Proceedings of the 3rd International Conference on Applied Physics and Mathematical Modeling
DOI: 10.54254/2753-8818/2025.DL29641

Learning to Slide: Cross-Size Graph Neural Heuristics for Optimal Puzzle Solving

Simon Cui[1*], Keyin Liu[2], Mingyang Lian[3]

[1]*College of Art and Science, New York University, New York, USA*
[2]*University of Toronto Mississauga, Mississaguga, Canada*
[3]*Faculty of Science and Engineering, University of Ningbo, Ningbo, China*
**Corresponding Author. Email: bc3579@nyu.edu*

Abstract. Heuristic techniques such as A* and IDA* with Manhattan distance or linear conflict as heuristics fall short for larger sliding-tile puzzles. This is quite unfortunate as the sliding-tile puzzle is often used as a heuristic search benchmark. We introduce PuzzleGNN, a size-adaptive graph neural network that learns cross-size heuristics for puzzles ranging from 3× 3 to 7 × 7. By encapsulating tiles and adjacencies as graph nodes and edges and embedding the board size into the model, PuzzleGNN functions as a predictor of optimal step counts for puzzles. After being trained on solutions generated by IDA*, PuzzleGNN achieved R2 ≥ 0.85 within its trained range, performing 10 to 30 times faster in inference compared to A*, and excelling on mid-sized boards with a mean absolute error (MAE) of 0.62 on 6 × 6. Nonetheless, while generalizing PuzzleGNN to unseen configurations remains challenging, a promising approach is to fuse neural and admissible heuristics as a means to achieving optimal, scalable search in combinatorial domains.

Keywords: Graph Neural Network, Sliding-tile puzzle, Heuristic search, Machine learning

1. Introduction

The classic sliding tile puzzle, particularly known as the 15-puzzle, is a well-known test problem for AI, heuristic search, and research in combinatorial optimization. The rules of the puzzle simply involve the moving of 'n' tiles within a 2D grid, and there exists a goal configuration where the tiles must be arranged. The deceptively simple rules of the puzzle lead to exponentially growing state space for larger puzzles. The rapid combinatorial explosion of possibilities, the traditional search methods, becomes computationally intractable, even if they are successful at smaller instances. Methods such as A* and Iterative Deepening A* (IDA*) have been able to efficiently solve the 15-puzzle using well-known heuristics like the Manhattan distance and linear conflict. Optimal solutions for 4x4 puzzles can be obtained in a matter of seconds. The reliance of such methods for 5x5 and 6x6 configurations, however, is far less accurate. This is due to a combination of heuristic accuracy and state space. There are limits to how far pattern databases and domain-specific optimizations can be applied, particularly in achieving real-time results and generalizing to larger instances.

© 2026 The Authors. This is an open access article distributed under the terms of the Creative Commons Attribution License 4.0 (https://creativecommons.org/licenses/by/4.0/).

DOI: 10.54254/2753-8818/2025.DL29641

Graph Neural Networks (GNNs) and advanced machine learning techniques are creating new opportunities for heuristic design. GNN approaches can formulate learning-based strategies and dynamically learn to extract essential features and relations from the states of puzzles, thus addressing complex positional dependencies that are typically ignored by classical heuristics. Additionally, sliding puzzles naturally conform to the area represented by the graph, enabling the use of the GNN's message-passing paradigms for the communication of local and global state information.

This paper presents a new framework called PuzzleGNN, which is a neural heuristic framework based on GNN with a size adapter module which permits a single model to function with puzzles from 3x3 to 7x7 in size. We uniquely contribute: 1. A multi-size sliding puzzle prediction using Puzzle GNN with size-aware embedding is designed with a single unified GNN architecture. 2. A never-ending data generation and processing pipeline, solvable puzzle states with optimal steps, transforms instruction data for supervised learning. 3. Crosssize generalization and long-path prediction accuracy along with computational efficiency is evaluated in detail in relation to traditional heuristics for multiple sizes and Puzzle GNN in a comprehensive approach. This work aims to combine the modern scalablilty methods of learning heuristics for puzzles with classical search optimality, identifying the potential and current boundaries of the approach.

