Towards Learning Foundation Models for Heuristic Functions to Solve Pathfinding Problems

Anonymous submission

Abstract

Pathfinding problems are prevalent in robotics, computational science, and natural sciences. Traditional methods to solve these require training deep neural networks (DNNs) for each new problem domain, consuming substantial time and resources. This study introduces a novel foundation model leveraging deep reinforcement learning to train heuristic functions that seamlessly adapt to new domains without further fine-tuning. Building upon DeepCubeA, we enhance the model by incorporating domain-specific state transition information into the heuristic function, improving adaptability. Using a puzzle generator for the 15-puzzle action space variation domains, we demonstrate our model's ability to generalize and solve unseen domains. We achieve strong correlations between learned and ground truth heuristic values across various domains, as evidenced by robust R-squared and Concordance Correlation Coefficient metrics. Additionally, we train a foundation model on domains generated using PDDLFuse, achieving better generalizability across various n-puzzle action variation domains compared to existing foundation models trained with supervised learning. These results underscore the potential of reinforcement learning-based foundation models in planning, establishing new benchmarks for domain generalizability in AI-driven solutions for complex pathfinding problems.

Introduction

Pathfinding aims to find a sequence of actions that forms a path from a given start state to a given goal state while minimizing the path's cost. Pathfinding problems are prevalent across computer science, robotics, mathematics, and the natural sciences. Heuristic search, one of the most prominent approaches to solving pathfinding problems, relies on a heuristic function. This function estimates the cost of the shortest path from a given state to the nearest goal state, commonly referred to as the "cost-to-go." Recently, deep reinforcement learning (DRL) (Sutton and Barto 2018) has emerged as a promising method for the automatic construction of domain-specific heuristic functions in a largely domain-independent fashion (Agostinelli et al. 2019; Agostinelli, Panta, and Khandelwal 2024). However, training such heuristic functions using deep neural networks (DNNs) (Schmidhuber 2015) can take days and require substantial computational resources, often necessitating retraining for even minor domain changes. These hardware and time constraints limit accessibility and scalability for broader research and applications.

Challenges similar to those in pathfinding have been encountered in computer vision and NLP, where foundation models have addressed generalization issues by being pre-trained on large, diverse datasets and adapting to new tasks with minimal fine-tuning. Creating a foundation model for heuristic functions could similarly transform heuristic search by enabling effective generalization across states and pathfinding domains. However, existing foundation models are limited by narrow set of domains they are trained on. Generating diverse domains allows for better generalization, as shown in reinforcement learning studies (Mehta et al. 2020; Ajani, Hur, and Mallipeddi 2023).

In this study, we propose a series of foundation models to address generalizability in pathfinding. The first model generalizes across 15-puzzle action space variations for a fixed puzzle size by incorporating action space information into state representations. We extend this with a more generalizable model that scales to any n-puzzle size without retraining, leveraging a single graph SLG representation (Chen, Thiébaux, and Trevizan 2024). Lastly, we train a foundation model on diverse domains generated with PDDLFuse (Khandelwal, Sheth, and Agostinelli 2024), capable of generalizing to n-puzzle variations and other domains without domain-specific training. These reinforcement learning-based models demonstrate superior performance compared to supervised learning-based foundation models, with strong correlations to ground truth metrics such as R-squared and Concordance Correlation Coefficient (CCC), underscoring their ability to generalize across unseen domains.

- We introduce a novel approach integrating state transition information with state representations to enhance generalizability of heuristics in pathfinding problems.
- We show that training on diverse generated domains with reinforcement learning improves generalizability, enabling models to handle 15-puzzle action variation domains and achieve competitive results against domain-specific and foundation models.

Background and Literature Review

This section provides an overview of the key concepts and existing research relevant to our study.

Foundation Model

Foundation models are pre-trained deep learning models using supervised or self-supervised techniques on extensive, diverse datasets. These models are adaptable to various downstream tasks.

Definition: A foundation model F_θ is trained to minimize the loss function L over a dataset D, with diverse inputs:

$$\theta^* = \arg\min_\theta L(F_\theta(x), x) \quad \text{for} \quad x \in D$$

Once trained, F_θ can generalize across different domains and tasks with or without fine-tuning. Foundation models have significant applications in natural language processing (NLP), computer vision, and healthcare.

Pathfinding Problems

Pathfinding domains are typically represented by directed weighted graphs where nodes symbolize states and edges denote transitions between states, with weights representing the costs of these transitions. In this context, a pathfinding problem is defined by a domain along with specified start and goal states. The enormity of state spaces in many such problems precludes the full representation of the graph, necessitating more abstract representations. We define an action space, $\mathcal{A}$, which delineates permissible actions for transitioning between states, a state transition function, T, that maps a state and an action to the resulting state, and a cost function, c, which assigns costs to transitions.

Traditionally, pathfinding employs heuristic search strategies such as A* search (Hart, Nilsson, and Raphael 1968), which uses a heuristic function h to prioritize node expansion in a search tree by combining the cost of a path with an estimated cost to the goal. The process continues until a node corresponding to a goal state is expanded. In the classical approach, the heuristic values, $h(s)$, are stored in a lookup table, each entry corresponding to a state s. The tabular value iteration, central to this approach, updates the cost-to-go for each state in the following form:

$$V'(s) = \min_{a \in \mathcal{A}} (c(s, a) + V(T(s, a))) \tag{1}$$

Here, V is a table mapping states to their cost-to-go estimates, and V' represents the updated estimates.

Approximate Value Iteration Due to the impracticality of tabular methods for domains with large state spaces, such as those involving complex combinatorial problems, we employ approximate value iteration (AVI). AVI uses a parameterized function to approximate the value iteration updates. In AVI, the heuristic function h is updated iteratively using a deep neural network (DNN) with parameters θ, approximating the value function. The heuristic update rule is given by:

$$h'(s) = \min_{a \in \mathcal{A}} (c(s, a) + h(T(s, a))) \tag{2}$$

where $c(s, a)$ is the cost of taking action a in state s, and $T(s, a)$ is the state reached by taking action a in state s. The network refines its estimates for the cost-to-go values by minimizing the loss:

$$L(\theta) = \left(\min_{a \in \mathcal{A}} (c(s, a) + h_{\theta^-}(T(s, a))) - h_\theta(s) \right)^2 \tag{3}$$

where $h_\theta(s)$ approximates the cost-to-go and θ^- denotes the parameters of a target network, periodically synchronized to θ to stabilize the training process against a non-stationary target.

For this study, we work with the 15-puzzle, or sliding tile puzzle (Keith 2011). The goal is to rearrange the numbered tiles from an initial state to the goal state (shown in Figure 4), using canonical moves (up (U), down (D), left (L), right (R)) and diagonal moves (upper-left (UL), upper-right (UR), down-left (DL), down-right (DR)). More details are in the Supplementary material.

Review of DeepCubeA

DeepCubeA (Agostinelli et al. 2019) applies deep reinforcement learning integrated with approximate value iteration (AVI) (Equation 2) and a weighted A* search to solve puzzles like Rubik's Cube, N-Puzzle, Sokoban, and Lights-out. DeepCubeA learns a domain-specific heuristic function in a largely domain-independent way without human guidance.

However, despite its effectiveness, DeepCubeA faces significant challenges, such as lengthy training times and the necessity to retrain the model from scratch for a slight change in the domain or its transition dynamics, increasing computational demands for larger puzzles. These challenges underscore the need for more adaptable and generalized solutions in AI pathfinding systems that can leverage learned heuristic functions across various domains without retraining from scratch.

Batch Weighted A* Search

A* search is a well-known pathfinding and graph traversal algorithm, which offers a robust method for finding the shortest path between nodes in a graph. A* search operates by maintaining a priority queue of paths, where the priority is determined by the sum of the path cost from the start node to the current node and an estimated cost from the current node to the goal. Here, each node represents a state in the state space. This estimated cost is provided by a heuristic function $h(x)$, which ideally never overestimates the actual minimum cost to reach the goal from node x. The fundamental operation of A* search can be described by the equation:

$$f(x) = g(x) + h(x) \tag{4}$$

where $g(x)$ represents the actual cost to reach node x from the start node, and $h(x)$ is the heuristic estimate of the cost to reach the goal from node x.

