a graph with sampled states representing their successor-predecessor relationships, and determining the operators used to reach the goal condition as preferred operators for each state. We show that an NN can learn the preferred operators from a subset of the state space and effectively extend this learning to the entire state space across diverse planning domains. Further-

more, the proposed approach outperforms the current best logic-based preferred operator method in the benchmark tasks. In particular, this work presents:

- A novel method based on shortest path graphs to discover preferred operators in an existing sample set (Section 3.2).
- A new sampling method tailored for discovering preferred operators (Section 3.3).
- An analysis of the quality of the learned preferred operators and a comparison to existing logic-based methods (Chapter 4).

2 BACKGROUND

This chapter provides essential information for comprehending the subsequent chapters of this work.

2.1 Classical Planning Notation

In this section we present two ways to represent a classical planning task. The first one, STRIPS (FIKES; NILSSON, 1971), represents a planning task using propositional facts (Boolean variables). The second way, SAS$^+$ (BÄCKSTRÖM; NEBEL, 1995), represents a planning task using multi-valued state variables.

2.1.1 STRIPS

Definition 1 (STRIPS Planning Task)**.** *A planning task in STRIPS representation is defined by* $\Pi = \langle \mathcal{F}, \mathcal{O}, s_0, s^* \rangle$*, where* $\mathcal{F}$ *has all the possible facts (propositions) that can be used to describe a state,* $\mathcal{O}$ *is a set of operators,* $s_0 \subseteq \mathcal{F}$ *is an initial state, and* $s^* \subseteq \mathcal{F}$ *is the goal condition that specifies the facts that should be true to solve the task. A state* s *is defined as* $s \subseteq \mathcal{F}$*, where each fact in* s *can be true (*$f \in s$*) or false (*$f \notin s$*). An operator* $o \in \mathcal{O}$ *is defined by a triple* $\langle \mathrm{pre}(o), \mathrm{add}(o), \mathrm{del}(o) \rangle$ *that specifies its precondition, add-effects and delete-effects, respectively, which are denoted as sets of facts.*

In STRIPS, we say that operator o is applicable to state s if its preconditions are satisfied by s, i.e., $\mathrm{pre}(o) \subseteq s$, and produces a successor state $s' = \mathrm{succ}(s, o) = (s \setminus \mathrm{del}(o)) \cup \mathrm{eff}(o)$. In other words, we progress a state s with operator o by setting the propositions in $\mathrm{del}(o)$ to false and in $\mathrm{add}(o)$ to true. The set of successor states of s is denoted by $\mathrm{succ}(s) = \{\mathrm{succ}(s, o) \mid o \in \mathcal{O}, \mathrm{pre}(o) \subseteq s\}$.

STRIPS formulas use conjunction of propositions with logical connectives to express compound preconditions and effects. STRIPS has no negated preconditions, and it is not possible to directly specify conditional effects, where the effect of an action depends on the initial state. However, we can simulate conditional effects by creating new operators in the planning task.

Definition 2 (Plan)**.** *A sequence of operators* $\pi = (o_1, \ldots, o_k)$ *is valid for a state* s_0 *if produces a sequence of states* $s_1, \ldots, s_k$ *such that* $s_i = \mathrm{succ}(s_{i-1}, o_i)$*. A sequence* π *for*

the initial state s_0 is called a plan if $s^* \subseteq s_k$. The plan length is defined as $|\pi|$. Among all the valid plans, the one with the minimum length is referred to as an optimal plan π^*. An optimal plan is the shortest plan that successfully achieves the goal condition from the initial state. Since in this work we only consider unitary cost operators, the plan cost is equal to the plan length.

Heuristics based on delete-relaxation obtain the heuristic from a *relaxed* version of the planning task. A relaxed planning task in STRIPS is defined as $\Pi^+ = \langle \mathcal{P}, \mathcal{O}', s_0, s^* \rangle$, where $\mathcal{O}' = \{\langle \mathrm{pre}(o), \mathrm{add}(o), \emptyset \rangle \mid o \in O\}$, i.e., the delete-effects of the planning task are ignored. Relaxed tasks can be solved efficiently even though finding the optimal solution is $\mathcal{NP}$-hard (BYLANDER, 1994). For example, the heuristic h^{add} approximates the perfect heuristic value for a state s as the sum of the costs of achieving each proposition in s^* independently of the others.

2.1.2 SAS$^+$

We also use the SAS$^+$ representation to describe classical planning tasks independent of any particular domain. SAS$^+$ shares similarities with STRIPS, but it allows state variables to have an arbitrary and potentially non-binary finite domain. These multi-valued variables can express mutexes that are not explicitly recognized in STRIPS. Suppose we have a planning task with a robot in a grid with n possible locations it needs to visit. With STRIPS, we need n facts to indicate all the possible locations where the robot can be, and the STRIPS representation does not explicitly capture the mutex that the robot cannot exist in multiple locations simultaneously. In SAS$^+$, this information can be naturally expressed using a single multi-valued variable for the location of the robot. The variable has a finite domain consisting of the n possible locations, enforcing that the robot can only occupy one location at a time. Thus, only one proposition from the n possibilities can be true in any state.

Definition 3 (SAS$^+$ Planning Task)**.** *A SAS$^+$ task is defined as $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s^* \rangle$, where $\mathcal{V}$ is a set of state variables, and each variable $v \in \mathcal{V}$ has a finite domain $\mathrm{dom}(v)$, that represents the possible mutually exclusive values of each variable, $\mathcal{O}$ is a set of operators where each operator $o \in \mathcal{O}$ is defined as a pair of preconditions and effects $(\mathrm{pre}(o), \mathrm{eff}(o))$, both partial states s defined as a partial function $s : \mathcal{V} \to \mathcal{D}$, where*

$\mathcal{D} = \cup_{v \in \mathcal{V}} \mathrm{dom}(v)$, *such that* $s(v) \in \mathrm{dom}(v)$ *whenever* $s(v)$ *is defined. Otherwise,* $s(v)$ *is undefined and written as* $s(v) = \perp$. *Given a partial state* s, $var(s) \in \mathcal{V}$ *is a finite set which lists all variables assigned in* s. *A (complete) state* s *is a partial state defined on all variables in* $\mathcal{V}$, *i.e,* $var(s) = \mathcal{V}$. *The state* s_0 *defines the initial state, and the partial state* s^* *defines the goal condition.*

An operator $o \in \mathcal{O}$ is applicable to a state s if its preconditions are fulfilled by s, i.e, $s \subseteq \mathrm{pre}(o)$, and it generates a successor state $s' = \mathrm{succ}(s, o) := \mathrm{eff}(o) \circ s$. Here, $s' = t \circ s$ is defined as $s'(v) = t(v)$ for all v such that $t(v)$ is defined, and for all other cases, $s'(v) = s(v)$. The set of all successor states resulting from state s is denoted by $\mathrm{succ}(s) = \{\mathrm{succ}(s, o) \mid o \in \mathcal{O}, s \subseteq \mathrm{pre}(o)\}$. A state variable can never be made undefined once made defined by an operator.

We can represent planning tasks in STRIPS using SAS$^+$ by converting each fact to a state variable with domain true and false. On the other hand, SAS$^+$ can be represented in STRIPS by converting each possible variable assignment to a fact. Note that SAS$^+$ supports partial assignments while STRIPS assumes all states are complete (unmentioned facts are considered false). SAS$^+$ tasks are generally more concise than STRIPS tasks. In STRIPS, if we have n facts, the size of the set of states would be 2^n. In SAS$^+$, with n variables, the size of the set of states would be $dom(v_1) \times dom(v_2) \times \ldots \times dom(v_n)$.

2.2 Regression

Progression involves determining the successor states of the current state by applying a sequence of operators, and regression involves determining predecessor states that can lead to the current partial state. In regression, an operator o is considered *relevant* for partial state s if $\mathrm{eff}_r = \mathrm{dom}(\mathrm{eff}(o)) \cap \mathrm{dom}(s) \neq \emptyset$, and *consistent* if $s \subseteq \mathrm{eff}(o)|_{\mathrm{eff}_r}$. 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. An operator o is *backward applicable* in partial state s if it is both relevant and consistent with s, and it leads to a predecessor $\mathrm{pred}(s, o)$ given by $\mathrm{pred}(s, o) = \mathrm{pre}(o) \circ (s|_{\mathrm{dom}(s) \setminus \mathrm{eff}_r})$. Note that $s \subseteq \mathrm{succ}(\mathrm{pred}(s, o), o)$, but can differ from s since the operator o may have changed the values of variables that were not defined in s.

A partial state s has predecessors $\mathrm{pred}(s) = \{\mathrm{pred}(s, o) \mid o \in \mathcal{O}\}$, where o_r is applicable to s. A regression sequence from state s_0 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. Since the goal is a partial state, a valid regression sequence $\rho = (o_1, \ldots, o_k)$ will generate a sequence of partial states that can reach the goal in at most k steps. Note that progression and regression in planning are asymmetric because each state in the regression (backward) state space can represent a set of states in the progression (forward) state space (ALCAZAR et al., 2013).

2.3 State Spaces

Let S be a set of states, $s_0 \in S$ be an initial state, $s^* \in S$ be the goal condition, and $\mathrm{succ} : S \to 2^S$ be a successor function, which maps each state to a set of possible successor states and determines the available transitions. A state space is a tuple SP $= \langle S, s_0, s^*, \mathrm{succ} \rangle$.

Definition 4 (State Space of Π)**.** *For any planning task Π with states S, initial state s_0, goal condition s^*, and successor function* succ*, the corresponding state space of Π is denoted as SP$(\Pi) = \langle S, s_0, s^*, \mathrm{succ} \rangle$.*

The forward state space (FSP) refers to the set of states that can be reached from the initial state s_0 by applying the successor function succ in the forward direction. It represents the states that can be reached forward from the initial state towards the goal condition.

Definition 5 (Forward State Space of Π)**.** *The forward state space for a planning task Π is denoted as FSP$(\Pi) = \langle S_F, s_0, s^*, \mathrm{succ} \rangle$, where S_F is the set of states reachable from the initial state, and $\mathrm{succ} : S_F \to 2^{S_F}$ is the successor function that maps each state to a set of possible successor states and determines the available transitions in the forward direction.*

In other words, FSP(Π) is defined as the subset of states and transitions within SP that are relevant to the planning task Π and the forward expansion towards the goal condition. States from the FSP are expanded when solving a task, for example by using a best-first search algorithm.

The backward state space (BSP), on the other hand, refers to the set of states that can be reached from the goal condition s^* by applying a predecessor function pred. It represents the states that can be reached backward from the goal condition towards the initial state (regression).

Definition 6 (Backward State Space of Π)**.** *The backward state space of a planning task Π is denoted as $BSP(\Pi) = \langle S_B, s_0, s^*, \mathrm{pred}\rangle$, where S_B is the set of states reachable from the goal condition and $\mathrm{pred} : S_B \to 2^{S_B}$ is a predecessor function that maps each state to a set of possible predecessor states and determines the available transitions in the backward direction.*

2.4 Heuristic Functions

A heuristic function $h : \mathcal{S} \to \mathbb{R}^{\geq 0} \cup \{\infty\}$ estimates the optimal plan length from a state s to the goal s^*, where the perfect heuristic function is defined as $h^*(s) = |\pi_s^*|$, i.e., the length of the optimal plan from s to s^*. Heuristic functions are used to guide search algorithms, optimization techniques, or decision-making processes by providing informed estimates or approximations based on available information or problem-specific knowledge. The goal of a heuristic function is to efficiently guide the search or decision-making process towards more promising paths or solutions, even in the absence of complete or perfect information. A heuristic function can have several properties that indicate its quality, such as:

- Admissibility: $h(s) \leq h^*(s)$, i.e., the heuristic never overestimates the true cost-to-goal.
- Consistency (or monotonicity): $h(s) \leq h(s') + cost(o)$ for all transitions from a state s to a successor s', i.e., $s \xrightarrow{o} s'$, where $cost(o)$ is the cost of applying operator o to reach s' from s.
- Goal-awareness: $h(s) = 0$ for all goal states.
- Safeness: $h(s) = \infty$ implies $h^*(s) = \infty$, for example in dead-ends.

Heuristics are typically derived from a model of the task, such as the SAS$^+$ model introduced earlier, but can also be obtained by learning the map of some state s to its heuristic value $h(s)$, where the desired output of the NN can be either the direct cost-to-goal estimates or some form of encoding representing these estimates.