2. Literature review

The sliding-tile puzzle is also known as the 15-puzzle. Its rules are straightforward. A player advances tiles on a board to reach a goal state. Johnson and Story (1879) discovered the parity rule that decides if a state is solvable. Korf proposed Iterative Deepening A*(IDA*) [1]. This method became a standard approach to solve the 15-puzzle. The performance of these methods relies on the heuristic function.

Later researchers developed classical admissible heuristics. Manhattan distance and Linear Conflict worked well on small puzzles [2]. These heuristics are fast to compute and explain. But they become weak on larger puzzles. Misplaced tiles are even faster, but give the least useful information.

Culberson and Schaeffer proposed Pattern Databases (PDBs) [3]. Felner extended this idea [4]. PDBs provide stronger heuristics, as they store solved sub-puzzles. PDBs also reduce the search space. But this comes at the cost of needing very much memory. Furthermore, they work only for a fixed size of puzzle. Zhou and Hansen studied parallel A* to accelerate these methods [5]. But still the state space grows too fast. Some researchers studied how humans solve puzzles. It turned out that humans often plan with subgoals and use high-level strategies. These results suggest that either multi-scale or relational heuristics might be useful.

Machine learning also became an interesting new direction. Some neural networks tried to predict the number of steps to reach the goal or guide search. These models worked only for fixed board sizes. Agostinelli showed that reinforcement learning with search can solve the Rubik's Cube [6]. Graph Neural Networks also became popular for combinatorial problems [7]. One can model tiles as nodes and the adjacencies between them as edges. But most learned heuristics did not generalize to different board sizes.

DOI: 10.54254/2753-8818/2025.DL29641

3. Methodology

3.1. Performance evaluation of A* and IDA* on sliding puzzles

Traditional search algorithms, such as A* [8] and IDA*, have been widely used to solve 15 puzzles. While these algorithms perform well on small puzzles, their effectiveness decreases rapidly as the dimension of the puzzle exceeds 4 × 4.

For an A* algorithm with 15 puzzles, the Manhattan distance heuristic can find the optimal solution in less than a second [9]. However, when applied to large puzzles like 5 × 5 (24 puzzles), the computation time increases to tens of seconds, even for relatively simple puzzles [10]. For 6 × 6 puzzles, this performance degradation is fatal, as computation times often exceed 30 minutes using standard heuristics.

These methods suffer from two main limitations: state space explosion and heuristic inaccuracy. The number of possible states grows factorially with the size of the puzzle ($(n^2)!$), making the computational overhead prohibitive. For example,

- 4×4: $16! \approx 2.09 \times 10^{13}$ states
- 5×5: $25! \approx 1.55 \times 10^{25}$ states
- 6×6: $36! \approx 3.72 \times 10^{41}$ states

In addition, conventional heuristics such as Manhattan distance and linear conflict suffer a significant decrease of accuracy as the size of the puzzle increases.

Recent attempts to improve scalability through pattern databases [3] and machine learning-enhanced heuristics [11] have shown promise, but still face fundamental limitations when $n > 5$. Even optimized IDA* implementations struggle with 6×6 puzzles, frequently requiring more than 10^7 node expansions and minutes of computation time per solution.

3.2. Graph Neural Networks methodology

3.2.1. Introduction of GNNs

Graph Neural Networks (GNNs) provide a powerful framework for processing relational data by iteratively aggregating neighborhood information. This approach is particularly well suited for sliding puzzles because each puzzle state naturally forms a grid graph $G = (V, E)$, where tiles correspond to nodes (V) and adjacency relationships form edges (E).

The core function of GNNs lies in their message-passing mechanism, which directly simulates the dynamics of the movement of tiles in the puzzle. This mechanism allows the network to learn complex patterns in puzzle states automatically, such as relative positional relationships and conflict situations between tiles, without requiring manually designed heuristic rules.

The message passing process operates through three basic functions that work together to analyze the state of the puzzle.

Message Construction (MSG) computes the interaction potential between adjacent tiles. For PuzzleGNN, this is implemented as a neural network layer that combines information from neighboring tiles and their connection direction. This function can identify potential tile movements and local conflicts.