Batch Weighted A* Search Despite its efficiency, A* can be computationally demanding, particularly in terms of memory usage, when applied to problems with vast state spaces or complex transition dynamics. Batch Weighted A* Search (BWAS) addresses this issue by introducing two modifications to the traditional A* algorithm: a weighting factor λ for the path cost, and batch processing of node expansions (Agostinelli et al. 2019). The cost function in BWAS is modified as follows:

$$f(x) = \lambda g(x) + h(x) \tag{5}$$

where λ is a coefficient that can be adjusted between zero (emphasizing heuristic guidance) and one (emphasizing path

cost). Lowering λ typically results in faster execution but may increase the path length, trading off optimality for computational efficiency.

Moreover, BWAS enhances computational performance by utilizing batch processing techniques. Instead of expanding nodes one at a time, BWAS expands a batch of nodes N with the lowest f-values simultaneously. This approach is particularly effective when implemented on parallel computing architectures, such as GPUs, where the heuristic function $h(x)$, often computationally intensive, can be calculated for multiple nodes concurrently.

BWAS has been successfully applied in complex problem domains, such as solving combinatorial puzzles like the Rubik's cube and other large-scale pathfinding tasks. By adjusting λ and N, practitioners can fine-tune the balance between exploration and exploitation, optimizing the algorithm's performance to suit specific problem characteristics or resource constraints.

Generalization in Pathfinding Problems

(Chen, Thiébaux, and Trevizan 2024) introduced three novel graph representations for planning tasks using Graph Neural Networks (GNNs) to learn domain-independent heuristics. Their approach mitigates issues with large grounded GNNs by leveraging lifted representations and demonstrates superior generalization to larger problems compared to models like STRIPS-HGN. However, it faces scalability issues with large graph construction. Similarly, (Chen, Thiébaux, and Trevizan 2023) proposed the GOOSE framework, using GNNs with novel grounded and lifted graph representations for classical planning. Their heuristics outperform STRIPS-HGN and hFF in various domains but require extensive training data and struggle with very large graphs. Additionally, (Toyer et al. 2018) utilized GNNs to improve coverage and plan quality in classical planning tasks through new graph representations, though their approach only generalizes across a subset of test domains. However, these approaches rely on supervised learning, which assumes the ability to solve moderately difficult problem instances with existing solvers, which may not always be the case for real-world problems.

Large language models (LLMs), pre-trained on extensive textual datasets, have shown potential in downstream natural language processing tasks. While attempts to use LLMs for pathfinding via in-context learning have shown modest performance (Sermanet et al. 2023; Li et al. 2023; Silver et al. 2023), they face notable challenges and lack inherent search capabilities. More details are provided in the Supplementary material.

The Fast Downward planner (Helmert 2006) uses domain-independent heuristics like the Fast Forward (FF) heuristic (Hoffmann and Nebel 2001) to solve pathfinding problems efficiently. These heuristics generalize across various domains but do not perform as well as learned heuristics (Agostinelli, Panta, and Khandelwal 2024). More details are provided in the Supplementary material.

Existing Foundation Models

We utilize the model introduced by (Chen, Thiébaux, and Trevizan 2024) as a baseline for evaluating our approach. This work addresses the challenge of generalization in planning by employing Graph Neural Networks (GNNs) trained in a supervised manner on a novel representation. Specifically, the model leverages the STRIPS Learning Graph (SLG) representation, a grounded graph structure that efficiently encodes planning tasks. SLG represents planning tasks with nodes corresponding to actions and propositions, while edges capture preconditions and add and delete effects. Node features indicate whether a proposition is part of the initial state or goal, resulting in a compact representation that retains critical planning semantics.

Domain Randomization and Generalization in Reinforcement Learning

Domain randomization is a key technique in reinforcement learning (RL) that improves robustness and generalizability by exposing agents to diverse training domains, thereby improving generalization to unfamiliar domains. Mehta et al. (2020) introduced Active Domain Randomization (ADR), which strategically manipulates challenging environmental parameters to improve policy robustness, particularly in robotic control. Similarly, Ajani, Hur, and Mallipeddi (2023) showed that varying physical properties like surface friction can significantly boost an RL agent's generalization ability. Kang, Chang, and Choi (2024) further refined this with Balanced Domain Randomization (BDR), which focuses training on rare and complex domains to enhance performance under demanding conditions. In non-physical tasks, Koo, Yu, and Lee (2019) applied adversarial domain adaptation to align feature representations across domains, enhancing policy generalization in complex tasks like dialogue systems. These studies demonstrate the power of domain randomization in building resilient AI, aligning with PDDLFuse's goal to generate diverse planning domains to improve generalization in automated planning.

PDDLFuse

PDDLFuse is a domain generation tool designed to create novel and diverse planning domains by fusing existing ones (Khandelwal, Sheth, and Agostinelli 2024). Unlike traditional methods that primarily reconstruct domains from natural language descriptions, PDDLFuse employs domain randomization to generate new domains. By systematically manipulating predicates, preconditions, and effects using probabilistic parameters, PDDLFuse produces highly varied planning problems. This process includes integrating object counts, predicate reversibility, and negations, resulting in challenging domains for evaluating foundational planning models. Generating complex, out-of-distribution domains enables training across diverse scenarios, improving model generalization.

Theoretical Framework

Pathfinding problems are represented as weighted directed graphs, but encoding the entire graph is impractical due to its

large size. To generalize across a domain, we need to understand several key components: the state space, the transition function, and the transition cost function.

This study focuses on a subset of this broader problem to demonstrate that deep reinforcement learning can achieve this. Specifically, we keep the state representation consistent, assume prior knowledge of the action space, and use a uniform transition cost function. Consequently, the state transition function is implicitly defined by the action space. For the 15-puzzle, which involves actions such as {(U), (D), (L), (R), (UL), (UR), (DL), (DR)}, we generate domain variations by randomly selecting the set of actions available for each cell within the puzzle. We concatenate the domain's action space information with the state representation and input is fed into the heuristic function $h(s, va)$, where va represents the action space information.

This additional context allows heuristic functions to more accurately predict cost-to-go values, even in previously unseen domains. Thus, we can present a new domain to our deep neural network with the action space information, facilitating the development of heuristic functions that are effective within a single domain and adaptable across various domains. This approach advances the field of pathfinding in AI, showcasing the potential for domain generalizability through Deep RL.

Methodology

This section details the techniques used to generate dynamic puzzle domains and an updated approach to approximate value iteration. We also outline the evaluation metrics employed to assess the correlation between learned and ground-truth heuristic values.

Action Variation N-Puzzle Environment Generator

This section outlines Algorithm 1 (provided in the Supplementary Material), which generates dynamic puzzle domains by initializing randomized actions for each cell while ensuring the availability of reversible actions. The environment generator assigns a random set of actions to each cell in the puzzle grid and validates their reversibility to ensure bidirectional navigation.

Reversible moves are essential as they ensure that every action in the environment has a counteraction to revert the state, confirming the puzzle's solvability from any configuration. Reversible moves are also used to obtain ground-truth heuristic values, as unsolvable states (with infinite costs) would complicate heuristic value determination. However, the updated AVI based model can also solve domains without reversible moves; this approach is chosen for convenience in this study.

For the 8-puzzle, 8 actions across 9 cell positions yield 72 possible mappings for defining available actions. Reversible moves occupy 18 mapping slots (9 cells $\times$ 2 actions), leaving $72 - 18 = 54$ slots, resulting in 2^{54} possible domain variations. Similarly, the 15-puzzle with 8 actions and 16 cell positions yields 2^{96} configurations, and the 24-puzzle with 8 actions and 25 cell positions gives 2^{150} configurations. This method ensures a comprehensive set of domains for effective model training and evaluation.

Updated Approximate Value Iteration

In the updated approach to approximate value iteration, we enhance the state representation by concatenating it with a one-hot encoding of the action space. This combined representation provides additional context to the heuristic function about the available actions at each state, leading to more accurate predictions. The updated cost-to-go function now incorporates this augmented representation, as shown in the equation below:

$$h'(s, va) = \min_{a} \left(c(s, T(s, a)) + h(T(s, a), va)\right) \quad (6)$$

Here, $h'(s, va)$ is the updated heuristic function, $c(s, T(s, a))$ is the transition cost, $T(s, a)$ represents the state transition function, and va denotes the action variation context. This modification allows the heuristic function to leverage both state and action information, thereby improving the generalization and accuracy of cost-to-go estimations.

Other Foundation Models

To extend our analysis, we train other foundation models. These include the GOOSE foundation model, a supervised learning-based domain-independent approach (Chen, Thiébaux, and Trevizan 2024), and two reinforcement learning-based models trained using PDDLFuse. The first PDDLFuse model is trained on diverse domains derived from n-puzzle action variation domains (these variations are based on the environment generator), while the second is trained on a mix of n-puzzle and other domains.