In this study, we use a propositional representation of a state to learn heuristic functions and preferred operators (FERBER et al., 2020; YU et al., 2020; FERBER et al., 2022; O'TOOLE et al., 2022; BETTKER et al., 2022). We use the notation from Bettker et al. (2022). Consider a planning task $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s^* \rangle$, where $\mathcal{V} = \{v_1, \ldots, v_n\}$ is a set of variables, and $D(v_i) = \{d_{i1}, \ldots, d_{i,s_i}\}$ the domains of variable v_i, where $i \in [n]$. A state s is represented as a sequence of facts $\mathcal{F}(s) = (f_{11}, f_{12}, \ldots, f_{1,s_1}, \ldots, f_{n1}, f_{n2}, \ldots, f_{n,s_n})$. Each fact $f_{ij} = [s(v_i) = d_{ij}]$ corresponds to a variable v_i taking on a specific value d_{ij} in state s. If the variable v_i has the assigned value d_{ij} in state s, then the fact f_{ij} is considered true. The facts $\mathcal{F}_i = \{f_{i1}, \ldots, f_{i,s_i}\}$ associated with variable v_i must adhere to the consistency condition $\sum_{f \in \mathcal{F}i} f \leq 1$. This means that each variable can take at most one value. When v_i is undefined, $\sum_{f \in \mathcal{F}_i} f = 0$.

For example, let $\mathcal{F}$ be a set of facts, and let $f_i, f_j \in \mathcal{F}$ be two facts. We say that f_i and f_j are mutex if $\neg(f_i \wedge f_j)$ holds, i.e., f_i and f_j cannot both be true at the same time in any valid state of the planning problem. We use the notation $\mathrm{mutex}(\mathcal{F})$ to denote when the constraint $\sum_{f \in \mathcal{F}}[f] \leq 1$ must be satisfied for the states of the planning task.

2.5 Greedy Best-First Search

Greedy best-first search (GBFS, Algorithm 1) is a best-first search algorithm typically used by planners to solve planning tasks by expanding the FSP. GBFS organizes states in a priority queue based on their cost-to-goal estimate, which is determined by a heuristic function h. GBFS expands states with the *lowest* cost-to-goal estimate first in order to find a solution. Unlike the A* algorithm (HART et al., 1968), GBFS does not consider the cost of the path taken (g-value). This *can* make GBFS efficient when the heuristic function is accurate, but it sacrifices optimality guarantees, whereas A* is optimal when the heuristic used is both admissible and consistent.

Algorithm 1 Greedy best-first search

```
1:  procedure GBFS(s_0, s*, h)
2:      Q ← priority queue ordered by lowest h
3:      if h(s_0) < ∞ then
4:          Insert s_0 into Q with priority h(s_0)
5:          Mark the initial state s_0 as visited
6:      end if
7:      while Q is not empty do
8:          s ← state with the highest priority in Q
9:          Remove s from Q
10:         if s satisfies the goal condition s* then
11:             return the solution
12:         end if
13:         S' ← {s' | s' ∈ succ(s)}
14:         for all successor states s' ∈ S' do
15:             if s' has not been visited and h(s') < ∞ then
16:                 Mark s' as visited
17:                 Insert s' into Q with priority h(s')
18:             end if
19:         end for
20:     end while
21:     return failure (no solution found)
22: end procedure
```

2.6 Preferred Operators

Preferred operators can be described as operators that, given a particular state s, tend to generate successors more likely to satisfy the goal condition when compared to other successors of s. Preferred operators provide a way to prioritize the expansion of certain states over others during the search. Although the method of identifying preferred operators varies, Hoffmann and Nebel (2001) introduced the first approach in the original Fast-Forward (FF), where preferred operators are computed alongside the FF heuristic. Specifically, the FF planner computes a relaxed planning graph (Algorithm 2) that represents the relaxed task, ignoring delete-effects. From the relaxed planning graph, FF extracts the relaxed plan (Algorithm 3) with its length as the cost-to-goal estimate for a state s. To extract the relaxed plan, the algorithm marks the goal facts that need to be achieved at each layer k of the relaxed planning graph, then iterate from the last layer to the initial layer, marking the actions at layer k that achieve the goal facts of the same layer. The preferred operators, then called *helpful actions*, are de-

Algorithm 2 Computing the relaxed planning graph

```
1: procedure RELAXEDPLANNINGGRAPH(relaxed planning task Π⁺)
2:     P_0 ← s_0 # Initial proposition layer.
3:     k ← 0 # Index of the current layer.
4:     while s* ⊈ P_k do
5:         k ← k + 1
6:         # Computes the action layer O_k at layer k, i.e., all the actions
7:         # with the preconditions satisfied in the preceding layer.
8:         O_k ← {o ∈ O | pre(o) ⊆ P_{k−1}}
9:         P_k ← P_{k−1}
10:        for all o ∈ O_k do
11:            # Computes the proposition layer P_k at layer k.
12:            P_k ← P_k ∪ add(o)
13:        end for
14:        if P_k = P_{k−1} then
15:            return failure
16:        end if
17:    end while
18:    G⁺ ← [P_0, O_1, P_1, ..., O_k, P_k] # The relaxed planning graph.
19:    return G⁺
20: end procedure
```

fined as the set $\{o \mid pre(o) \subseteq s, add(o) \cap S^*_{k=1} \neq \emptyset\}$, where $S^*_{k=1}$ denotes the set of goal facts at layer 1 of the relaxed plan (note that the preconditions of operators that add a fact of the goal are also inserted to S^* as subgols). In summary, the preferred operators of the FF planner are a subset of all possible operators that satisfy two conditions: they can be applied given the current state and contribute to achieving at least one goal fact at the first layer of the relaxed plan.[1]

In the original implementation of the FF planner, the preferred operators prune the search space and only evaluate successors generated via preferred operators. However, this makes the search incomplete (RICHTER; HELMERT, 2009), i.e., it does not guarantee a solution or determine if there is none, and FF restarts without preferred operators if they fail.

The current approach to extract preferred operators is the one implemented in the Fast Downward planning system (HELMERT, 2009), based on domain transition graphs, compatible with SAS$^+$, instead of planning graphs as previously described, where the preferred operators obtained with the computation of the FF heuristic are the current best. Fast Downward extends the "vanilla" GBFS algorithm to support preferred operators and ensure completeness. This is

[1] Algorithms 2 and 3 were modified from Wickler and Tate (2013).

Algorithm 3 Extracting the relaxed plan

procedure RELAXEDPLAN(relaxed planning graph G^+, goal facts s^*)
  $P \leftarrow$ proposition layers of G^+
  $O \leftarrow$ action layers of G^+
  $plan \leftarrow \emptyset$
  # firstlayer(x,y) returns the number of the first layer at which x appears in y.
  # M = maximum(index of the first layer where each fact of s^* first appears in P).
  $M \leftarrow \max\{\text{firstlayer}(s_i^*, P) \mid s_i^* \in s^*\}$
  for $k \leftarrow 0$ **to** M **do**
    # S_k^* are the goal facts that need to be achieved in P_k.
    $S_k^* \leftarrow \{s_i^* \in s^* \mid \text{firstlayer}(s_i^*, P_k) = k\}$
  end for
  for $k \leftarrow M$ **to** 1 **do**
    for all $s_k^* \in S_k^*$ **do**
      # Selects the action o that achieves the goal fact s_k^*
      # and appears for the first time in layer k.
      $o \leftarrow \text{firstlayer}(o, O_k) = k \mid s_k^* \in add(o)$
      $plan \leftarrow plan \cup \{o\}$
      # Now add the preconditions of o as sub-goals to S^*
      # in the layer where p first appears.
      for all $p \in pre(o)$ **do**
        $S^*_{firstlayer(p,P)} \leftarrow S^*_{firstlayer(p,P)} \cup \{p\}$
      end for
    end for
  end for
  return $plan$
end procedure

achieved by introducing a dual-queue approach: the "default queue" receives all generated states (default behavior without preferred operators), while the "preferred queue" only accepts states generated by preferred operators. We call this DQ-GBFS (dual-queue greedy best-first search). In this version of the algorithm, the expansion of states occurs alternately from both queues or may use boosting (RICHTER; HELMERT, 2009). With a boost value n, when a state with a lower h-value than any previously expanded state is encountered during the search (meaning the search progresses) the preferred queue is used for the next n expansions, as long as it has elements. The boost value is cumulative, so each time the search progresses, n expansions are added to the remaining number of subsequent expansions from the preferred queue.

In Fast Downward, the initial expansion in the search originates from the default queue, so even if we have preferred operators that always generate a successor closest to the goal, an inaccurate heuristic can result in a suboptimal plan.

Furthermore, there are cases where a preferred operators generates a successor state that has already been generated. Consequently, the state is not added to the preferred queue, which means it cannot be expanded first (as the preferred queue has higher priority), and another state, potentially further from the goal, is expanded instead.

2.7 Neural Networks

A multi-layer perceptron is a commonly used NN architecture comprising an input layer, one or more hidden layers, and an output layer. The neurons within the hidden layers typically apply an activation functions to the weighted sum of their inputs, which can introduce linear and nonlinear transformations. A bias term represents a constant value that is optionally added to the weighted sum of inputs of a neuron before applying the activation function, introducing an offset in the activation that allows the NN to account for any systematic errors or deviations in the data. The weights and biases of the neurons are learned through a process called backpropagation, which optimizes a loss function using gradient descent or its variations. For more information on backpropagation and gradient descent, refer to Goodfellow et al. (2016).

The mean squared error (MSE) loss function is commonly used for regression problems, where the goal is to predict continuous values. By minimizing the MSE, the NN aims to make its predictions as close as possible to the actual values.

Definition 7 (Mean Squared Error)**.** *Given a prediction $\hat{y}$ and the corresponding target value y, the MSE loss is computed as the mean of the squared differences between the prediction and the target:*

$$MSE(\hat{y}, y) = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2,$$

where $\hat{y}_i$ represents the i-th element of the prediction vector $\hat{y}$, y_i represents the i-th element of the target vector y, and N is the total number of elements in the vectors.

The binary cross-entropy (BCE) loss function is commonly used for binary classification problems, where the goal is to predict a binary outcome, or multi-label classification problems where there can be more than one correct outcome (TSOUMAKAS; KATAKIS, 2007).

Definition 8 (Binary Cross-Entropy)**.** *Given a prediction $\hat{y}$ and the corresponding binary target value y, the BCE loss is computed as the average of the element-wise cross-entropy between the prediction and the target:*

$$BCE(\hat{y}, y) = -\frac{1}{N}\sum_{i=1}^{N}\left[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)\right],$$

where $\hat{y}_i$ represents the i-th element of the prediction vector $\hat{y}$, y_i represents the i-th element of the target vector y, and N is the total number of elements in the vectors.

In this study, we use a feedforward NN with residual blocks. Let us consider a feedforward NN with L layers. For simplicity, assume each layer has the same number of neurons, denoted by N. The input to the network is represented x, and the output is denoted by y. The activation function of a neuron in the l-th layer is denoted as $a^l(\cdot)$, and the weights connecting the i-th neuron in layer $l-1$ to the j-th neuron in layer l are denoted as w_{ij}^l. The bias of the j-th neuron in layer l is denoted as b_j^l. The output of the j-th neuron in layer l is given by:

$$z_j^l = \sum_{i=1}^{N} w_{ij}^l a_i^{l-1} + b_j^l$$

The activation of the j-th neuron in layer l is then computed as:

$$a_j^l = a^l(z_j^l)$$

Fig. 2.1 shows an example of a simple fully-connected NN. Common activation functions include sigmoid, rectified linear units (ReLU), and tanh (Fig. 2.2).

The data "fed" to an NN is typically divided into two sets. The training set is used to train the model by adjusting its parameters based on input samples and corresponding target values. The validation set is used to evaluate the performance of the model and detect overfitting during training, which occurs when a model excessively fits the training data but fails to generalize to unseen data.

An epoch is an iteration of the entire training dataset, involving feeding each sample, updating weights based on the loss function, and performing forward and backward propagation. Early stop, or patience, halts training if there is no improvement in a chosen metric for a specified number of consecutive epochs, preventing overfitting and and preserving the performance of the model at the point of best validation metric.

Figure 2.1 – A neural network with n input neurons, two hidden layers with m neurons, and an output layer with k neurons.

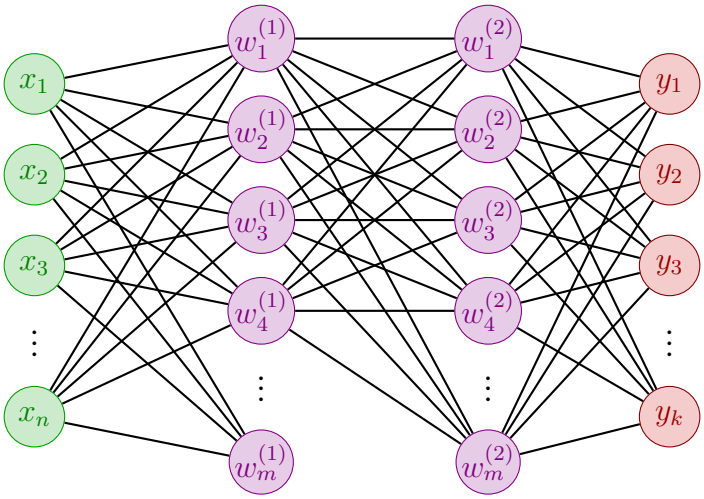
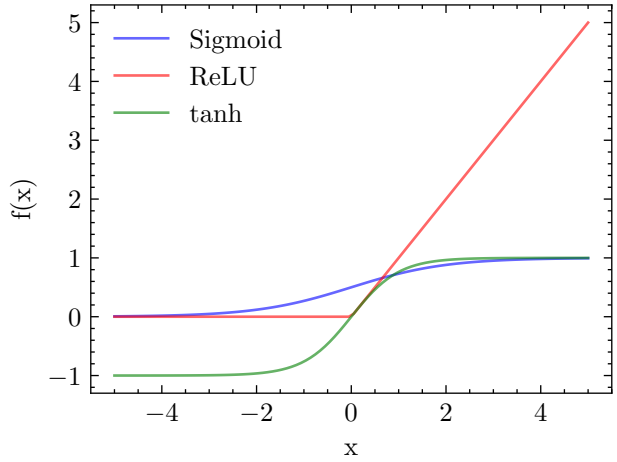


Figure 2.2 – Sigmoid, ReLU, and tanh activation functions.