Aggregation (AGG) synthesizes neighborhood information by combining messages from all adjacent tiles. We use mean aggregation to identify common conflict patterns like linear conflicts (tiles in correct row/column but reversed order). This step creates a summary of the local neighborhood that captures essential constraints.

DOI: 10.54254/2753-8818/2025.DL29641

Update (UPDATE) integrates new information with the tile's previous state using a Gated Recurrent Unit (GRU). This gating mechanism can maintain memory of distant tile dependencies, allowing the model to preserve information about tiles that are far apart but need coordinated movement.

3.2.2. Why GNNs solve puzzles effectively

PuzzleGNN overcomes traditional heuristic limitations through several key mechanisms. Relational Reasoning enables the model to capture dependencies within neighborhoods of tiles. Each layer in the GNN extends the model's "vision" to more distant tiles. This hierarchical understanding allows the model to recognize complex patterns that distance metrics miss.

Size Invariance is implemented by coordinate normalization, where the positions are scaled relative to the size of the board. This proportional scaling allows the same model to handle different puzzle dimensions without retraining. For example, a corner tile in any size puzzle is represented similarly, but the right first tile is represented differently, enabling knowledge transfer across scales.

3.3. System implementation

3.3.1. Data processing pipeline

Algorithm 1. Graph representation construction

	Graph representation construction
1:	Input: Puzzle state S (size n × n)
2:	Output: Graph G = (V, E, X)
3:	
4:	Step 1: Create nodes for each tile position
5:	for each position (r, c) in the puzzle grid do
6:	Create node $v_{rc} \in V$
7:	Store normalized features(X):
8:	• Current row: r/(n − 1)
9:	• Current column: c/(n − 1)
10:	• Target row: g_r/(n − 1) (from goal position)
11:	• Target column: g_c/(n − 1)
12:	• Tile value: tile/n^2
13:	end for
14:	
15:	Step 2: Connect adjacent tiles
16:	for each node $v_{rc} \in V$ do
17:	for each direction (dr, dc) in {up, down, left, right} do
18:	Calculate neighbor position $(r', c') = (r + dr, c + dc)$

DOI: 10.54254/2753-8818/2025.DL29641

Algorithm 1. (continued)

19:	if position (r', c') exists within grid then
20:	Add undirected edge between v_{rc} and $v_{r'c'}$ to E
21:	end if
22:	end for
23:	end for

Algorithm 2. Size adaptation and processing

	Size adaptation and processing
1:	Input: Graph G = (V, E, X), puzzle size n
2:	Output: Predicted optimal step count $\hat{y}$
3:	
4:	Encode size information
5:	Look up size embedding e_n for dimension n
6:	Transform e_n to match node feature dimension
7:	Add size embedding to each node's initial features
8:	
9:	Step 2: Message passing (repeat 4 times)
10:	for each node $v_i \in V$ do
11:	Collect information from neighbor nodes
12:	Compute combined neighborhood message
13:	Update node state using GRU mechanism:
14:	• Maintain important long-range dependencies
15:	• Filter irrelevant local information
16:	end for
17:	
18:	Step 3: Global aggregation and prediction
19:	Compute graph-level representation by averaging all node states
20:	Process through neural network layers
21:	Output predicted optimal move count $\hat{y}$

DOI: 10.54254/2753-8818/2025.DL29641

3.3.2. Training procedure

Algorithm 3. Model training workflow

	Model training workflow
1:	Initialize: Model parameters θ
2:	
3:	while training not converged do
4:	Model prediction and learning:
5:	Make predictions $\hat{y}$ = PuzzleGNN$_\theta$(G)
6:	Compute Huber loss $\mathscr{L}$ = $\sum$loss$(\hat{y}, y^*)$
7:	Calculate parameter gradients $\nabla_\theta \mathscr{L}$
8:	Limit gradient magnitude to prevent instability
9:	Update parameters using AdamW optimizer
10:	end while