Evaluation Metrics

The correlation between the learned heuristic value and the ground truth heuristic value is evaluated using two statistical measures: **1. Concordance Correlation Coefficient (CCC)**: CCC measures the agreement between two variables (learned and true heuristic values) by considering both precision and accuracy, capturing how closely the predictions match the ground truth. **2. Coefficient of Determination (R-squared, R^2)**: R^2 quantifies the proportion of variance in the ground truth heuristic values that is predictable from the learned heuristic values, providing insight into the goodness of fit of the model. In addition to statistical metrics, planning-specific metrics such as **average path length**, **optimality percentage**, **nodes expanded**, **time taken**, **nodes expanded per second**, and **percentage of problems solved** are used to evaluate solution quality, search efficiency, computational performance, and model robustness. These metrics provide a holistic view of the model's practical applicability in planning tasks. More details are in the Supplementary material.

Experimental Setup and Results

Our experimental setup evaluates the performance of foundation models on n-puzzle problems across 8-puzzle, 15-puzzle, and 24-puzzle domains. The models are trained using various inputs and paradigms, with detailed experimental configurations provided in the Supplementary Material.

Trained Foundation Models

We trained several foundation models using different methodologies and architectures:

UAVI Model: The UAVI (Updated Approximate Value Iteration) model is trained on domains generated by the action variation n-puzzle environment generator. It employs a Residual Network (ResNet) (He et al. 2016) implemented in PyTorch (Paszke et al. 2019). The architecture includes an input layer for state or combined state-action space, initial processing layers that expand the input to a hidden space of size 5000 with ReLU activation, and four residual blocks, each operating in a 1000-dimensional hidden space, to facilitate deep learning without performance degradation. The output layer then transforms the processed data into heuristic estimates.

GOOSE Model: The GOOSE model is a supervised learning-based foundation model trained on 50,000 data points from existing IPC domains (Chen, Thiébaux, and Trevizan 2024). It uses an SLG graph representation and a Message Passing Neural Network (MPNN) with 16 message-passing layers, mean aggregation, a hidden dimension of 64. Training optimizes Mean Squared Error (MSE) loss with the Adam optimizer and an initial learning rate of 0.001.

PDDLFuse-Based Models: Two models were trained on domains generated using PDDLFuse (Khandelwal, Sheth, and Agostinelli 2024), leveraging the SLG representation and MPNN architecture similar to GOOSE Model.

- **Action-GNN (AGN):** Trained exclusively on PDDLFuse-generated domains derived from action variations in n-puzzle domains (N ranging from 8 to 24). This model focuses on structural variability in actions and their effects, training for 850,000 iterations with a batch size of 100 using approximate value iteration.
- **Fuse-GNN (FGN):** Trained on domains generated by PDDLFuse, covering diverse base domains such as Blocksworld, Sokoban, n-puzzle, and other IPC domains. This model emphasizes cross-domain generalization and is trained for 1,000,000 iterations with a batch size of 100 using approximate value iteration.

Baseline Models

To assess the impact of training configurations, we also trained the original DeepCubeA model on three n-puzzle domains: canonical actions (C: U, D, L, R), diagonal actions (D: UL, UR, DL, DR), and all actions combined (C+D) for 8-puzzle, 15-puzzle, and 24-puzzle. DeepCubeA, using the original deep value iteration method (Equation 2), observes 10 billion examples during training due to its constant action space, enabling faster example generation.

In contrast, the UAVI model generates approximately 1 billion examples, with each state derived from a new action space variation domain, resulting in slower example generation. This model is trained using approximate value iteration with state transition information (Equation 6) for 8-puzzle, 15-puzzle, and 24-puzzle. Additionally, another baseline model was trained on examples from new action space variation domains using approximate value iteration without state transition information (Equation 2) for the 8-puzzle and 15-puzzle.

Test Data Generation

To evaluate the performance of UAVI, we generated three distinct test datasets:

- **Data1:** 500 states per domain (C, D, and C+D) generated via random walks ranging from 1,000 to 10,000 steps from the goal state.
- **Data2:** 100 states for each of 500 unique action space variation domains, generated via random walks of up to 500 steps from the goal state.
- **Data3:** 1,500 states per domain (C, D, and C+D) generated via random walks of 0 to 500 steps from the goal state, specifically designed to evaluate heuristic accuracy against optimal ground truth values.

Ground truth heuristic values for all test data were computed using the Fast Downward planner with the merge-and-shrink heuristic (Sievers 2018). This approach simplifies the state space by merging and shrinking states, yielding optimal heuristics. However, it is computationally demanding. For example, obtaining ground truth values for **Data3** required running 15 parallel instances of 100 problems each, with a 144-hour cutoff per instance, taking over three weeks to complete.

Comparison Against Ground Truth

We evaluated UAVI's heuristic accuracy by comparing its predictions against ground truth values for the 15-puzzle domain.

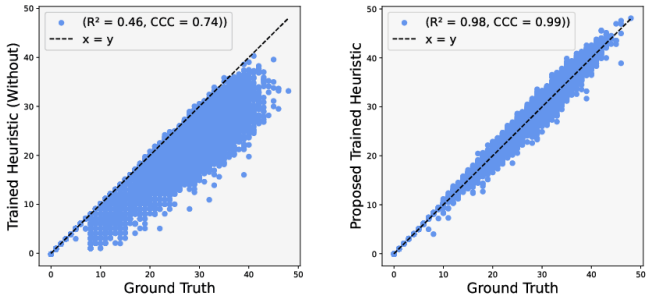
Plotting Ground Truth Heuristic Values. Figure 1 illustrates the results for UAVI with action information and the baseline model without action information. UAVI achieved a CCC of 0.99 and R^2 of 0.98, significantly outperforming the model without action information (CCC of 0.74 and R^2 of 0.46).

For the 15-puzzle domain, UAVI achieved:

- $R^2 = 0.96$ and CCC = 0.98 for C.
- $R^2 = 0.93$ and CCC = 0.97 for D.
- $R^2 = 0.99$ and CCC = 1.0 for C+D.

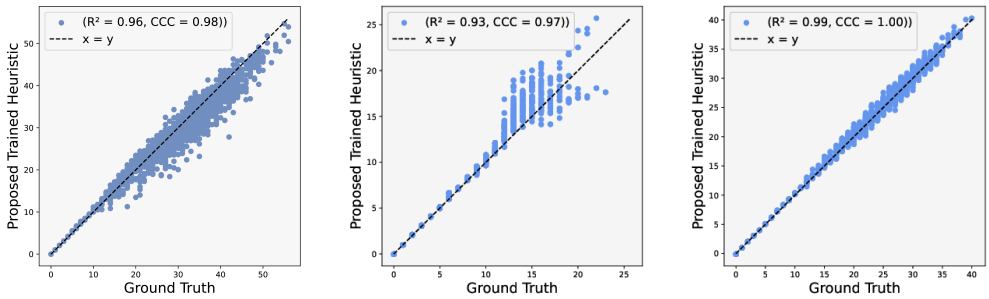
In comparison, DeepCubeA achieved near-perfect results (R^2 and CCC of 1.0) across all domains except for C, where $R^2 = 0.99$ and CCC = 1.0 (Figures 2 and 3). Results for 8-puzzle and 24-puzzle are provided in the Supplementary Material.

To assess performance, we compare UAVI, DeepCubeA, AGN, FGN, GOOSE, and the Fast Downward Planner using the Fast Forward heuristic (FD(FF)) across puzzle sizes (8-puzzle, 15-puzzle, and 24-puzzle) and action domains (C, D, and C+D). UAVI, AGN, FGN, GOOSE and DeepCubeA used weighted A* search with a weight of 0.8 and a batch size of 1000 under a 200-second time limit, while FD(FF) employed standard A* search with the same time constraint.



(a) Without Action Info (b) With Action Info

Figure 1: Comparison of heuristic values predicted by the UAVI (1b) and the model without action information (1a) against ground truth heuristic values for 15-puzzle. The model with action information performs significantly better.



(a) C: P vs GT (b) D: P vs GT (c) C+D: P vs GT

Figure 2: Comparison of trained and ground truth (GT) heuristic values for the 15-puzzle domain for the UAVI (P) variants, each showing heuristic values for 1500 states across canonical actions (C), diagonal actions (D), and canonical + diagonal actions (C+D).