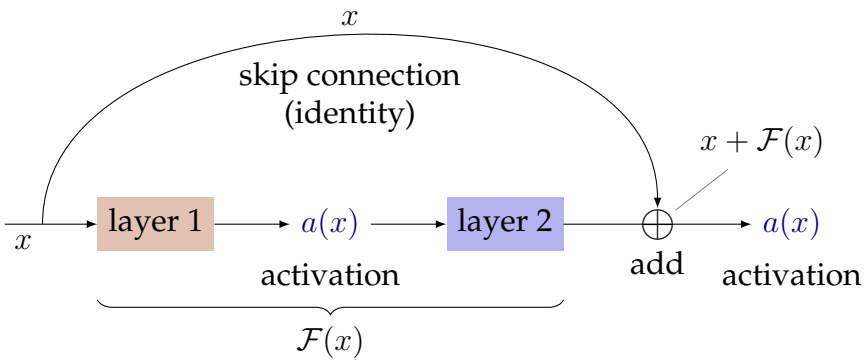
Definition 9 (Early Stop). *Let M represent the chosen metric (e.g., validation loss) and n denote the number of consecutive epochs without improvement allowed. Given the current epoch t, the training is stopped if the following condition is met:*

$$M(t) > \min\{M(t-1), M(t-2), \ldots, M(t-n)\}$$

where $M(t)$ represents the metric value at epoch t.

During training, data can be fed in batches, which are subsets of training samples processed together in each epoch iteration. This method enhances computational efficiency by simultaneously processing multiple samples, instead of updating network weights after each training sample (which is inefficient). Larger batch sizes enable parallel processing but demand more memory, while

Figure 2.3 – A residual block with two hidden layers.



smaller batch sizes consume less memory but may lead to slower training convergence due to more frequent weight updates.

Standard NNs may suffer from the vanishing gradient problem (HOCHREITER, 1991). The gradients tend to diminish as they propagate backward through multiple layers, making it challenging for earlier layers to learn meaningful representations. This problem hampers the optimization process and restricts the overall performance of the network.

Residual Neural Networks (ResNets) (HE et al., 2016) can reduce the vanishing gradient problem. ResNets use skip connections that bypass layers, allowing the information to flow directly from one layer to another. This bypassing mechanism mitigates the vanishing gradient problem and facilitates the training of deep networks. Fig. 2.3 has a visual representation of a small *residual block* with a skip connection.

Definition 10 (Skip Connection)**.** *Let us consider a ResNet architecture with L layers. The output of the l-th layer is denoted by a^l, and the output of the previous layer is denoted by a^{l-1}. The residual connection between the l-th and $l-1$-th layers can be represented as:*

$$a^l = a^{l-1} + \mathcal{F}(a^{l-1}, W^l),$$

where $\mathcal{F}$ represents a residual function, typically implemented as a fully connected layer, and W^l denotes the learnable parameters of this function.

2.8 Learning Heuristic Functions

There have been several studies on learning heuristic functions to solve planning tasks. These studies can be categorized into two primary approaches. The first approach heavily relies on domain logic to generate samples and instantiate networks. Its objective is to generalize across domains or a planning formalism. On the other hand, the second approach minimally uses domain logic for generating samples and focuses on generalization within a state space.

The first approach uses structured NNs with architectures specifically designed to align with the characteristics and requirements of the given input task. Examples include learning domain-independent heuristic functions with hypergraph NNs (SHEN et al., 2020), and learning policies with action schema networks (TOYER et al., 2018; TOYER et al., 2020) and graph NNs (STÅHLBERG et al., 2022). These approaches are generally applicable to state spaces that differ from the ones they were originally trained on. The major limitation of this set of approaches is the heavy reliance on domain logic, and it can also have significant computational overhead, as in the case of action schema networks. Moreover, in domain-independent approaches such as hypergraph NNs, where the network can be applied to a different domain than the one it was originally trained on, the search performance is considerably inferior when compared to logic-based heuristics in terms of coverage (number of solved tasks within a given time limit).

The second approach uses supervised learning to train a feedforward NN with pairs of states and cost-to-goal estimates. Samples can be generated with forward search from the initial state (FERBER et al., 2020), backward search from the goal (YU et al., 2020; O'TOOLE et al., 2022; BETTKER et al., 2022), or a combination of both (FERBER et al., 2022). This set of approaches requires fewer computational resources but is limited to the specific state space they were trained on. Search performance is also a problem, since these approaches typically solve less tasks than simple logic-based heuristics such as goal-count. An advantage is that this set of approaches relies on domain logic to a lesser extent, and only during sample generation, such as determining applicable operators and mutexes.

Regarding sampling by backward search, Yu et al. (2020) use depth-first search, O'Toole et al. (2022) use random walks, and Bettker et al. (2022) use a combination of breadth-first search and random walks. In these cases, the cost-

to-goal estimates are assigned as the lowest distance from the goal at which the states were generated. Ferber et al. (2022) use a combination of backward and forward searches, using bootstrap learning (ARFAEE et al., 2011). Specifically, Bettker et al. (2022) introduce several cost-to-goal improvement methods to enhance sample quality and emphasize that the effectiveness of learned heuristics relies on two crucial factors: a sample set that encompasses diverse regions of the state space and accurate cost-to-goal estimates; one without the other is insufficient.

In this study, we aim to use minimal domain logic. As such, we use learned heuristic functions and learned preferred operators in most experiments. In particular, we follow the second set of approaches described previously. In the following sections, we introduce relevant concepts for the proposed approach to compute sample-based preferred operators in Chapter 3.

2.8.1 Generating Samples with FSM

To learn heuristic functions, sampled states are labeled with a cost-to-goal estimate. In regression-based methods, the value assigned to a sampled state is determined by its distance to the goal condition. In this work, we generate samples by regression, expanding partial states in the backward state space BSP. Precisely, we follow the best-performing approach used by Bettker et al. (2022), using a combination of breadth-first search (BFS) with multiple random walk (RW) rollouts, named "focused sampling method" (FSM), which aims to achieve good coverage near the goal (BFS) while obtaining a diverse set of samples from the remaining state space (RW).

A regression rollout refers to a sequence of partial state expansions, concluding under two conditions: when the last expanded state has no predecessors or when it reaches the regression limit (depth) L. The process of generating samples halts once the desired number of samples N is reached. Random walks can have multiple rollouts due to the regression limit L, whereas BFS has a single rollout. Repeated states are not sampled and expanded during each RW rollout to avoid cycles during a backward search, and the current rollout ends abruptly if only repeated states are available to continue the search. However, repeated states are permitted between rollouts. Bettker et al. (2022) set the regression limit to $L = \lceil L_F / \overline{\text{eff}} \rceil$, where $\overline{\text{eff}} = \sum_{o \in \mathcal{O}} |\,\text{eff}(o)|/|\mathcal{O}|$, i.e., the number of facts per mean

number of effects in the operators of the input task. Unlike using a fixed L (YU et al., 2020; O'TOOLE et al., 2022), this adaptive L aims to approximate the longest distance d^* between the goal condition and any potential initial state.

FSM consists of two phases. The first phase uses BFS to generate p_{FSM} of the N samples. The BFS expands a state from layer k and generates n states from layer $k+1$. These generated states are sampled only when $N + n \leq p_{\text{FSM}}N$; otherwise, no states are sampled, and BFS expands another state. The set of unexpanded states, i.e., leaves, is denoted as Q. The second phase involves multiple random walk rollouts starting from randomly selected states in Q. This process continues until the sample set reaches N. During the random walk phase, states already sampled in the BFS phase are not sampled again.

After finishing regression, Bettker et al. (2022) improve the cost-to-goal estimates of each sampled partial state, and complete them to full states (Section 2.8.2). Generally, accurate cost-to-goal estimates result in improved learned heuristics, leading to fewer expanded states during a search. However, this correlation is not always guaranteed (HOLTE, 2010). To enhance the cost-to-goal estimates of each sampled state, Bettker et al. (2022) developed two methods used in this study. The first method, SAI (Section 2.8.3), minimizes estimates across repeated samples, while the second method, SUI (Section 2.8.4), minimizes estimates across the successors of samples.

2.8.2 Sample Completion

Regression sampling produces a set of partial states with undefined variables. However, during the search, the NN is trained on complete states and expects complete states as input. Each partial state can be completed by assigning a value $s(v) \in \text{dom}(v)$ to all fact pairs $(v, s(v))$ where $s(v) = \bot$. As Bettker et al. (2022) and Ferber et al. (2022), we use a method that assigns a random value $s(v) \in \text{dom}(v)$ to each undefined variable to complete partial states, ensuring that the assigned values satisfy mutexes derived from the planning task. We process the undefined variables in random order and assign them random values that do not violate the mutexes. The undefined variables are set to false if we cannot complete the state after $10\,\text{K}$ attempts. We use the mutexes automatically deduced by Fast Downward (HELMERT, 2006; HELMERT, 2009).

2.8.3 Sample Improvement

It is possible to generate duplicate states with different estimates due to multiple random walk rollouts. Thus, we update the cost-to-goal estimate for each sampled state s to the minimum estimate $h(s) = \min\{h_i \mid s = s_i, i \in [N]\}$. This procedure is called "sample improvement" (SAI), and is applied twice: first on partial states after regression is complete, and then on the completed states, as two partial states can be transformed into the same state.

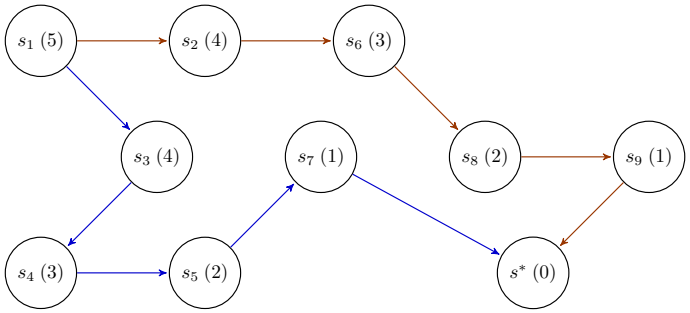
2.8.4 Successor Improvement

Sampling neighboring states in the state space is also common, especially for states close to the goal. Using the following approach, we can leverage this to enhance the cost-to-goal estimates, starting by constructing a graph G with the relations between all partial states in the sample set.

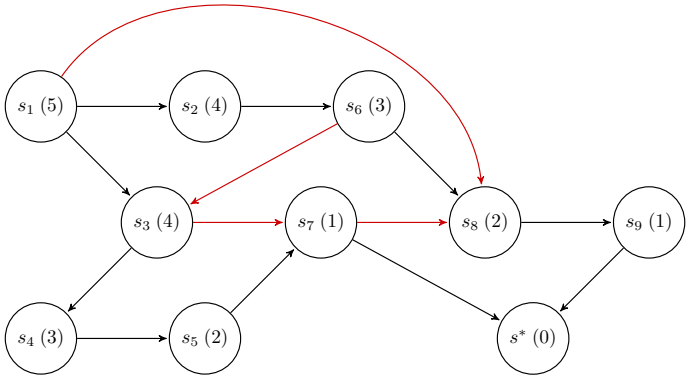
Definition 11 (Sample Set Graph)**.** *A sample set graph is a directed and labeled graph $G = (V, A)$ defined by the states obtained in the sampling. Each vertex is labeled by a sampled partial state, i.e., $V = \{s_i \mid i \in [N]\}$. (Note that the undefined variables of a partial state can typically be completed in multiple ways, so each vertex represents a set of complete states.) Set A contains an arc (s, t) for each operator $o \in \mathcal{O}$ where $\mathrm{succ}(s, o) \subseteq t$.*

With partial states generated by regression, we can always find at least one successor, except for the goal s^*. We compute the shortest paths to the goal in the graph G using Dijkstra's algorithm. Afterward, we update the cost-to-goal estimates of each state accordingly. This process is called "successor improvement" (SUI) and is applied after regression, before completion. Fig. 2.4 illustrates how SUI enhances cost-to-goal estimates in practice.