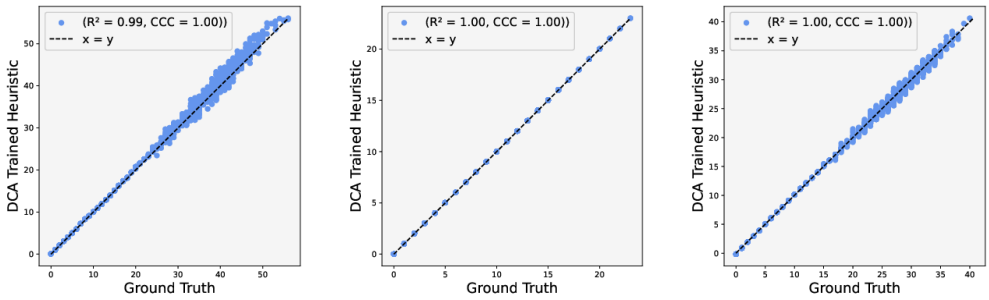
Comparing solvers. Table 1 presents the results for the 15-puzzle domain, comparing solvers across canonical (C), diagonal (D), and combined canonical + diagonal (C+D) action variants. UAVI and DeepCubeA, as domain-specific models, achieve near-optimal results across all variants. UAVI demonstrates strong generalization and matches DeepCubeA's 100% success rate in all cases, with slightly reduced optimality in the canonical domain (C) and higher efficiency in the diagonal domain (D) (3.28E+04 vs. 2.73E+04 nodes/second). In the C+D domain, UAVI achieves near-optimal performance, illustrating its ability to balance generalization and computational efficiency.

AGN and FGN, trained on PDDLFuse-generated domains, show robust generalization and significantly outperform the supervised GOOSE model. AGN achieves a 100% success rate across all domains but with lower efficiency and optimality compared to UAVI and DeepCubeA. FGN demonstrates competitive performance (e.g., 94% success in C+D) but does not match AGN's generalizability. GOOSE, hindered by its reliance on supervised learning, struggles to handle diverse action spaces, with success rates as low as 40% in C and 70% in C+D. While RL-based approaches like AGN and FGN offer superior generalization, they require significantly more data and training time (approximately 12 days) compared to GOOSE, which completes training in about one day. Results for 8-puzzle and 24-puzzle are provided in the Supplementary Material.

Discussion and Future Work

To the best of our knowledge, this is one of a kind study leveraging reinforcement learning to achieve generalization across pathfinding domains using Deep Approximate Value Iteration (DAVI). In contrast to domain-specific models like DeepCubeA, our approach integrates state transition information into state representations, enabling models like UAVI, AGN, and FGN to adapt to unseen domains without retraining. UAVI demonstrates strong generalization and competitive performance with domain-specific planners, achieving high efficiency and success rates across various 15-puzzle action space variation domains.

AGN and FGN illustrate the benefits of training on PDDLFuse-generated action-variation domains, emphasizing the value of reinforcement learning and domain randomization in generalization. AGN achieves robust generalization and adaptability across diverse domains, while FGN extends these capabilities to broader base domains,



(a) C: DCA vs GT (b) D: DCA vs GT (c) C+D: DCA vs GT

Figure 3: Comparison of trained and ground truth (GT) heuristic values for the 15-puzzle domain for DeepCubeA (DCA) variants, each showing heuristic values for 1500 states across canonical actions (C), diagonal actions (D), and canonical + diagonal actions (C+D).

Domain	Solver	Len	Opt	Nodes	Secs	Nodes/Sec	Solved
15 Puzzle **(C)**	DeepCubeA	**52.03**	**99.4%**	**1.82E+05**	**4.31**	4.22E+04	**100%**
	UAVI	52.18	93.76%	3.62E+05	10.39	3.48E+04	**100%**
	AGN	59.70	75%	5.00E+05	30.39	1.67E+04	**100%**
	FGN	73.60	7%	7.00E+06	85.63	8.24E+04	70%
	FD(FF)	52.75	24.80%	2.92E+06	42.11	**6.94E+04**	80.80%
	GOOSE	80.20	3%	8.00E+06	100.08	8.00E+04	40%
15 Puzzle **(D)**	DeepCubeA	**10.80**	**100%**	8.20E+02	**0.03**	2.73E+04	**100%**
	UAVI	10.81	99.8%	1.64E+03	0.05	**3.28E+04**	**100%**
	AGN	14.78	90%	2.50E+03	3.03	8.33E+02	**100%**
	FGN	20.22	60%	3.00E+04	8.10	3.75E+03	**100%**
	FD(FF)	10.86	96.8%	**4.18E+01**	0.21	1.99E+02	**100%**
	GOOSE	22.20	50%	4.00E+04	12.30	3.33E+03	90%
15 Puzzle **(C+D)**	DeepCubeA	**25.66**	**100%**	1.78E+05	**3.74**	**4.76E+04**	**100%**
	UAVI	25.67	99.8%	1.78E+05	4.72	3.77E+04	**100%**
	AGN	29.90	70%	2.50E+06	20.05	1.25E+05	**100%**
	FGN	39.36	36%	7.00E+05	30.11	2.33E+04	94%
	FD(FF)	29.32	13.4%	**8.40E+03**	**1.17**	7.18E+03	**100%**
	GOOSE	45.74	20%	8.00E+05	40.21	2.00E+04	70%

Table 1: Performance comparison on the 15-puzzle domain across different solvers and action variants. The UAVI model demonstrates high optimality and efficiency, surpassing PDDL-Fuse variants and GOOSE, especially in complex action sets.

though at a slightly higher computational cost and reduced efficiency. Comparatively, supervised models like GOOSE struggle with generalizability, particularly in complex and diverse domains, highlighting the limitations of domain-independent supervised approaches.

Future work will focus on refining these models by exploring advanced techniques, such as dynamic PDDL-based domain generation and integrating knowledge graphs to encode transitions. Incorporating knowledge graphs could improve heuristic function accuracy by allowing explicit representation of domain constraints and transitions, particularly in dynamic applications like robotics. These advancements aim to create more generalizable and robust heuristic functions capable of solving pathfinding problems in a wider variety of domains with enhanced efficiency and adaptability.

Conclusion

This work demonstrates that integrating state transition information with state representations enables heuristic functions, such as those in UAVI, AGN, and FGN, to generalize across diverse 15-puzzle action space variation domains. UAVI showcases competitive performance with domain-specific planners like DeepCubeA, while AGN and FGN leverage reinforcement learning and domain randomization to achieve significant generalization improvements compared to supervised models like GOOSE. These findings underscore the potential of reinforcement learning to enhance generalization and efficiency in solving complex pathfinding problems. Future work will expand these techniques to broader domains and explore integrating graph-based models and knowledge graphs to further enhance adaptability, efficiency, and robustness.

References

Agostinelli, F.; McAleer, S.; Shmakov, A.; and Baldi, P. 2019. Solving the Rubik's cube with deep reinforcement learning and search. *Nature Machine Intelligence*, 1(8): 356–363.

Agostinelli, F.; Panta, R.; and Khandelwal, V. 2024. Specifying goals to deep neural networks with answer set programming. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 34, 2–10.

Ajani, O. S.; Hur, S.-h.; and Mallipeddi, R. 2023. Evaluating Domain Randomization in Deep Reinforcement Learning Locomotion Tasks. *Mathematics*, 11(23): 4744.

Besta, M.; Blach, N.; Kubicek, A.; Gerstenberger, R.; Gianinazzi, L.; Gajda, J.; Lehmann, T.; Podstawski, M.; Niewiadomski, H.; Nyczyk, P.; et al. 2023. Graph of thoughts: Solving elaborate problems with large language models. *arXiv preprint arXiv:2308.09687*.

Chen, D. Z.; Thiébaux, S.; and Trevizan, F. 2023. GOOSE: Learning domain-independent heuristics. In *NeurIPS 2023 Workshop on Generalization in Planning*.

Chen, D. Z.; Thiébaux, S.; and Trevizan, F. 2024. Learning Domain-Independent Heuristics for Grounded and Lifted Planning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, 20078–20086.

Chicco, D.; Warrens, M. J.; and Jurman, G. 2021. The coefficient of determination R-squared is more informative than SMAPE, MAE, MAPE, MSE and RMSE in regression analysis evaluation. *Peerj computer science*, 7: e623.