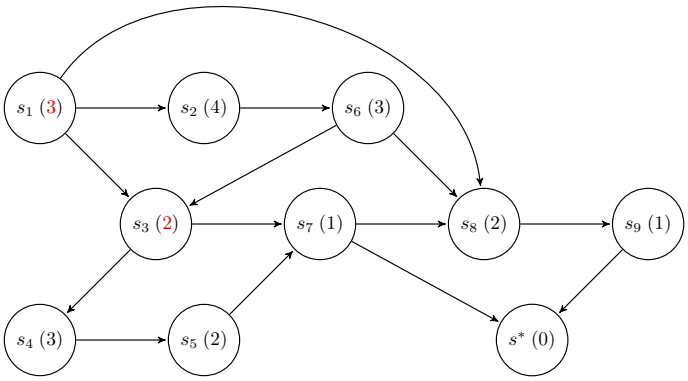
Figure 2.4 – Constructing the sample set graph G and updating the h-values of sampled states. The h-value of each state is in parenthesis.



(a) Collected samples from two regression rollouts (blue and orange).



(b) Constructed graph G with new successor-predecessor relationships.



(c) Updating the cost-to-goal estimates according to the newly discovered relationships.

3 PROPOSED APPROACH

Our objective is to discover preferred operators from a sample set and use them to train a NN for predicting preferred operators. In this chapter, we first introduce ideal preferred operators, then show the proposed approach for identifying preferred operators within an existing sample set. Finally, we present a sampling algorithm designed for discovering preferred operators.

3.1 Ideal Preferred Operators

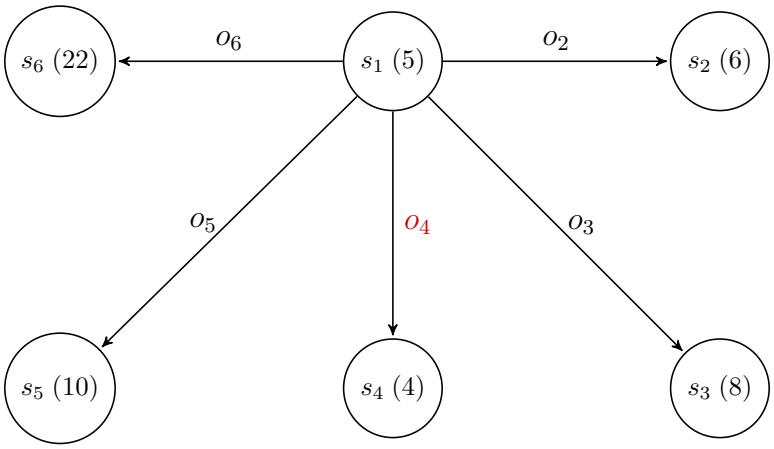
Ideally, we want preferred operators that help solve a task with the least effort, which we call ideal preferred operators.

Definition 12 (Ideal Preferred Operator)**.** *An ideal preferred operator generates a state with the shortest distance to the goal among all successors. Given a state s and an operator $o \in \mathcal{O}$ where $\mathrm{succ}(s, o) = t$, o is considered an ideal preferred operator if $h^*(t) = \min_{s' \in \mathrm{succ}(s)} h^*(s')$.*

Every state s that has a plan is associated with at least one preferred operator. These preferred operators generate successor states t where $h^*(t) < h^*(s)$. Thus, only goal states and dead-end states lack preferred operators.

Fig. 3.1 presents a visual example where o_4 is an ideal preferred operator of state s_1, since it leads to the successor state s_4 of minimum h-value among all successors of s_1.

Figure 3.1 – Example of an ideal preferred operator o_4 of state s_1. The h-value of each state is in parenthesis.



Property 1. *In a solvable planning task with unitary cost operators, if DQ-GBFS is guided by a blind heuristic function (equal value for all the expanded states but goal-aware and safe), tie-breaks by larger g-value, uses ideal preferred operators, only expands states from the preferred queue, and includes the initial state s_0 in the preferred queue, then the number of expansions made by DQ-GBFS will be equal to the number of states that are part of an optimal path.*

Proof. Base case: Initially, the only state in the preferred queue $\mathcal{Q}$ is the initial state s_0. DQ-GBFS expands states exclusively from the preferred queue. Since the task is solvable, we have $h^*(s_0) < \infty$, and because the blind heuristic is safe, then $h^{\text{blind}}(s_0) < \infty$. Thus, DQ-GBFS does not ignore s_0. If s_0 is a goal state, then DQ-GBFS returns it. If not, DQ-GBFS expands s_0, and this expansion satisfies the property since s_0 is part of any optimal path.

Inductive step: Assume that after k expansions, an optimal path exists such that only states from the optimal path were expanded. At the k-th expansion, DQ-GBFS expands a state s. Since the expansion occurred, s is not a goal state and had the largest g-value g_s in the queue $\mathcal{Q}$ before its removal. Given the blind heuristic, every state in $\mathcal{Q}$ has equal and finite h-values. Therefore, s was removed from $\mathcal{Q}$ because DQ-GBFS tie-breaks by larger g-value. We must prove that the $(k+1)$-th state removed from the queue $\mathcal{Q}$ is a successor of s and part of an optimal path.

By Definition 12, every ideal preferred operator generates a state with the shortest distance to the goal among all successors. Thus, when expanding state s at the k-th expansion step, DQ-GBFS will generate all successors of s, including *all* successors generated by ideal preferred operators. Because state s is part of an optimal path, s has at least one successor t generated by an ideal preferred operator that is also part of an optimal path, such that $h^*(t) = \min_{s' \in \text{succ}(s)} h^*(s')$, and $h^*(t) < \infty$. Since the blind heuristic is safe, $h^{\text{blind}}(t) < \infty$. Therefore, DQ-GBFS does not ignore t and inserts it in the queue $\mathcal{Q}$. The maximum g-value of a state in $\mathcal{Q}$ after the removal of s and before the insertion of its successors is equal to g_s. If there was a larger g-value than g_s, then state s would not be removed. Furthermore, all the successors of s have g-values of $g_s + 1$.

Consider the $(k+1)$-th removal from the queue $\mathcal{Q}$. If one of the successors of s is a goal state s^*, then $h(s^*) = 0$ because blind is goal-aware, so DQ-GBFS will remove it from the queue $\mathcal{Q}$ since a goal state has the minimum h-value among all states, and return the solution. Otherwise, since all the states in $\mathcal{Q}$ have equal

and finite h-values, and DQ-GBFS tie-breaks by larger g-values, DQ-GBFS will remove and expand one of the successors of state s generated by ideal preferred operators, with a distance to the goal of $h^*(s) - 1$, i.e., an expansion of a state generated by an ideal preferred operator makes progress on an optimal path.

Thus, since the $(k + 1)$-th step satisfies the property, we conclude that the number of expansions made by DQ-GBFS will be equal to the number of states that are part of an optimal path, i.e., the length of the optimal path equals the number of expansions of DQ-GBFS. □

3.2 Discovered Preferred Operators

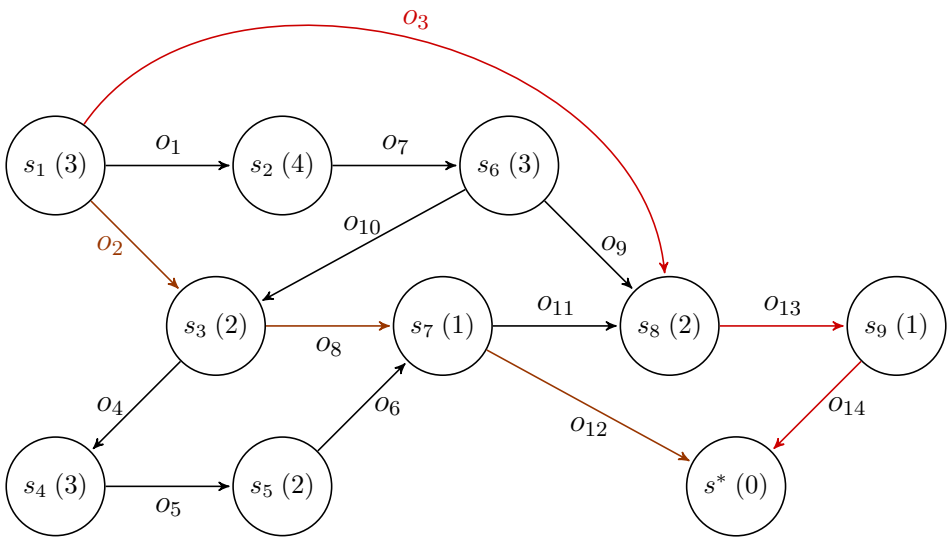
Obtaining ideal preferred operators is challenging for large tasks as we need access to the entire search state space, and computing the h^*-values for all states may be intractable. To address this, we compute the preferred operators within a sample set of the state space, without relying on h^*-values or logic-based methods such as computing relaxed planning graphs. We construct a sample set graph G (Definition 11 on page 34) by mapping operators that transition between two samples and then determine the operators contributing to a shortest path to the goal. In the graph G, each arc represents an applicable operator between two sampled states. Except for the goal condition, every sample has at least one successor, so it is feasible to trace multiple paths from a sampled state to a goal condition. We can identify the preferred operators for reaching a goal condition by selecting the operators associated with a shortest path.

Definition 13 (Discovered Preferred Operators)**.** *Let $G = (V, A)$ be a graph representing a set of samples, and let $G' = (V, A')$ be the shortest path directed acyclic graph generated using Dijkstra's algorithm. Dijkstra's algorithm is applied to G to find a shortest path from the goal condition to each vertex $v \in V$. For each vertex v, the arcs in A that belong to a shortest path to v are included in A'. The discovered preferred operators of the set of states s (since s is a partial state with undefined variables) are the operators represented by the outgoing arcs $a \in A'$ of the vertex labeled by s.*

The quality of the shortest paths relies on the accuracy of the h-values assigned to each sampled state, since they influence the construction of the shortest path graph G'. If the h-values are accurate (close to h^*), the algorithm is more

likely to prioritize the paths that are truly the shortest to reach the goal condition. However, if the h-values are inaccurate, the algorithm may be misled and prioritize suboptimal paths, leading to lower-quality (or missing) discovered preferred operators. Furthermore, a state can have multiple discovered preferred operators if it has multiple shortest paths. Fig. 3.2 has an example of a sample set graph G with two shortest paths from the partial state s_1 to the goal s^*. The preferred operators of s_1 are thus $\{o_2, o_3\}$. In the same example, suppose s_3 had an inaccurate h-value of 4 instead of 2. In this case, we would fail to "discover" operator o_2 as a preferred operator, since s_3 would have a greater h-value than s_1.

Figure 3.2 – Example sample set graph with two shortest paths (red, orange) from s_1 to the goal s^*. The h-value of each state is in parenthesis.



3.3 Generating Samples with XRS

The sample generation approach described in Section 2.8.1 is unsuitable for learning preferred operators due to duplicate samples within rollouts. This leads to a sample set that contains numerous repeated samples, which offer no additional value when constructing the shortest path graph G' for identifying preferred operators. Consequently, this approach fails to expand the state space and identify additional preferred operators effectively. To address this, we developed a new sample generation method designed to discover preferred operators, dubbed "expansion from random successors" (XRS), divided into two phases.

Let S_1 and S_2 denote the sets of samples generated in the first and second phases of XRS. The complete sample set is represented by $S = S_1 \cup S_2$, consisting of N distinct samples. Let k_1 and k_2 be variables satisfying $k_1 + k_2 = 1$. In the first phase of the algorithm, we use BFS by applying backward applicable operators from the goal condition until expanding $k_1 N$ states, which are then added to S_1. In the second phase, we maintain two structures first initialized during BFS: an open queue, containing states generated but not yet expanded, and a closed set, comprising states that have been expanded and already sampled. During each iteration, we randomly select a state from the open queue, move it to the closed set, add it to the sample set S_2, and insert its predecessors into the open queue if they are not already present in open or closed. The sampling process continues until $|S_2| = k_2 N$. See Algorithm 4. The primary distinction between phases one and two is the selection process for expanding states from the open queue. Unlike phase one, where the first-in, first-out approach of BFS is used, phase two involves randomly selecting a state from the open queue for expansion. We do not use a regression limit L as in FSM, but in practice our maximum regression depth remained close to d^* (Appendix A on page 59).

After regression, we apply SAI and SUI, extract the preferred operators, and complete the states as previously described. We compare XRS with FSM for discovering preferred operators in Section 4.4.