Dalal, M.; Chiruvolu, T.; Chaplot, D. S.; and Salakhutdinov, R. 2023. Plan-Seq-Learn: Language Model Guided RL for Solving Long Horizon Robotics Tasks. In *2nd Workshop on Language and Robot Learning: Language as Grounding*.

Gramopadhye, M.; and Szafir, D. 2022. Generating executable action plans with environmentally-aware language models. *arXiv preprint arXiv:2210.04964*.

Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2): 100–107.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 770–778.

Helmert, M. 2006. The fast downward planning system. *Journal of Artificial Intelligence Research*, 26: 191–246.

Hoffmann, J.; and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14: 253–302.

Hu, H.; Lu, H.; Zhang, H.; Lam, W.; and Zhang, Y. 2023. Chain-of-Symbol Prompting Elicits Planning in Large Langauge Models. *arXiv preprint arXiv:2305.10276*.

Huang, W.; Wang, C.; Zhang, R.; Li, Y.; Wu, J.; and Fei-Fei, L. 2023. Voxposer: Composable 3d value maps for robotic manipulation with language models. *arXiv preprint arXiv:2307.05973*.

Kang, C.; Chang, W.; and Choi, J. 2024. Balanced Domain Randomization for Safe Reinforcement Learning. *Applied Sciences*, 14(21): 9710.

Keith, M. 2011. Vintage plastic sliding-letter puzzles. *Word Ways*, 44(4): 22.

Khandelwal, V.; Sheth, A.; and Agostinelli, F. 2024. PDDL-Fuse: A Tool for Generating Diverse Planning Domains. *arXiv preprint*.

Koo, S.; Yu, H.; and Lee, G. G. 2019. Adversarial approach to domain adaptation for reinforcement learning on dialog systems. *Pattern Recognition Letters*, 128: 467–473.

Lawrence, I.; and Lin, K. 1989. A concordance correlation coefficient to evaluate reproducibility. *Biometrics*, 255–268.

Li, Y.; Kamra, N.; Desai, R.; and Halevy, A. 2023. Human-Centered Planning. *arXiv preprint arXiv:2311.04403*.

Logeswaran, L.; Sohn, S.; Lyu, Y.; Liu, A. Z.; Kim, D.-K.; Shim, D.; Lee, M.; and Lee, H. 2023. Code Models are Zero-shot Precondition Reasoners. *arXiv preprint arXiv:2311.09601*.

Lu, Y.; Lu, P.; Chen, Z.; Zhu, W.; Wang, X. E.; and Wang, W. Y. 2023. Multimodal Procedural Planning via Dual Text-Image Prompting. *arXiv preprint arXiv:2305.01795*.

Mehta, B.; Diaz, M.; Golemo, F.; Pal, C. J.; and Paull, L. 2020. Active domain randomization. In *Conference on Robot Learning*, 1162–1176. PMLR.

Pallagani, V.; Muppasani, B.; Murugesan, K.; Rossi, F.; Horesh, L.; Srivastava, B.; Fabiano, F.; and Loreggia, A. 2022. Plansformer: Generating symbolic plans using transformers. *arXiv preprint arXiv:2212.08681*.

Pallagani, V.; Muppasani, B.; Murugesan, K.; Rossi, F.; Srivastava, B.; Horesh, L.; Fabiano, F.; and Loreggia, A. 2023a. Understanding the Capabilities of Large Language Models for Automated Planning. *arXiv preprint arXiv:2305.16151*.

Pallagani, V.; Muppasani, B.; Srivastava, B.; Rossi, F.; Horesh, L.; Murugesan, K.; Loreggia, A.; Fabiano, F.; Joseph, R.; Kethepalli, Y.; et al. 2023b. Plansformer Tool: Demonstrating Generation of Symbolic Plans Using Transformers. In *IJCAI*, volume 2023, 7158–7162. International Joint Conferences on Artificial Intelligence.

Parakh, M.; Fong, A.; Simeonov, A.; Gupta, A.; Chen, T.; and Agrawal, P. 2023. Human-Assisted Continual Robot Learning with Foundation Models. *arXiv preprint arXiv:2309.14321*.

Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32.

Schmidhuber, J. 2015. Deep learning in neural networks: An overview. *Neural networks*, 61: 85–117.

Sermanet, P.; Ding, T.; Zhao, J.; Xia, F.; Dwibedi, D.; Gopalakrishnan, K.; Chan, C.; Dulac-Arnold, G.; Maddineni, S.; Joshi, N. J.; et al. 2023. RoboVQA: Multimodal Long-Horizon Reasoning for Robotics. *arXiv preprint arXiv:2311.00899*.

Sievers, S. 2018. Merge-and-shrink heuristics for classical planning: Efficient implementation and partial abstractions. In *Proceedings of the International Symposium on Combinatorial Search*, volume 9, 90–98.

Silver, T.; Dan, S.; Srinivas, K.; Tenenbaum, J. B.; Kaelbling, L. P.; and Katz, M. 2023. Generalized Planning in PDDL Domains with Pretrained Large Language Models. *arXiv preprint arXiv:2305.11014*.

Singh, I.; Blukis, V.; Mousavian, A.; Goyal, A.; Xu, D.; Tremblay, J.; Fox, D.; Thomason, J.; and Garg, A. 2023. ProgPrompt: program generation for situated robot task planning using large language models. *Autonomous Robots*, 1–14.

Sutton, R. S.; and Barto, A. G. 2018. *Reinforcement learning: An introduction*. MIT press.

Toyer, S.; Trevizan, F.; Thiébaux, S.; and Xie, L. 2018. Action schema networks: Generalised policies with deep learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32.

Valmeekam, K.; Marquez, M.; and Kambhampati, S. 2023. Can Large Language Models Really Improve by Self-critiquing Their Own Plans? *arXiv preprint arXiv:2310.08118*.

Valmeekam, K.; Olmo, A.; Sreedharan, S.; and Kambhampati, S. 2022. Large Language Models Still Can't Plan (A Benchmark for LLMs on Planning and Reasoning about Change). *arXiv preprint arXiv:2206.10498*.

Wang, L.; Xu, W.; Lan, Y.; Hu, Z.; Lan, Y.; Lee, R. K.-W.; and Lim, E.-P. 2023. Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models. *arXiv preprint arXiv:2305.04091*.

Yao, S.; Yu, D.; Zhao, J.; Shafran, I.; Griffiths, T. L.; Cao, Y.; and Narasimhan, K. 2023. Tree of thoughts: Deliberate problem solving with large language models. *arXiv preprint arXiv:2305.10601*.

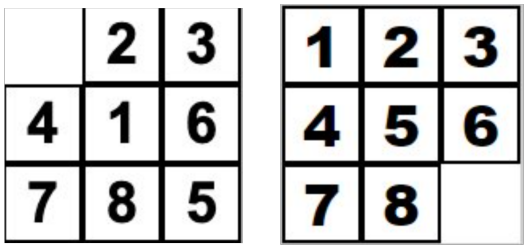
Zelikman, E.; Huang, Q.; Poesia, G.; Goodman, N.; and Haber, N. 2023. Parsel: Algorithmic Reasoning with Language Models by Composing Decompositions. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Appendix

Background and Literature Review

N-Puzzle

The n-puzzle, also known as the sliding tile puzzle (Keith 2011), is a classic problem in artificial intelligence. The puzzle consists of a grid of $n + 1$ tiles, where n tiles are numbered sequentially, and one tile is left empty. The objective is to rearrange the tiles from a given initial state to a goal state, typically with tiles ordered sequentially from top-left to bottom-right, and the empty tile in the bottom-right corner, as shown in Figure 4b. Traditionally, actions correspond to moving a tile into the empty space, which can be one of the following moves: up (U), down (D), left (L), and right (R). These moves define the canonical action set. In addition to the canonical moves, we consider diagonal actions: upper-left (UL), upper-right (UR), lower-left (DL), and lower-right (DR). These additional moves increase the possible transitions from any given state. Figure 4 illustrates an example of a scrambled state and the goal state.



(a) Scrambled State (b) Goal State

Figure 4: Example of a scrambled state and the goal state for the 8-puzzle domain. The cost-to-go is significantly reduced when including diagonal moves: 16 steps for canonical moves only versus 2 steps for canonical and diagonal moves combined.

Large Language Models in Pathfinding Problems

Large language models (LLMs) are a foundation model pretrained on extensive textual datasets using self-supervised learning techniques. These models, characterized by their ability to generate and comprehend human-like text, have shown potential in various natural language processing (NLP) tasks. LLMs can be fine-tuned as foundation models for several downstream tasks, such as machine translation and question-answering. There have been several attempts to use LLMs for solving pathfinding problems and generating a sequence of actions to solve the problem.

While attempts to use causal LLMs to solve pathfinding problems via in-context learning have shown modest performance and indicates notable challenges (Sermanet et al. 2023; Li et al. 2023; Silver et al. 2023; Parakh et al. 2023; Zelikman et al. 2023; Besta et al. 2023; Huang et al. 2023; Dalal et al. 2023; Wang et al. 2023; Valmeekam et al. 2022; Valmeekam, Marquez, and Kambhampati 2023; Gramopadhye and Szafir 2022; Singh et al. 2023). Other prompting techniques have been introduced to enhance LLMs' reasoning capabilities (Hu et al. 2023; Yao et al. 2023).

Efforts to generate multimodal, text, and image-based sequences of actions are exemplified by (Lu et al. 2023). Additionally, a subset of studies investigates the fine-tuning of seq2seq, code-based language models (Pallagani et al. 2022, 2023b), which are noted for their advanced syntactic encoding. These models show promise within the confines of their training datasets (Logeswaran et al. 2023) yet exhibit limitations in generalizing to out-of-distribution domains (Pallagani et al. 2023a), highlighting a gap in their adaptability across diverse pathfinding problem domains.

One of the major drawbacks of LLMs is that they do not inherently possess search capabilities, nor do these papers employ heuristic search techniques. This limitation presents a significant challenge in applying LLMs to pathfinding problems, as effective search strategies are crucial for optimal solutions.

Domain-Independent Planning

The Fast Downward planner (Helmert 2006) is a well-known automated planning system that utilizes various heuristics to solve complex pathfinding problems. One key heuristic employed by Fast Downward is the Fast Forward (FF) heuristic (Hoffmann and Nebel 2001). The FF heuristic estimates the cost-to-go by ignoring the delete effects of actions, allowing it to compute heuristic values with minimal computational overhead rapidly. These heuristics are domain-independent, meaning they can generalize across various planning domains without requiring domain-specific adjustments. This generalizability makes them highly versatile and efficient for various applications. However, despite their efficiency, domain-independent heuristics do not perform as well as learned heuristics (Agostinelli, Panta, and Khandelwal 2024). This underscores the limitations of domain-independent heuristics and highlights the potential benefits of leveraging training to achieve superior pathfinding performance.

Methodology

Environment Generator

This section lists the environment generator algorithm.

Evaluation Metrics

The correlation between the learned heuristic value and the ground truth heuristic value is evaluated using two statistical measures:

1. Concordance Correlation Coefficient (CCC): Denoted as ρ_c, CCC measures the agreement between two variables (learned and true heuristic values), considering both the precision and accuracy (Lawrence and Lin 1989). It is defined as:

$$\rho_c = \frac{2\rho\sigma_x\sigma_y}{\sigma_x^2 + \sigma_y^2 + (\mu_x - \mu_y)^2}$$

Where ρ is the Pearson correlation coefficient between the variables, σ_x and σ_y are the standard deviations, and μ_x and μ_y are the means of the true and predicted values, respectively.

Algorithm 1: Environment Generator - NPuzzle with action variation

Input: n - dimension of the puzzle grid (e.g., $n = 3$ for an 8-puzzle)
Output: Puzzle environment P with reversible actions validated
procedure INITIALIZEANDVALIDATEPUZZLE(n)
    Let P be an $n \times n$ grid of cells
    Define $A = \{\text{U, D, L, R, UL, UR, DL, DR}\}$ as the set of all possible actions
    for each cell $c_{ij} \in P$ **do**
        Assign to $c_{ij}.A$ a random subset of A
    end for
    for each cell $c_{ij} \in P$ **do**
        for each action $a \in c_{ij}.A$ **do**
            Determine the target cell c_{kl} from c_{ij} using action a
            Identify the reverse action a' corresponding to a
            if $a' \notin c_{kl}.A$ **then**
                Remove a from $c_{ij}.A$
            end if
        end for
    end for
    return P
end procedure

This metric quantifies the proportion of variance in the dependent variable (ground truth heuristic values) that is predictable from the independent variable (predicted heuristic values). It primarily measures the trend of the predictive accuracy but does not account for how close the predictions are to the identity line (perfect agreement).

2. Coefficient of Determination (R-squared, R^2): This metric quantifies the proportion of variance in the dependent variable that is predictable from the independent variable, providing insight into the goodness of fit of the model to the data (Chicco, Warrens, and Jurman 2021).

$$R^2 = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \bar{y})^2}$$

where y_i are the ground truth values, $\hat{y}_i$ are the predicted heuristic values, and $\bar{y}$ is the mean of the ground truth values.

This metric assesses the agreement between two variables, considering the precision and the accuracy. This metric captures the relationship trend between predicted and true values and evaluates how closely the predictions conform to the identity line, indicating perfect prediction accuracy.

In addition to these statistical metrics, we also evaluate the models on planning-specific metrics: **path length**, **optimality percentage**, **number of nodes expanded**, **time taken**, **nodes expanded per second**, and **percentage of problems solved (% solved)**. These metrics provide a comprehensive assessment of model performance: - *Path length* and *optimality percentage* evaluate the quality of the generated solutions compared to optimal solutions. - *Number of nodes expanded* and *nodes expanded per second* measure the efficiency of the heuristic in guiding the search process. - *Time taken* captures computational efficiency. - *% solved* reflects the robustness of the model in solving problems within a given time limit.

These metrics are critical for understanding the practical applicability and computational feasibility of the UAVI models in real-world planning tasks.

Experimental Setup

Hardware Details

We have used two servers to run our experiments: one with 48-core nodes each hosting 2 V100 32G GPUs and 128GB of RAM, and another with 256 cores, eight A100 40GB GPUs, and 1TB of RAM. The processor speed is 2.8 GHz.

Test Data Generation

To evaluate the performance of UAVI, we generated three distinct test datasets:

- **Data1:** 500 states per domain (C, D, and C+D) generated via random walks ranging from 1,000 to 10,000 steps from the goal state.
- **Data2:** 100 states for each of 500 unique action space variation domains, generated via random walks of up to 500 steps from the goal state.
- **Data3:** 1,500 states per domain (C, D, and C+D) generated via random walks of 0 to 500 steps from the goal state, specifically designed to evaluate heuristic accuracy against optimal ground truth values.

These datasets were essential for thoroughly comparing the models' performance and adaptability across different action spaces.

Using the Fast Downward planner with the merge-and-shrink heuristic (Sievers 2018), we obtained the ground truth heuristic values for all generated test data. This heuristic simplifies the state space by merging and shrinking states, providing optimal heuristics. However, obtaining ground truth for complex puzzles like the 24-puzzle is computationally intensive.

For **Data1**:

- **8-puzzle**: The process took approximately one day with a cutoff of 160 minutes per problem, running one instance of the planner per domain.
- **15-puzzle**: The process took about a week with a cutoff of 160 minutes per problem, running five instances of the planner per domain.
- **24-puzzle**: The process took around 2 weeks with a cutoff of 144 hours per problem, running 50 instances of the planner per domain.

For **Data2**:

- **8-puzzle**: The process took one week with a cutoff of 160 minutes per problem, running 50 instances overall.
- **15-puzzle**: The process also took one week with a cutoff of 160 minutes per problem, running 500 instances overall.

For **Data3**:

- **8-puzzle**: The process took approximately 2 days with a cutoff of 160 minutes per problem, running one instance of the planner per domain.
- **15-puzzle**: The process took about a week with a cutoff of 160 minutes per problem, running 15 instances of the planner per domain.
- **24-puzzle**: The process took over 3 weeks, running 100 instances per domain, with 15 problems per instance and a cutoff of 144 hours per problem.

These computations highlight the significant computational resources and time required to obtain ground truth heuristic values for large-scale puzzles, underscoring the challenges in benchmarking heuristic functions for complex domains.

Results

To quantitatively assess the performance of the trained heuristic model, we employ several metrics that reflect both the accuracy of the heuristic values and the efficiency of the pathfinding process.

Plotting Ground Truth Heuristic Values 8-Puzzle. We generate **(D2)** for the 8-puzzle domain. We plot the ground truth optimal heuristic values against the trained heuristic values for both the UAVI model (with action information) and the model trained without action information, as shown in Figure 5. The UAVI model achieved a CCC of 0.98 and $R2$ of 0.97, while the model without action information achieved a CCC of 0.69 and $R2$ of 0.44, demonstrating the UAVI model's superior performance.