Algorithm 4 Sampling states using XRS

procedure XRS(s^*, N, k_1, k_2)
    $open = \emptyset$
    $S_1 = \emptyset$
    # Phase 1: populates S_1 and $open$ with BFS.
    $S_1, open \leftarrow$ BFS(s^*, $open$, $k_1 N$)
    $closed = S_1$
    # Phase 2: populates S_2 with random elements s from $open$.
    $S_2 = \emptyset$
    while $|S_2| < k_2 N$ **do**
        if $open = \emptyset$ **then**
            return $S_1 \cup S_2$ (backward state space fully expanded)
        end if
        $i \leftarrow$ random index of $open$
        # Initialize a partial state s.
        $s \leftarrow open[i]$
        Remove element at $open[i]$
        $closed \leftarrow closed \cup \{s\}$
        $S_2 \leftarrow S_2 \cup \{s\}$
        $P \leftarrow \{s' \mid s' \in pred(s)\}$
        for all partial states $s' \in P$ **do**
            if $s' \notin (open \cup closed)$ **then**
                Insert s' into $open$
            end if
        end for
    end while
    return $S_1 \cup S_2$
end procedure

4 EXPERIMENTS

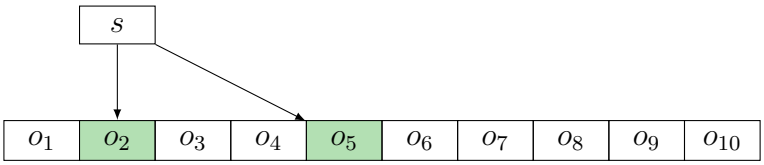
This section presents four experiments. In the first (Section 4.2), we investigate learning preferred operators and establish an upper-bound performance measure using the ideal preferred operators, comparing them to the proposed approach. The second experiment (Section 4.3) evaluates learning discovered preferred operators from sample sets of varying sizes and compares them to a logic-based preferred operator. The third experiment (Section 4.4) compares the results with preferred operators learned from a sample set generated using two distinct sampling methods, as described in Section 2.8.1 and Section 3.3. Finally, in the fourth experiment (Section 4.5), we analyze the learned preferred operators in conjunction with several logic-based heuristic functions.

4.1 Configuration

We use the same network architecture as Ferber et al. (2022), O'Toole et al. (2022), and Bettker et al. (2022) with modifications to support learning preferred operators. Specifically, we use a ResNet with He initialization (HE et al., 2015), consisting of two hidden layers followed by a residual block containing two hidden layers. Each hidden layer has 250 neurons and is ReLU-activated. The training uses the Adam optimizer (KINGMA; BA, 2015) with a learning rate of 10^{-4}, a batch size of 64, and a patience of 100 epochs. We use the MSE loss function to learn heuristic values in a regression context, and for learning preferred operators, we opt for the BCE loss since learning preferred operators is a multi-label classification problem. The input of the NNs consists of samples in the format $\langle \mathcal{F}(s), h(s) \rangle$ for the regression network, or $\langle \mathcal{F}(s), O_{pref} \subseteq O \rangle$ for the classification network, where $\mathcal{F}(s)$ is a Boolean representation of the input state s, with 0 if a proposition is false and 1 otherwise (Section 2.4), $h(s)$ is the target value representing the h-value for s, and O_{pref} are the target values in the representing the preferred operators for state s.

The output for the regression network is a single ReLU-activated neuron representing the learned h-value, and the output for classification is a sigmoid-activated tensor with values in the range $[0, 1]$ and a size equal to the number of operators of the input task. In the case of classification, each output neuron

Figure 4.1 – Example tensor with two preferred operators as target values.



corresponds to an indexed operator. For example, in a planning task with ten operators, in which a sampled state s has target preferred operators two and five, the indexes two and five of the output tensor must be maximized (Fig. 4.1). Finally, the training set has $90\,\%$ of the data, while the validation set contains the remaining $10\,\%$.

4.1.1 Extracting Preferred Operators

We extract the predicted preferred operators from the NN by selecting the operator with the highest output value. Additionally, if there are any operators with output values greater than 0.9, we include them in the selection. We do this to avoid false positives since most tasks considered in this work have a mean number of preferred operators close to one. Through experimentation, we found that this strategy produced better results than selecting only operators with output values above an arbitrary threshold, as lowering the threshold resulted in poorer performance.

4.1.2 Dataset

Our dataset consists of planning tasks with unitary operators and state spaces ranging from $40\,$K to $1\,$M states, identical to the dataset used by Bettker et al. (2022). We generate samples using state spaces from the following domains: Blocks World (Blocks), Grid, N-Puzzle, Rovers, Scanalyzer, Transport, and VisitAll, which are common in the IPC. We chose these state spaces because they have small descriptions and sizes, allowing us to access the h^*-values of every state. Table 4.1 has relevant information regarding each state space, and Appendix B on page 60 contains a small description of each domain. The output neurons in the classification NN correspond to the number of operators, whereas the regression

Table 4.1 – Information on the training tasks used for each domain. The forward state space size, the number of facts, and the number of operators.

Domain	FSP size	# facts	# operators
Blocks	65990	64	98
Grid	452353	76	252
N-Puzzle	181440	81	192
Rovers	565824	32	57
Scanalyzer	46080	42	300
Transport	637632	66	572
VisitAll	79931	31	48

NN used for learning heuristics produces a single numerical value as its output. The number of facts determines the input neurons.

We evaluate the results based on the number of expanded states since all planning tasks are solved. Each forward state space FSP has 50 randomly generated initial states, resulting in 50 planning tasks. In the tables, each cell represents the mean value across 50 planning tasks and 25 seeds, i.e., 5 sample seeds $\times\ 5$ network seeds. Each sample seed represents a different set of samples, while the network seeds are used to initialize the parameters of the NN. Additionally, when referring to a specific number of samples, we use the notation "n percent of the state space" to denote a fraction of the FSP, e.g., $5\,\%$ of Block World's FSP is $\lceil 65990 \times 0.05 \rceil = 3300$ samples.

4.1.3 Training

We implement sample generation in Neural Fast Downward (FERBER et al., 2020) and use PyTorch 1.9.0 (PASZKE et al., 2019) to define and train the NNs. We conducted the experiments on Ubuntu 20.04 LTS GNU/Linux machines with an AMD Ryzen 9 3900X 12-core processor (4.2 GHz), a memory limit of 4 GB, and one core per process. The NNs were trained until convergence, with the most complex networks requiring approximately two hours of training (Appendix E on page 63). We did not use GPUs due to our small training datasets.

4.1.4 Evaluation

We measure search performance by the number of expanded states since we have perfect coverage for all the selected domains. We use the Fast Downward planning system (HELMERT, 2006) with vanilla GBFS guided by a heuristic function to solve all 50 initial states of each domain using a 5 minute search time limit for each initial state. When preferred operators are used along the heuristic function, we use DQ-GBFS. Due to better experimental results, for the following experiments, we use a boost value of 1000, as in Richter and Helmert (2009). Results for searches with boosting disabled are available in Appendix C on page 61.

4.1.5 Sampling

We obtain the learned heuristic $\hat{h}$ by training over samples generated as described in Section 2.8.1, with the best configuration from Bettker et al. (2022), i.e., sample set size equal to $1\,\%$ of the state space, regression limit of $L = \lceil L_F/\overline{\text{eff}} \rceil$, and setting the quantity of samples generated by BFS to $p_{\text{FSM}} = 10\,\%$ of the total number of samples. As Bettker et al. (2022), $20\,\%$ of the total number of samples are randomly generated samples, with cost-to-goal estimates equal to the maximum estimate of the existing samples obtained through regression plus one.

Since in this work we are only interested in the effects of learned preferred operators, the learned heuristic functions $\hat{h}$ for each task have the fixed configuration described earlier. To obtain the learned discovered preferred operators $\hat{po}^{\text{G}}$, we train the NNs over sample sets generated according to the method described in Section 3.3, with $k_1 = 0.1$ and $k_2 = 0.9$, since these values had better results in preliminary experiments (Appendix D on page 62). In this configuration, we do not use randomly generated samples.

4.2 Learning Preferred Operators

This section establishes baselines for comparing our proposed approach of learned discovered operators. In particular, we compare task-solving performance with several configurations. We present results using (DQ-)GBFS guided

Table 4.2 – Expanded states of (DQ-)GBFS guided by h^*, $\hat{h}$, and $\hat{h}$ with ideal preferred operators and discovered preferred operators.

	h^*	$\hat{h}$				
Domain	-	-	po^*	$\hat{p}o^*$	$\hat{p}o^{G^*}$	$\hat{p}o^G$
Blocks	19.4	57.0	21.0	42.1	43.0	43.0
Grid	20.8	66.5	23.1	23.2	70.4	67.4
N-Puzzle	22.6	80.9	23.9	28.0	53.3	53.3
Rovers	10.3	13.4	10.7	10.6	12.2	12.2
Scanalyzer	9.2	28.3	10.6	10.7	29.1	30.7
Transport	13.3	25.2	13.9	13.9	21.3	21.4
VisitAll	11.9	21.8	12.8	12.8	21.2	20.5
Geo. mean	14.5	35.0	15.7	17.7	30.7	30.6

by the perfect heuristic h^* and the learned heuristic $\hat{h}$, with and without preferred operators. We also use two classes of preferred operators. The first class represents the ideal preferred operators po^* and $\hat{p}o^*$, where po^* is an oracle and $\hat{p}o^*$ was trained over the complete FSP of each task. The second class represents the proposed approach with learned discovered operators $\hat{p}o^{G^*}$ and $\hat{p}o^G$, which were trained on a $1\,\%$ sample set obtained through regression with the method described in Section 3.3, the only difference being that $\hat{p}o^{G^*}$ has perfect h^*-values for each sample.

Table 4.2 shows that all approaches using preferred operators lead to fewer expansions than using only the learned heuristic $\hat{h}$. In particular, $\hat{h}$ with the ideal preferred operators po^* and $\hat{p}o^*$ closely approach the optimal expansion values with h^*. The ideal preferred operators serve as performance limits, representing the best achievable results as defined in Section 3.1. When comparing the learned ideal preferred operators $\hat{p}o^*$ with the oracle po^*, we find that the NN can learn the preferred operators for the entire state space in all domains, except for Blocks World and N-Puzzle, where performance degrades but remains better than $\hat{h}$.

The learned ideal preferred operators $\hat{p}o^*$ surpass using only the learned heuristic $\hat{h}$ in all domains. They significantly reduce the number of expansions by over $50\,\%$ in Grid, N-Puzzle, and Scanalyzer. When we reduce the state space to $1\,\%$ and use the shortest path graph G' to identify preferred operators, the proposed approach $\hat{p}o^G$ exhibits an increase of approximately $73\,\%$ in the geometric mean of expanded states compared to the learned ideal preferred operators $\hat{p}o^*$. Still, $\hat{h}$ with $\hat{p}o^G$ expand fewer states than using only $\hat{h}$ in all domains except Grid

and Scanalyzer, with significant improvements in mean expansions of approximately $25\,\%$ and $34\,\%$ observed in Blocks World and N-Puzzle, respectively.

When comparing the discovered preferred operators $\hat{po}^{\mathrm{G}^*}$ to $\hat{po}^{\mathrm{G}}$, both have close results despite $\hat{po}^{\mathrm{G}^*}$ using the perfect heuristic h^* to compute the shortest paths in the sample set. The reason for this similarity is that the arcs in the shortest path graph G' for $\hat{po}^{\mathrm{G}}$ are largely retained when compared to $\hat{po}^{\mathrm{G}^*}$, and equivalent states in the sample sets of both methods tend to have similar preferred operators, indicating that they are likely to have similar shortest paths to follow. In particular, close to $100\,\%$ of the samples have the same set of preferred operators in $\hat{po}^{\mathrm{G}^*}$ and $\hat{po}^{\mathrm{G}}$ in all domains except Scanalyzer ($92\,\%$) and VisitAll ($68\,\%$). For all states with different preferred operators in all domains, the preferred operators of $\hat{po}^{\mathrm{G}}$ are a subset of the preferred operators of $\hat{po}^{\mathrm{G}^*}$. As seen in Table 4.2, this difference is negligible for Scanalyzer and VisitAll, where $\hat{po}^{\mathrm{G}^*}$ and $\hat{po}^{\mathrm{G}}$ differ by less than one expansion on average. This result indicates that discovering preferred operators from the shortest path graph G' with quality similar to ideal preferred operators is possible even without perfect h^*-values.

This experiment demonstrated the effectiveness of learning high-quality preferred operators that enhance suboptimal heuristics. The learned ideal preferred operators $\hat{po}^*$ outperformed $\hat{h}$ in all domains, improving the geometric mean of expansions by about $50\,\%$. The proposed approach $\hat{po}^{\mathrm{G}}$ had fewer expansions than $\hat{h}$ in five out of seven domains, improving the geometric mean by approximately $13\,\%$.

4.3 Discovering Preferred Operators on Different Sample Set Sizes

In this experiment, we use (DQ-)GBFS guided by the learned heuristic $\hat{h}$ and compare the learned discovered preferred operators $\hat{po}^{\mathrm{G}}$ to the logic-based preferred operators po^{FF} from FF (HOFFMANN; NEBEL, 2001) implemented in Fast Downward (HELMERT, 2006). In addition to using a sample set equivalent to $1\,\%$ of the forward state space, as in the previous experiment, we also examine other percentages: $5,\,\%$, $10\,\%$, $20\,\%$, $30\,\%$, $40,\,\%$, $50\,\%$. Table 4.3 shows the results.