For the 8-puzzle domain (Figure 6), DCA graphs show a perfect correlation with R^2 and CCC of 1.0. The UAVI model achieves an R^2 of 0.96 and CCC of 0.98 for the C only domain, and perfect correlation (R^2 and CCC of 1.0) for the D and C+D domains. The UAVI model demonstrates near-perfect accuracy, especially in the more complex action domains.

Plotting Ground Truth Heuristic Values 24-Puzzle. For the 24-puzzle domain (Figure 7), DCA graphs show perfect correlation (R^2 and CCC of 1.0). The UAVI model achieves R^2 of 0.97 and CCC of 0.98 for C only, R^2 of 0.94 and CCC of 0.97 for D only, and near-perfect correlation (R^2 of 0.98 and CCC of 0.99) for C+D. The UAVI model maintains high accuracy across all domains, demonstrating its generalizability.

To assess performance, we compare UAVI, DeepCubeA, AGN, FGN, GOOSE, and the Fast Downward Planner using the Fast Forward heuristic (FD(FF)) across puzzle sizes (8-puzzle, 15-puzzle, and 24-puzzle) and action domains (C, D, and C+D). UAVI, AGN, FGN, GOOSE and DeepCubeA used weighted A* search with a weight of 0.8 and a batch size of 1000 under a 200-second time limit, while FD(FF) employed standard A* search with the same time constraint.

Comparing solvers for 8-Puzzle. Table 2 presents the results for the 8-puzzle domain, comparing solver performance across canonical (C), diagonal (D), and combined canonical + diagonal (C+D) action variants.

UAVI and DeepCubeA achieve near-identical performance, with both models consistently solving 100% of the problems across all action variants. In the canonical domain (C), UAVI matches DeepCubeA in solution length (18.38) and optimality while demonstrating slightly reduced efficiency (4.07E+04 nodes/second compared to 5.20E+04 for DeepCubeA). In the diagonal domain (D), UAVI maintains its 100% success rate with slightly improved efficiency (4.05E+03 nodes/second compared to 1.95E+03 for DeepCubeA). Similarly, in the C+D domain, UAVI matches DeepCubeA in optimality and solution length, processing nodes at a comparable rate (3.99E+04 nodes/second).

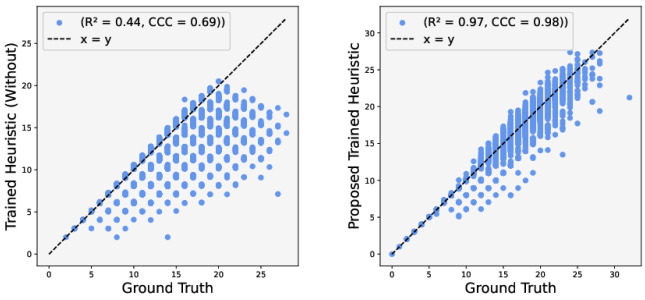
AGN and FGN, trained on PDDLFuse-generated domains, exhibit strong generalizability but with reduced efficiency and success rates compared to UAVI and DeepCubeA. AGN achieves 100% success in the canonical and diagonal domains, with solution lengths slightly longer than UAVI and DeepCubeA (e.g., 18.50 in C and 1.46 in D). FGN demonstrates moderate generalization, achieving 100% success in the canonical domain but falling to 90% in the C+D domain, with significantly higher computational costs and reduced efficiency.

GOOSE, relying on supervised learning, shows limited generalizability and struggles with efficiency. In the canonical domain, GOOSE achieves 100% success but with a longer average solution length (22.00) and lower nodes processed per second (3.00E+04). Its performance further deteriorates in the diagonal and C+D domains, achieving only 55% and 50% success rates, respectively, highlighting its limitations in handling diverse action configurations.

Overall, UAVI and DeepCubeA deliver the best performance, combining optimality, efficiency, and adaptability across all action variants. AGN and FGN demonstrate the strengths of reinforcement learning-based approaches in generalization, although their efficiency and success rates are less consistent. GOOSE highlights the challenges faced by supervised learning-based domain-independent planners, further emphasizing the benefits of reinforcement learning and domain randomization for handling diverse problem configurations.

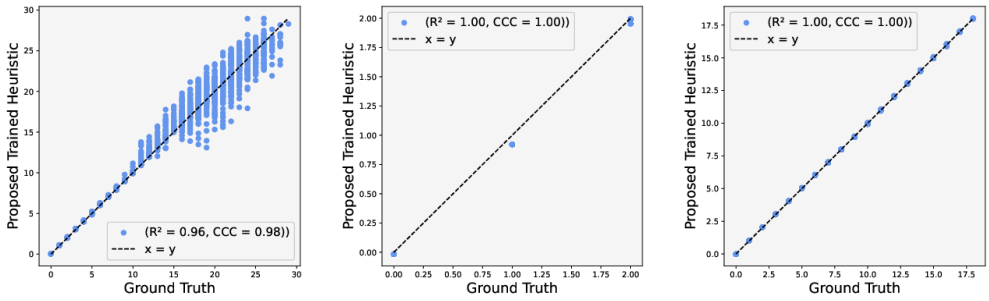
Comparing solvers for 24-Puzzle. Table 3 presents the results for the 24-puzzle domain, comparing solver performance across canonical (C), diagonal (D), and combined canonical + diagonal (C+D) action variants.

UAVI and DeepCubeA maintain strong performance despite the increased complexity of the 24-puzzle. DeepCubeA achieves near-optimal results across all domains, consistently solving 100% of the problems. UAVI performs competitively, matching DeepCubeA's success rate of 100% while demonstrating slightly reduced optimality in some cases. In the canonical domain (C), UAVI achieves a success rate of 100%, with a marginally longer average solution length (92.80 vs. 89.48) and slightly lower efficiency (3.16E+04 nodes/second compared to 4.15E+04 for DeepCubeA). In the diagonal domain (D), UAVI maintains a 100% success rate with efficiency (3.78E+04 nodes/second) close to that of DeepCubeA (5.43E+04). Similarly, in the C+D domain, UAVI achieves 100% success with near-

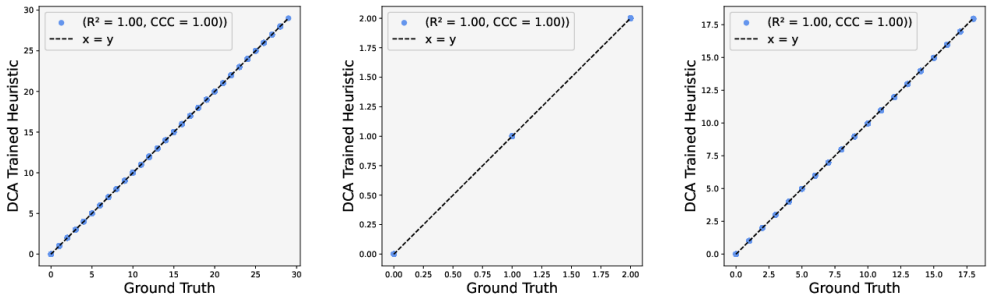


(a) Without Action Info (b) With Action Info

Figure 5: Comparison of heuristic values predicted by the UAVI model (5b) and the model without action information (5a) against ground truth heuristic values for 8-puzzle. The model with action information performs significantly better.