By increasing the sample set size, the number of arcs and vertices in the shortest path graph G' expands, leading to the discovery of new shortest paths. As expected, the performance of the learned discovered preferred operators $\hat{po}^{\mathrm{G}}$

Table 4.3 – Expansions of (DQ-)GBFS guided by $\hat{h}$ without preferred operators, and with po^{FF} and $\hat{po}^{\text{G}}$ trained on the discovered learned operators, varying the size of the sample set according to different percentages of the state space size. Rovers maintained the same results from $20\,\%$ onward as they sampled the complete backward state space, which numerically represents approximately $15\,\%$ of the forward state space.

	$\hat{h}$								
			$\hat{po}^{\text{G}}$						
Domain	-	po^{FF}	$1\,\%$	$5\,\%$	$10\,\%$	$20\,\%$	$30\,\%$	$40\,\%$	$50\,\%$
Blocks	57.0	41.1	43.0	**26.2**	29.6	32.2	34.9	38.0	38.8
Grid	66.5	32.7	67.4	53.0	46.4	27.4	23.8	23.6	**22.9**
N-Puzzle	80.9	100.2	53.3	34.8	31.6	28.5	28.1	**27.4**	**27.4**
Rovers	13.4	18.5	12.2	**11.7**	13.0	17.2	17.2	17.2	17.2
Scanalyzer	28.3	17.1	30.7	18.1	13.0	11.7	**11.5**	11.6	**11.5**
Transport	25.2	17.0	21.4	16.3	15.8	15.1	14.7	14.4	**14.3**
VisitAll	21.8	18.7	20.5	17.1	**15.7**	**15.7**	15.8	16.1	16.3
Geo. mean	35.0	28.0	30.6	22.4	21.0	19.8	**19.5**	19.7	19.6

improves as the sample set size increases, reaching a plateau after $20\,\%$. However, Rovers shows an exception where the learned preferred operators result in increased expansions after $5\,\%$. In this domain, both the logic-based preferred operators po^{FF} and $\hat{po}^{\text{G}}$ trained on a $1\,\%$ sample set size yield worse results than $\hat{h}$ alone. This suggests that in Rovers, preferred operators may not be helpful since $\hat{h}$ already approaches optimality (13.4 vs. 10.3), as shown in Table 4.2.

As shown in Section 2.6, in Fast Downward, repeated generated states are not added to the preferred queue. This is more significant in Blocks World and Rovers, increasing the number of expansions. Furthermore, this happens more frequently when training over larger sample sets, as multiple preferred operators are predicted more frequently. To mitigate this issue, a potential approach may be disabling boosting or using a more restrictive threshold when extracting preferred operators.

Training with $5\,\%$ of the sample set size is sufficient to achieve $20\,\%$ fewer expansions on average compared to po^{FF}, i.e., the learned preferred operators $\hat{po}^{\text{G}}$ can outperform logic-based ones given a large enough sample set size. Specifically, with $5\,\%$, we achieve better results than po^{FF} except in Grid (53.0 vs. 32.7) and Scanalyzer (18.1 vs. 17.1). As the sample size increases, we eventually surpass po^{FF} in all domains. Notably, po^{FF} degrades performance in N-Puzzle compared to only using $\hat{h}$, whereas $\hat{po}^{\text{G}}$ with a $1\,\%$ sample set substantially improves

it.

Fig. 4.2 shows the standard deviations for the expansions in Table 4.3. Smaller sample sets generally exhibit higher variability in the number of expanded states. For instance, with a $1\,\%$ sample set size, Blocks World, Grid, N-Puzzle, and Scanalyzer have standard deviations of approximately 12, 27, 8, and 10, respectively. As the sample set size increases, the standard deviation decreases, except for Blocks World, which maintains a standard deviation of approximately 10. Notably, all domains, except Blocks World, have standard deviation values close to one for sample set sizes of $20\,\%$ and above. Bettker et al. (2022) also observed similar variability in their results.

4.4 Comparison to Alternative Sampling Method

This experiment compares the number of expansions between two methods for discovering operators. The first method is the sampling approach proposed in Section 3.3, while the second method is the one introduced by Bettker et al. (2022) and described in Section 2.8.1 (differently from learning $\hat{h}$, we exclude randomly generated samples since applicable operators do not necessarily generate them). Table 4.4 shows the results.

The learned preferred operators $\hat{po}^{\text{G}}$ consistently expands fewer states than $\hat{po}^{\text{FSM}}$, except for Blocks World (from $20\,\%$ onward) and VisitAll. Considering all the sample set sizes, $\hat{po}^{\text{G}}$ has about $14\,\%$ fewer expansions on average. Notably, $\hat{po}^{\text{FSM}}$ needs ten times the number of samples (e.g., $50\,\%$ vs. $5\,\%$) to achieve comparable results to $\hat{po}^{\text{G}}$ on average, highlighting the efficiency of XRS in contrast to FSM when it comes to learning preferred operators.

In Section 3.3 we mention FSM has many repeated states, which are not considered when constructing the shortest path graph. To provide quantitative insights, we examined a sample set size of $5\,\%$ consisting of samples with partial states (without completion). While $100\,\%$ of the states generated by XRS in all domains are unique, for FSM the percentages of unique states are as follows: $72\,\%$ in Blocks World, $84\,\%$ in Grid, $96\,\%$ in N-Puzzle, $58\,\%$ in Rovers, $89\,\%$ in Scanalyzer, $96\,\%$ in Transport, and $86\,\%$ in VisitAll.

Table 4.4 – Expansions of DQ-GBFS guided by $\hat{h}$ with $\hat{po}^{\mathrm{G}}$ and $\hat{po}^{\mathrm{FSM}}$ using FSM from Bettker et al. (2022), varying the size of the sample set according to different percentages of the forward state space size.

	1 %		5 %		10 %		20 %	
Domain	$\hat{po}^{\mathrm{G}}$	$\hat{po}^{\mathrm{FSM}}$	$\hat{po}^{\mathrm{G}}$	$\hat{po}^{\mathrm{FSM}}$	$\hat{po}^{\mathrm{G}}$	$\hat{po}^{\mathrm{FSM}}$	$\hat{po}^{\mathrm{G}}$	$\hat{po}^{\mathrm{FSM}}$
Blocks	**43.0**	56.8	**26.2**	31.7	**29.6**	30.4	**32.2**	32.8
Grid	**67.4**	74.3	**53.0**	67.8	**46.4**	67.3	**27.4**	69.9
N-Puzzle	**53.3**	67.6	**34.8**	48.5	**31.6**	38.1	**28.5**	32.3
Rovers	12.2	**12.1**	**11.7**	13.1	**13.0**	15.7	**17.2**	19.9
Scanalyzer	**30.7**	33.7	**18.1**	21.0	**13.0**	14.5	**11.7**	11.9
Transport	**21.4**	23.6	**16.3**	19.9	**15.8**	18.2	**15.1**	16.9
VisitAll	20.5	**19.8**	17.1	**15.6**	15.7	**15.3**	15.7	**14.8**
Geo. mean	**30.6**	34.2	**22.4**	26.4	**21.0**	24.3	**19.8**	23.8

	30 %		40 %		50 %	
	$\hat{po}^{\mathrm{G}}$	$\hat{po}^{\mathrm{FSM}}$	$\hat{po}^{\mathrm{G}}$	$\hat{po}^{\mathrm{FSM}}$	$\hat{po}^{\mathrm{G}}$	$\hat{po}^{\mathrm{FSM}}$
Blocks	34.9	**33.2**	38.0	**32.1**	38.8	**32.3**
Grid	**23.8**	64.5	**23.6**	59.1	**22.9**	55.0
N-Puzzle	**28.1**	29.6	**27.4**	28.9	**27.4**	27.7
Rovers	**17.2**	20.8	**17.2**	21.3	**17.2**	21.4
Scanalyzer	**11.5**	11.6	11.6	**11.5**	**11.5**	11.7
Transport	**14.7**	16.1	**14.4**	15.8	**14.3**	15.5
VisitAll	15.8	**14.9**	16.1	**15.6**	16.3	**15.4**
Geo. mean	**19.5**	23.2	**19.7**	22.9	**19.6**	22.5

4.5 Using Learned Preferred Operators with Other Heuristic Functions

We now examine the effects of using the learned preferred operators $\hat{po}^{\mathrm{G}}$, trained on a $1\,\%$ sample set size, on the performance of (DQ-)GBFS guided by different logic-based heuristic functions. Specifically, we use the more informed heuristics h^{FF} and h^{add}, the less informed heuristic h^{GC} (goal-count), and the blind heuristic h^{blind} without information. We also compare our results with po^{FF}.

The outcomes are summarized in Table 4.5. When using $\hat{po}^{\mathrm{G}}$ with the h^{GC} heuristic, we significantly reduce the number of expansions from 124.9 to 41.4. This performance is competitive with the baseline h^{FF} and h^{add} heuristics without preferred operators, yielding fewer expansions in Blocks World, N-Puzzle, and VisitAll. Additionally, when using h^{GC} with $\hat{po}^{\mathrm{G}}$ trained on a $5\,\%$ sample set size instead of $1\,\%$ as shown in the table, we achieve a geometric mean of 28.1 (vs. 41.4), which is about $25\,\%$ lower than the results obtained with the baseline h^{FF}

Table 4.5 – Expansions of (DQ-)GBFS guided by logic-based heuristics without preferred operators, and with preferred operators obtained by FF po^{FF} and the preferred operators $\hat{po}^{\text{G}}$ trained on a $1\,\%$ sample set size.

	h^{FF}			h^{add}		
Domain	-	po^{FF}	$\hat{po}^{\text{G}}$	-	po^{FF}	$\hat{po}^{\text{G}}$
Blocks	183.0	52.8	**46.6**	94.9	51.2	**39.4**
Grid	33.6	**30.0**	**30.0**	48.5	33.3	**30.5**
N-Puzzle	139.9	205.9	**59.1**	155.7	198.0	**69.3**
Rovers	11.5	**10.6**	**10.6**	11.4	19.0	**10.6**
Scanalyzer	28.5	**16.9**	29.3	21.6	**14.3**	23.4
Transport	17.8	**15.6**	19.9	17.9	**16.4**	20.0
VisitAll	27.3	23.8	**20.4**	30.4	29.4	**19.6**
Geo. mean	39.0	30.0	**27.0**	37.1	33.2	**26.0**
	h^{GC}			h^{blind}		
	-	po^{FF}	$\hat{po}^{\text{G}}$	-	po^{FF}	$\hat{po}^{\text{G}}$
Blocks	332.7	62.5	**60.5**	54K	10K	**306.8**
Grid	265.6	**60.4**	91.5	51K	11K	**152.0**
N-Puzzle	818.7	1.2K	**77.4**	67K	67K	**368.3**
Rovers	61.5	**17.5**	21.2	4K	832.1	**126.4**
Scanalyzer	31.9	**18.4**	28.9	5K	3K	**446.1**
Transport	200.5	44.1	**40.2**	145K	15K	**193.4**
VisitAll	16.7	**13.9**	19.7	2K	2K	**277.5**
Geo. mean	124.9	51.3	**41.4**	19.7K	6.7K	**244.3**

and h^{add} heuristics. These findings highlight that using preferred operators can lead to more significant performance improvements in task-solving compared to changing to a more informed heuristic, as previously noted by Corrêa et al. (2022).

Less informed heuristics diminish the effectiveness of po^{FF}. Notably, h^{blind} with $\hat{po}^{\text{G}}$ reduces the number of expansions by approximately $99\,\%$, whereas po^{FF} reduces the expansions by about $66\,\%$. In blind search, the heuristic function does not serve to guide the search process – it only assists in identifying a goal state. Therefore, the preferred operators act as a policy. The results indicate that po^{FF} fails to effectively serve as a guiding policy for the search, while the learned preferred operators $\hat{po}^{\text{G}}$ successfully fulfill this role.

Overall, these findings demonstrate the adaptability of $\hat{po}^{\text{G}}$ with different heuristics. We also show the standard deviation of expanded states for each domain in Fig. 4.3. Generally, using more informed heuristics with the learned preferred operators lead to reduced standard deviations.

Figure 4.2 – Mean number of expansions and its standard deviation per domain for DQ-GBFS guided by $\hat{h}$ with $\hat{po}^{\mathrm{G}}$ trained using sample sets of different sizes.

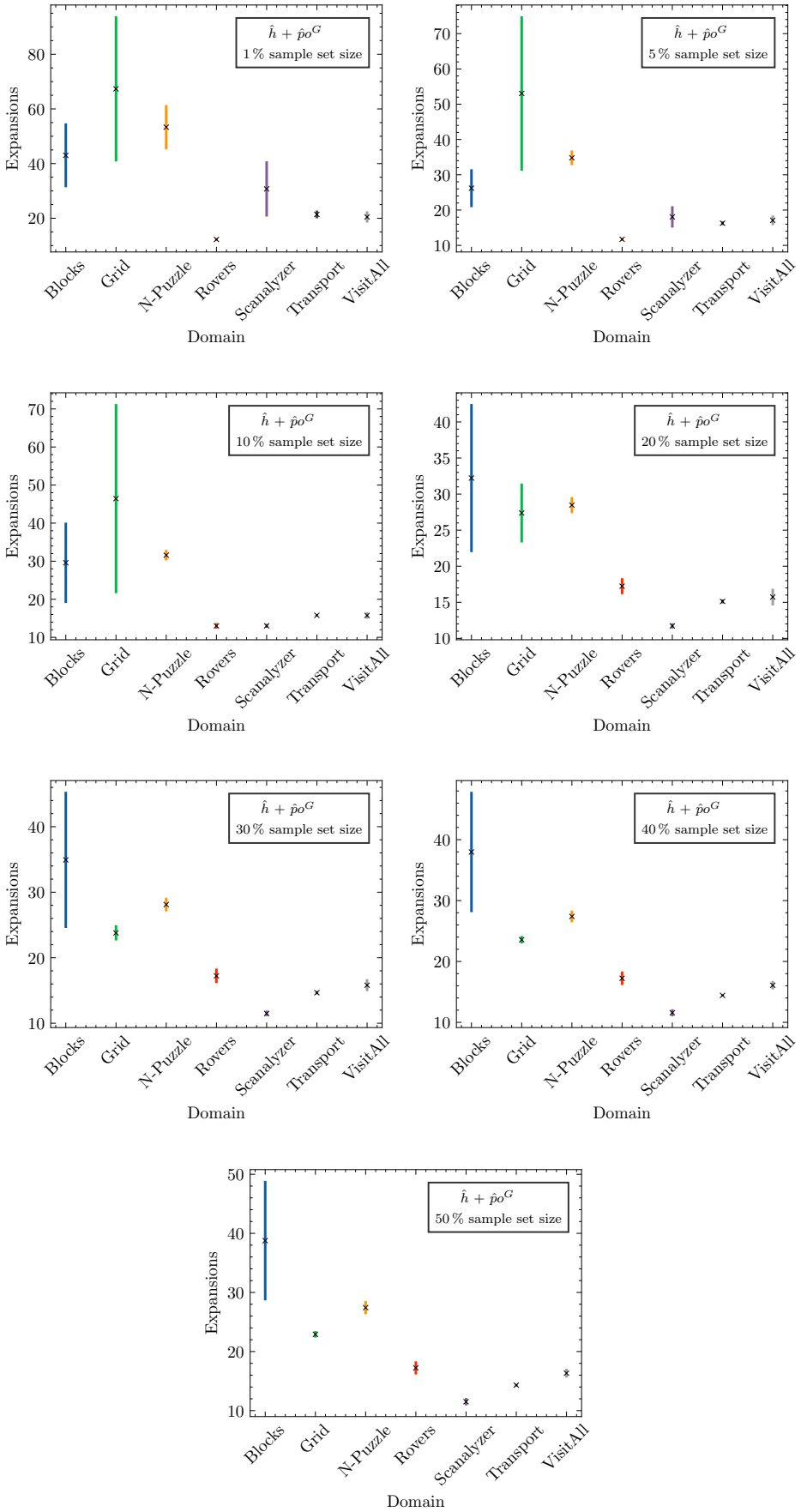
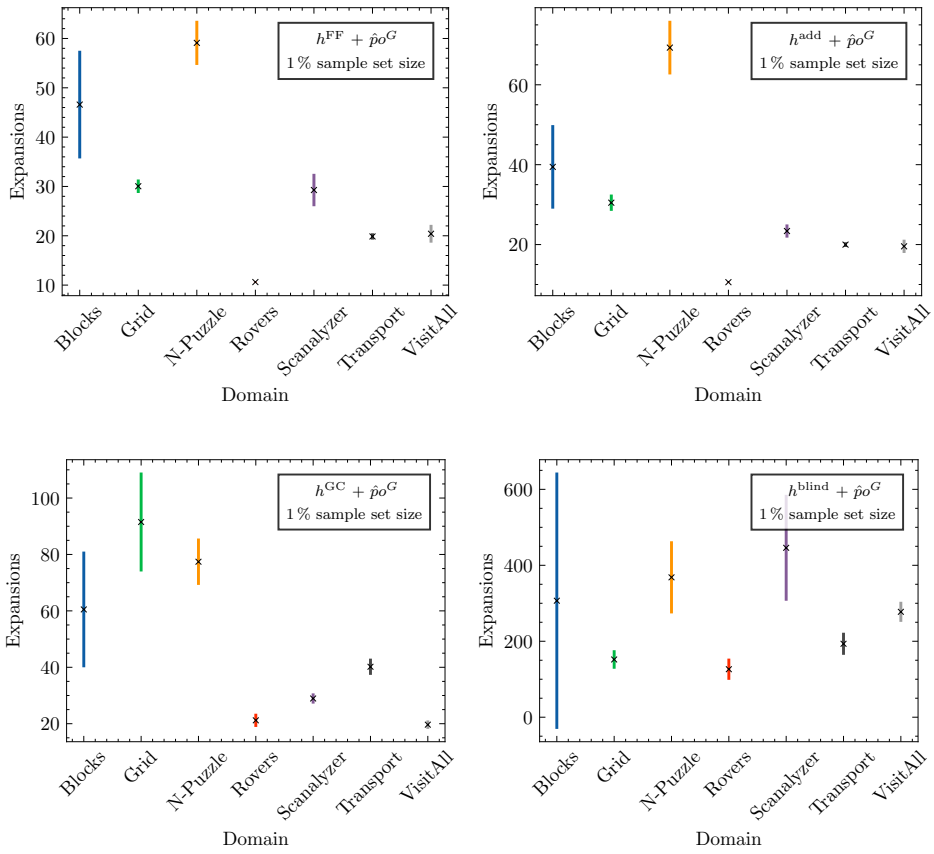


Figure 4.3 – Mean number of expansions and its standard deviation per domain for DQ-GBFS guided by different logic-based heuristics and $\hat{po}^{\mathrm{G}}$.



5 CONCLUSION

Researchers have extensively investigated learning sample-based heuristic functions for solving planning tasks. However, in this study, we explored the sampling and learning of preferred operators for the first time. Preferred operators are akin to policies, but they serve a distinct purpose as filters within the context of heuristic search, assisting the heuristic function rather than seeking a solution. Meanwhile, policies in reinforcement learning aim to find solutions based on rewards: the goal is to find the policy that maximizes the expected cumulative reward.

The findings suggest that learning-based approaches have a considerable potential to outperform logic-based methods. We show that learned preferred operators can surpass logic-based preferred operators like po^{FF} over the considered planning tasks. Furthermore, the learned preferred operators have fewer expansions than po^{FF} when paired with uninformed or less informed heuristics, although further investigation is needed to understand this phenomenon fully. The experiments highlight the ability to discover preferred operators and train an NN capable of generalizing for the entire task using a subset of the state space. Analysis between successor and predecessor states proved to be an effective method for discovering preferred operators from a limited sample set.

A limitation of learned preferred operators is the number of samples required for training. Our experiments used small state spaces and a relatively simple NN as a first step to explore learning preferred operators. Consequently, we had access to the complete state space of a task. However, this is infeasible for harder tasks, as state spaces tend to grow exponentially as the amount of information needed to describe them increases, meaning that if computational resources are a concern, even sampling $1\,\%$ of the state space can be impractical. Another problem involves the size of the output tensor, where certain domains have thousands of classes in larger tasks, significantly affecting training efficiency.

Future research can improve learning efficiency by investigating alternative learning architectures for generalizing preferred operators with smaller sample sets or using an alternative representation of preferred operators as the output tensor. Additionally, inspired by Lipovetzky et al. (2015), future research could expand our proposed approach to non-logic domains such as Atari games.

Overall, this research presents new opportunities for enhancing heuristic search in planning using learned preferred operators, highlighting the potential of learning-based methods requiring minimal domain logic.

REFERENCES

ALCAZAR, V.; BORRAJO, D.; FERNANDEZ, S.; FUENTETAJA, R. Revisiting Regression in Planning. In: ROSSI, F. (Ed.). Beijing, China: AAAI Press, 2013. p. 2254–2260.

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.

BETTKER, R. V.; MININI, P. P.; PEREIRA, A. G.; RITT, M. Understanding Sample Generation Strategies for Learning Heuristic Functions in Classical Planning. **arXiv preprint arXiv:2211.13316**, 2022.

BYLANDER, T. The Computational Complexity of Propositional STRIPS Planning. **Artificial Intelligence**, Elsevier, v. 69, n. 1-2, p. 165–204, 1994.

CORRÊA, A. B.; POMMERENING, F.; HELMERT, M.; FRANCES, G. The FF Heuristic for Lifted Classical Planning. In: **Proceedings of the AAAI Conference on Artificial Intelligence**. [S.l.]: AAAI Press, 2022. v. 36, n. 9, p. 9716–9723.

CULBERSON, J. C.; SCHAEFFER, J. Pattern Databases. **Computational Intelligence**, v. 14, n. 3, p. 318–334, 1998.

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

FERBER, P.; GEIßER, F.; TREVIZAN, F.; HELMERT, M.; HOFFMANN, J. Neural Network Heuristic Functions for Classical Planning: Bootstrapping and Comparison to Other Methods. In: **International Conference on Automated Planning and Scheduling**. [S.l.]: AAAI Press, 2022.

FERBER, P.; HELMERT, M.; HOFFMANN, J. Neural Network Heuristics for Classical Planning: A Study of Hyperparameter Space. In: **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.

GOODFELLOW, I. J.; BENGIO, Y.; COURVILLE, A. **Deep Learning**. Cambridge, MA, USA: MIT Press, 2016. <http://www.deeplearningbook.org>.

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.

HE, K.; ZHANG, X.; REN, S.; SUN, J. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. In: **International Conference on Computer Vision (ICCV)**. [S.l.]: IEEE, 2015. p. 1026–1034.

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

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

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

HELMERT, M.; DOMSHLAK, C. Landmarks, Critical Paths and Abstractions: What's the Difference Anyway? In: **Proceedings of the 19th International Conference on Automated Planning and Scheduling**. Thessaloniki, Greece: AAAI Press, 2009. p. 162–169.

HOCHREITER, S. Untersuchungen zu dynamischen neuronalen Netzen. **Diploma, Technische Universität München**, v. 91, n. 1, 1991.

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.

HOLTE, R. C. Common Misconceptions Concerning Heuristic Search. In: **Proceedings of the International Symposium on Combinatorial Search**. Atlanta, US: AAAI Press, 2010. v. 1, n. 1, p. 46–51.

KINGMA, D.; BA, J. Adam: A Method for Stochastic Optimization. In: **International Conference on Learning Representations**. [S.l.: s.n.], 2015.

LIPOVETZKY, N.; RAMIREZ, M.; GEFFNER, H. Classical planning with simulators: Results on the atari video games. In: **ICAPS Workshop on Heuristics and Search for Domain-independent Planning**. Jerusalem, Israel: ICAPS, 2015.

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

RICHTER, S.; HELMERT, M. Preferred Operators and Deferred Evaluation in Satisficing Planning. In: **International Conference on Automated Planning and Scheduling**. Thessaloniki, Greece: AAAI Press, 2009. v. 19, p. 273–280.

RICHTER, S.; WESTPHAL, M. The LAMA Planner: Guiding Cost-Based Anytime Planning with Landmarks. **Journal of Artificial Intelligence Research**, v. 39, p. 127–177, 2010.

RICHTER, S.; WESTPHAL, M.; HELMERT, M. LAMA 2008 and 2011. **International Planning Competition**, p. 117–124, 2011.

SEIPP, J.; RÖGER, G. Fast Downward Stone Soup 2018. **International Planning Competition**, p. 72–74, 2018.

SHEN, W.; TREVIZAN, F.; THIÉBAUX, S. Learning Domain-Independent Planning Heuristics with Hypergraph Networks. In: **Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)**. [S.l.]: AAAI Press, 2020. v. 30, 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 International Conference on Automated Planning and Scheduling (ICAPS)**. Singapore Management University, Singapore: AAAI Press, 2022. v. 32, p. 629–637.

TOYER, S.; THIÉBAUX, 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.; THIÉBAUX, S.; XIE, L. Action Schema Networks: Generalised Policies With Deep Learning. In: **Proceedings of the AAAI Conference on Artificial Intelligence**. New Orleans, US: AAAI Press, 2018. v. 32, n. 1.

TSOUMAKAS, G.; KATAKIS, I. Multi-Label Classification: An Overview. **International Journal of Data Warehousing and Mining (IJDWM)**, IGI Global, v. 3, n. 3, p. 1–13, 2007.