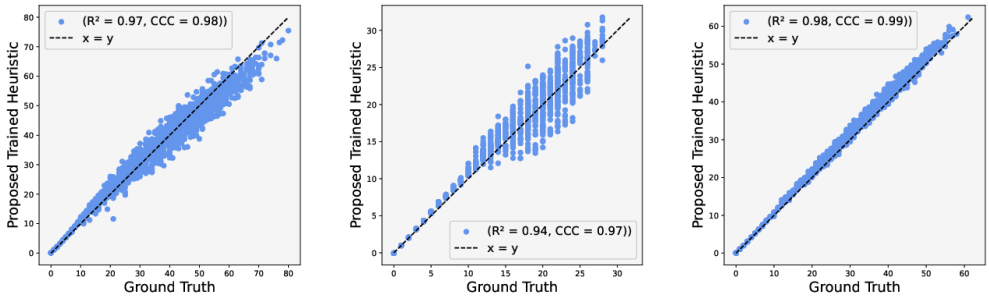


(a) C: UAVI vs GT (b) D: UAVI vs GT (c) C+D: UAVI vs GT

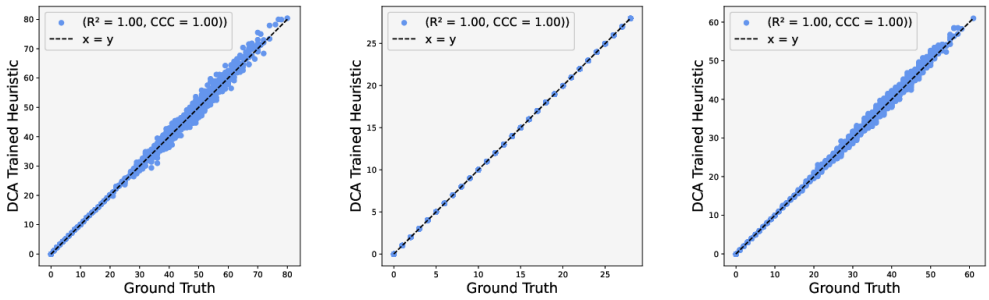


(d) C: DCA vs GT (e) D: DCA vs GT (f) C+D: DCA vs GT

Figure 6: Comparison of trained and ground truth (GT) heuristic values for the 8-puzzle domain. 6a, 6b, 6c for the UAVI model (UAVI), and 6d, 6e, 6f for DeepCubeA (DCA) variants, each showing heuristic values for 1500 states across canonical actions (C), diagonal actions (D), and canonical + diagonal actions (C+D).



(a) C: UAVI vs GT (b) D: UAVI vs GT (c) C+D: UAVI vs GT



(d) C: DCA vs GT (e) D: DCA vs GT (f) C+D: DCA vs GT

Figure 7: Comparison of trained and ground truth (GT) heuristic values for the 24-puzzle domain. 7a, 7b, 7c for the UAVI model (UAVI), and 7d, 7e, 7f for DeepCubeA (DCA) variants, each showing heuristic values for 1500 states across canonical actions (C), diagonal actions (D), and canonical + diagonal actions (C+D).

Domain	Solver	Len	Opt	Nodes	Secs	Nodes/Sec	Solved
8 Puzzle **(C)**	DeepCubeA	**18.38**	**100%**	3.59E+04	0.69	**5.20E+04**	**100%**
	UAVI	**18.38**	**100%**	7.17E+04	1.76	4.07E+04	**100%**
	AGN	18.50	98%	8.50E+04	3.50	2.43E+04	**100%**
	FGN	20.50	55%	1.10E+05	6.50	1.69E+04	**100%**
	FD(FF)	18.80	81%	**5.56E+02**	**0.11**	5.05E+03	**100%**
	GOOSE	22.00	60%	1.20E+05	4.00	3.00E+04	**100%**
8 Puzzle **(D)**	DeepCubeA	**1.44**	**100%**	1.95E+01	**0.01**	1.95E+03	**100%**
	UAVI	**1.44**	**100%**	4.05E+01	0.01	**4.05E+03**	**100%**
	AGN	1.46	98%	5.00E+01	0.05	1.00E+03	**100%**
	FGN	1.60	65%	6.00E+01	0.50	1.20E+02	**100%**
	FD(FF)	**1.44**	**100%**	**2.45E+00**	0.20	1.23E+01	**100%**
	GOOSE	1.70	55%	5.00E+01	0.50	1.00E+02	95%
8 Puzzle **(C+D)**	DeepCubeA	**11.84**	**100%**	6.20E+04	**1.18**	**5.26E+04**	**100%**
	UAVI	**11.84**	**100%**	6.23E+04	1.56	3.99E+04	**100%**
	AGN	11.90	96%	7.00E+04	3.50	2.00E+04	**100%**
	FGN	20.50	50%	7.50E+05	12.00	6.25E+04	90%
	FD(FF)	12.90	54.2%	**8.68E+01**	**0.13**	6.68E+02	**100%**
	GOOSE	22.00	50%	8.00E+04	5.50	1.45E+04	90%

Table 2: Performance comparison on the 8-puzzle domain across different solvers and action variants. The UAVI model matches DeepCubeA in optimality and solution length, outperforming PDDL-Fuse variants and GOOSE in efficiency and nodes processed per second.

Domain	Solver	Len	Opt	Nodes	Secs	Nodes/Sec	Solved
24 Puzzle **(C)**	DeepCubeA	**89.48**	**96.98%**	**3.34E+05**	**8.05**	**4.15E+04**	**100%**
	UAVI	92.80	22.03%	7.60E+05	24.06	3.16E+04	**100%**
	AGN	97.50	53%	1.00E+06	63.30	1.57E+04	80%
	FGN	121.24	5%	8.00E+06	167.08	4.78E+04	33%
	FD(FF)	81.00	0.40%	2.68E+06	89.84	**2.98E+04**	1.01%
	GOOSE	125.30	0.5%	9.00E+06	225.60	3.98E+04	5%
24 Puzzle **(D)**	DeepCubeA	**14.90**	**100%**	2.55E+04	**0.47**	**5.43E+04**	**100%**
	UAVI	14.92	99.8%	5.10E+04	1.35	3.78E+04	**100%**
	AGN	16.45	76%	1.00E+05	10.2	1.00E+04	**100%**
	FGN	29.18	30%	1.50E+06	40.32	3.75E+04	55%
	FD(FF)	15.16	89.2%	**2.64E+02**	**0.12**	2.20E+03	**100%**
	GOOSE	35.10	15%	2.00E+06	60.98	3.33E+04	40%
24 Puzzle **(C+D)**	DeepCubeA	**31.33**	**100%**	2.27E+05	**4.83**	**4.70E+04**	**100%**
	UAVI	31.34	99.6%	2.27E+05	6.78	3.35E+04	**100%**
	AGN	35.75	52.6%	3.50E+05	25.90	1.40E+04	83%
	FGN	72.42	11%	1.00E+06	90.34	1.11E+04	64%
	FD(FF)	36.81	13.8%	**1.70E+04**	**5.35**	3.18E+03	99.4%
	GOOSE	75.80	5%	1.20E+06	110.21	1.09E+04	40%

Table 3: Performance comparison on the 24-puzzle domain across different solvers and action variants. The UAVI model maintains high performance, while PDDL-Fuse variants and GOOSE show significant performance degradation due to increased puzzle complexity and limited training data.

optimal results, matching DeepCubeA in nodes processed.

AGN and FGN, trained on PDDLFuse-generated domains, exhibit strong generalization but face challenges in handling the increased complexity of the 24-puzzle. AGN achieves competitive success rates across all action variants, including 100% in the diagonal domain (D) and 83% in the C+D domain. However, AGN incurs higher computational costs and reduced efficiency, processing fewer nodes per second compared to UAVI and DeepCubeA. FGN shows moderate generalization, achieving a 64% success rate in the C+D domain but struggling with efficiency and optimality, as seen in its significantly longer solution lengths (e.g., 72.42 in C+D).

GOOSE, relying on supervised learning, struggles significantly with the 24-puzzle's complexity. Its success rates drop to 5% in the canonical domain (C) and 40% in the diagonal domain (D), highlighting its inability to generalize effectively to larger problem spaces. Additionally, GOOSE demonstrates limited efficiency and suboptimal solution lengths across all domains.

Overall, UAVI and DeepCubeA outperform other solvers, maintaining robust performance and high efficiency. AGN and FGN demonstrate the potential of reinforcement learning and domain randomization for generalization, though they face limitations with larger puzzles. GOOSE's significant performance degradation underscores the limitations of supervised learning-based approaches when addressing complex, high-dimensional problems.

GOOSE vs PDDLFuse variations

The training time for PDDLFuse-based models (AGN and FGN) and the GOOSE model highlights a key distinction in their methodologies. GOOSE is trained in a supervised manner on a fixed dataset of 50,000 examples, requiring approximately one day to complete training. In contrast, PDDLFuse-based models are trained using an unsupervised reinforcement learning approach with Approximate Value Iteration (AVI), taking around 12 days. One major challenge with PDDLFuse is that the generated domains often have significantly higher branching factors compared to the fixed domains used by GOOSE. This increased complexity requires processing a greater number of state transitions and heuristic updates during training, which contributes to the extended training time. While the longer training duration of PDDLFuse models is a limitation, the reinforcement learning framework enables these models to generalize across a much broader set of domains, demonstrating adaptability and performance that supervised approaches like GOOSE cannot achieve.