WICKLER, G.; TATE, A. **AI Planning Course**. [S.l.]: University of Edinburgh, 2013. <https://www.aiai.ed.ac.uk/project/plan/ooc/credits.html>.

YU, L.; KUROIWA, R.; FUKUNAGA, A. Learning Search-Space Specific Heuristics Using Neural Network. In: **ICAPS Workshop on Heuristics and Search for Domain-independent Planning**. [S.l.]: ICAPS, 2020.

APPENDIX A — MAXIMUM REGRESSION DEPTH WITH XRS

Table A.1 shows the maximum regression depth reached (h-value) found in samples generated with XRS using different sample set sizes, compared to the longest distance d^* between the goal condition and any potential initial state in the forward state space, and the regression limit L by Bettker et al. (2022).

Table A.1 – Maximum regression depth found in samples generated with XRS on different sample set sizes, compared to the longest distance d^* and the regression limit L.

			XRS						
Domain	d^*	L	$1\,\%$	$5\,\%$	$10\,\%$	$20\,\%$	$30\,\%$	$40\,\%$	$50\,\%$
Blocks	24	17	20	22	23	24	25	25	26
Grid	32	44	15	19	21	23	24	24	25
N-Puzzle	31	41	21	26	28	31	32	33	34
Rovers	19	27	15	18	19	20	20	20	20
Scanalyzer	15	20	10	14	15	16	17	16	17
Transport	17	35	12	16	17	19	20	20	20
VisitAll	15	17	11	14	16	17	17	18	18

APPENDIX B — TASKS OF EACH DOMAIN

The Blocks World domain involves manipulating blocks on a table to achieve a desired configuration through a sequence of actions. We used a planning task with 7 blocks.

In the Grid domain, an agent capable of holding a key traverses a grid, with certain cells being locked and requiring specific keys to open. We used a planning task with a 4×4 grid, four locked cells, and three keys with two possible shapes, circle or square.

The N-Puzzle domain consists of a square grid with numbered tiles. The objective is to rearrange the tiles from their initial scrambled state to a desired goal condition. We used a planning task with a 3×3 grid.

The Rovers domain involves rovers navigating a grid-based environment with missions like exploration or resource gathering. Each rover has specific capabilities and limitations, including movement range, sensing, and interaction abilities. We used a planning task with two rovers, each with one storage capacity and one camera, and four waypoints with different objectives.

The Scanalyzer domain models automated greenhouse logistics, using imaging facilities to collect plant data and conveyor belts to transport plants between smart greenhouses and imaging facilities. We use a planning task with six conveyor belts and six batches of plants.

The Transport domain consists of transporting packages from one location to another using trucks with specific capacities. We use a planning task with nine cities, two trucks, and four packages.

The VisitAll domain consists of a robot that needs to visit all the cells of a grid once. We use a planning task with a 4×4 grid.

APPENDIX C — DISCOVERED PREFERRED OPERATORS WITHOUT BOOSTING

Table C.1 shows the number of expanded states using the discovered preferred operators $\hat{po}^{\mathrm{G}}$ but no boosting during search, i.e., states are expanded alternately from the default and preferred queues.

Table C.1 – Expansions of DQ-GBFS guided by $\hat{h}$ with $\hat{po}^{\mathrm{G}}$ trained with the discovered learned operators varying the size of the sample set according to different percentages of the state space size. Boosting is disabled.

	$\hat{po}^{\mathrm{G}}$						
Domain	1 %	5 %	10 %	20 %	30 %	40 %	50 %
Blocks	43.6	32.0	33.3	34.6	34.6	36.1	36.5
Grid	63.7	55.6	47.5	35.0	31.9	31.5	30.9
N-Puzzle	57.5	44.1	42.1	40.5	39.8	38.9	39.0
Rovers	14.0	13.4	13.5	13.3	13.3	13.3	13.3
Scanalyzerunit	31.1	21.6	17.2	16.1	16.0	15.8	16.3
Transportunit	22.4	19.7	19.3	19.1	18.7	18.6	18.4
VisitAll	20.0	18.9	18.2	18.2	18.0	18.1	18.6
Geo. mean	31.6	26.2	24.6	23.2	22.7	22.7	22.9

APPENDIX D — XRS WITH DIFFERENT VALUES OF k

Table D.1 shows the number of expanded states of $\hat{po}^{\mathrm{G}}$ trained on sample sets with varying k_1, i.e., number of samples generated by BFS in the first phase of XRS. The values of k_1 used are 0.05, 0.1, 0.2, 0.3, 0.4, 0.5, and 0.6. Note that $k_2 = 1 - k_1$.

Table D.1 – Expansions of DQ-GBFS guided by $\hat{h}$ with $\hat{po}^{\mathrm{G}}$ on $1\,\%$ sample set size, with varying number of samples generated by BFS (k_1) in the first phase of XRS.

	$\hat{po}^{\mathrm{G}}$						
Domain	0.05	0.1	0.2	0.3	0.4	0.5	0.6
Blocks	38.9	43.0	**42.9**	45.2	48.9	54.5	54.6
Grid	72.0	**67.4**	77.0	77.0	79.1	79.0	81.6
N-Puzzle	57.2	53.3	**49.5**	53.8	51.8	51.2	52.6
Rovers	**12.0**	12.2	12.3	12.2	**12.0**	**12.0**	12.1
Scanalyzer	31.9	**30.7**	31.8	32.5	31.3	32.6	31.4
Transport	22.0	**21.4**	21.5	22.0	21.6	**21.4**	21.6
VisitAll	21.4	20.5	19.8	20.2	20.1	**19.3**	19.9
Geo. mean	31.2	**30.6**	31.0	31.8	31.8	32.2	32.5

APPENDIX E — TRAINING DETAILS

Table E.1 presents the mean training information for the regression networks used to learn the heuristic $\hat{h}$ across 25 different seed runs for each task, trained over $1\,\%$ sample set size. The table shows the epoch associated with the minimum validation loss, the value of the minimum validation loss, and the elapsed time required to train the NN. Table E.2 has the same information, but for learning $\hat{p}o^{\mathrm{G}}$ with classification networks using varied sample set sizes, and also showing the number of preferred operators per sample. All the networks early-stopped.

Table E.1 – Training summary to learn $\hat{h}$ over 25 seeds for each domain.

Domain	Best epoch	Val. loss	Elapsed time (s)
Blocks	325.2	1.0601	19.1
Grid	1169.1	6.8435	402.6
N-Puzzle	312.6	43.0032	52.5
Rovers	15.7	27.6334	44.2
Scanalyzer	115.2	12.2408	7.2
Transport	1634.1	11.0641	739.2
VisitAll	71.2	5.2799	8.8

Table E.2 – Training summary to learn $\hat{po}^{\mathrm{G}}$ over 25 seeds for each domain.

Domain	Sample set size (%)	Best epoch	Val. loss	Elapsed time (s)	# ops./sample
Blocks	1	167.9	0.0113	16.8	1.1
	5	75.0	0.0130	53.9	1.2
	10	52.9	0.0118	92.9	1.2
	20	42.2	0.0104	174.2	1.3
	30	38.6	0.0092	252.3	1.3
	40	39.0	0.0083	342.7	1.4
	50	35.4	0.0076	415.6	1.4
Grid	1	105.5	0.0022	99.2	1.2
	5	53.1	0.0018	378.4	1.3
	10	45.3	0.0016	730.3	1.3
	20	39.2	0.0014	1448.1	1.3
	30	35.8	0.0013	2079.7	1.3
	40	34.3	0.0012	2758.8	1.3
	50	34.5	0.0011	3449.9	1.3
N-Puzzle	1	115.0	0.0101	39.6	1.0
	5	57.9	0.0069	145.5	1.0
	10	48.6	0.0070	273.3	1.1
	20	47.9	0.0069	557.3	1.1
	30	44.8	0.0069	811.0	1.1
	40	42.8	0.0070	1093.8	1.1
	50	39.3	0.0070	1325.7	1.1
Rovers	1	30.6	0.0756	62.8	2.1
	5	24.7	0.0735	305.7	2.9
	10	22.8	0.0678	609.4	3.3
	20	27.8	0.0619	966.3	3.4
	30	27.4	0.0618	969.2	3.4
	50	28.2	0.0618	986.9	3.4
Scanalyzer	1	186.9	0.0244	15.2	1.8
	5	111.4	0.0158	54.8	2.5
	10	107.7	0.0133	105.6	2.9
	20	109.7	0.0131	213.7	3.3
	30	109.9	0.0130	323.0	3.6
	40	110.8	0.0132	441.9	3.8
	50	115.0	0.0131	554.3	4.1
Transport	1	80.6	0.0041	170.6	1.3
	5	45.7	0.0035	694.8	1.4
	10	40.3	0.0036	1326.4	1.4
	20	39.6	0.0035	2728.8	1.4
	30	39.4	0.0036	3974.5	1.5
	40	40.1	0.0036	5404.9	1.5
	50	41.0	0.0036	6433.7	1.5
VisitAll	1	196.8	0.0258	21.8	1.4
	5	105.7	0.0319	72.5	1.5
	10	84.9	0.0335	130.1	1.5
	20	75.2	0.0339	242.3	1.6
	30	69.5	0.0338	355.1	1.7
	40	73.3	0.0337	431.1	1.7
	50	74.0	0.0333	540.4	1.8

APPENDIX F — RESUMO EXPANDIDO

No contexto de tarefas de planejamento, os agentes precisam a selecionar a melhor ação possível dentre um conjunto consideravelmente vasto de opções disponíveis em cada etapa. Planejadores lógicos têm sido usados para lidar com esse problema, aplicando operadores preferidos que reduzem significativamente o fator de ramificação. Planejadores que incorporam operadores preferidos na busca foram os vencedores da trilha *satisficing* da *International Planning Competition* (IPC) em 2004 (HELMERT, 2006), 2008 (RICHTER; WESTPHAL, 2010), 2011 (RICHTER et al., 2011), and 2018 (SEIPP; RÖGER, 2018). No entanto, este trabalho apresenta um método que vai além dessas abordagens convencionais, introduzindo uma estratégia de amostragem e aprendizado de operadores preferidos com o objetivo de generalização em todo o espaço de estados de uma tarefa de planejamento.

O objetivo principal deste trabalho é identificar os operadores preferidos ideais, que se encontram nos caminhos mais curtos que levam a um objetivo específico. O desafio reside no fato de que os espaços de estado geralmente são extremamente grandes, o que dificulta a exploração completa de todas as possibilidades. Para contornar essa limitação, desenvolvemos uma nova abordagem de amostragem adaptada, projetada para extrair operadores preferidos de alta qualidade de um conjunto de amostras que representa uma fração do espaço de estados completo. Os resultados demonstram que essa abordagem reduzida ainda é capaz de alcançar excelentes desempenhos nas tarefas de planejamento consideradas.

Para fornecer uma análise mais abrangente dessa nova categoria de operadores preferidos, realizamos experimentos controlados em tarefas com espaços de estados pequenos, onde conseguimos estimativas perfeitas para o goal. Comparamos sistematicamente os resultados da abordagem proposta com os operadores preferidos do Fast-Forward (FF) (HOFFMANN; NEBEL, 2001; HELMERT, 2006), avaliamos a eficácia dos operadores preferidos aprendidos em diversos tamanhos de amostra e exploraramos seu desempenho ao serem combinados com diferentes funções heurísticas. Essa investigação detalhada permite uma compreensão mais profunda das vantagens e limitações dos operadores preferidos aprendidos.

Este estudo representa a primeira tentativa de descobrir operadores preferidos a partir de um conjunto de amostras e usar uma NN para aprendê-los. Apresentamos um novo método de amostragem e uma nova técnica para identificar operadores preferidos em amostras. A técnica de amostragem envolve regressão do estado objetivo, construindo um grafo com estados amostrados que representam suas relações sucessor-predecessor e determinando, para cada estado, o conjunto de operadores usados para atingir o estado objetivo no menor caminho como operadores preferidos. Este estudo revela que uma rede neural pode aprender os operadores preferidos a partir de um subconjunto do espaço de estados e estender esse aprendizado de forma eficaz no espaço de estados em diversos domínios de planejamento. A abordagem proposta supera o melhor método atual de operadores preferidos do FF nas tarefas de referência. Em particular, este trabalho apresenta:

- Um novo método de amostragem adaptado para descobrir operadores preferidos.
- Um novo método baseado em grafos de caminho mais curto para descobrir operadores preferidos em um conjunto de amostras existente.
- Uma análise da qualidade dos operadores preferidos aprendidos e uma comparação com os operadores preferidos do FF.

Um exemplo de resultado consiste em uma rede treinada sobre uma quantidade de amostras equivalente a $5\,\%$ do espaço de estados completo de uma tarefa. Nesse caso, os operadores preferidos aprendidos superam significativamente uma busca guiada apenas por uma heurística aprendida, com diminuição de $36\,\%$ em número de estados expandidos. Além disso, eles também superam os operadores do FF em média, com aproximadamente $20\,\%$ de expansões a menos.