Key Implementation Details:
• Size Embedding: Maps discrete sizes to continuous vectors (e.g., 3 → [0.2, -0.5, ..., 0.7])
• GRU Update: Balances new information with previous state:
New state = update gate × candidate + (1 − update gate) × previous
• Huber Loss: Robust to prediction outliers:

$$\mathscr{L} = \begin{cases} 0.5(\hat{y} - y^*)^2 & \text{if } |\hat{y} - y^*| \le 1 \\ |\hat{y} - y^*| - 0.5 & \text{otherwise} \end{cases}$$

4. Results

4.1. Experimental results and analysis

4.1.1. Training process and performance

Model training employed Huber loss and AdamW optimizer, with key strategies including gradient clipping and early stopping. Training results show rapid convergence, reaching optimal performance in just 3 epochs:

Table 1. Model training performance

Epoch	Train Loss	Val Loss	Val MAE	Val R^2
1	2.5670	1.7373	2.17	0.4710
2	1.3839	0.9615	1.34	0.7463
3	0.7902	0.5850	0.7313	0.9163
4	0.6241	0.8787	1.26	0.7924

DOI: 10.54254/2753-8818/2025.DL29641

As shown in Table 1, the model reaches optimal performance at the third epoch (MAE=0.73 steps, R^2=0.9163), but showing signs of overfitting after that. This rapid convergence demonstrates the model's efficiency in learning the mapping between puzzle states and optimal solution lengths.

4.1.2. Cross-size performance evaluation

Evaluation across puzzle sizes demonstrated consistently strong performance within the training distribution (Table 2). All sizes achieved $R^2 > 0.85$, with 6×6 puzzles showing exceptional accuracy (MAE=0.62, R^2=0.94). This peak performance at 6x6 puzzles stems from several complementary factors:

First, 6×6 puzzles (36 tiles) provide an optimal complexity level where the GNN's four-layer receptive field covers most of tile interactions—sufficient for meaningful pattern recognition without excessive information dilution. Second, the training dataset contained the most diverse state configurations (8,050 samples) within the model's effective prediction range.

Even for the 7×7 puzzle, inference latency remains below 75 milliseconds, a 100-3000x improvement over traditional A* search. This efficiency stems from parallel graph processing instead of sequential state exploration, and the use of parametric heuristics instead of computationally intensive heuristic functions.

Table 2. Performance evaluation on trained sizes

Size	Samples	MAE	R2	Inference Time(ms)
3×3	7990	0.7444	0.8568	15
4×4	7997	0.7313	0.9037	22
5×5	7994	0.7795	0.9183	35
6×6	8050	0.6216	0.9396	52
7×7	7969	0.7804	0.9191	75

4.1.3. Generalization capability assessment

Performance decreases dramatically on untrained 8×8 puzzles:

Table 3. Performance on unseen size (8×8)

Size	Samples	MAE	R2
8×8	1000	7.3176	– 2.4472

The negative value of R^2 indicates that the predictions are worse than simply using the target mean, which means that the current architecture cannot effectively generalize to puzzle sizes beyond the training distribution.

The reasons of the fatal performance drop on 8×8 puzzles is from fundamental limitations in both model architecture and training methodology.

First, the 8×8 configuration (64 nodes) represents a 31% increase in node count over the largest trained size (7×7, 49 nodes). This exceeds the generalization capacity of the size-adapter module, which lacks mechanisms to extrapolate to unseen graph sizes.

Second, The 8×8 state space ($64! \approx 1.3\times10^{89}$) is 10^{40} times larger than 7×7 ($49! \approx 6.1 \times 10^{62}$). Our training data covered less than 10^{-60}% of this space, resulting in massive extrapolation errors.

DOI: 10.54254/2753-8818/2025.DL29641

4.1.4. Long-path prediction analysis

Solution paths exceeding 30 steps exhibited dramatically increased errors due to three compounding factors: First, training data contained few long-path examples (< 5%′s samples > 20steps), creating representation gaps in complex state regions. Second, the model's limited compositional capacity hindered hierarchical reasoning over extended action sequences. These observations corroborate theoretical limitations in neural planning architectures [12,13].

4.2. Model strengths and limitations

4.2.1. Innovative advantages

The PuzzleGNN framework demonstrates three significant innovations that advance neural heuristic design. First, it achieves a computational efficiency breakthrough with inference speeds of 22-75ms across trained puzzle sizes, representing a 100-3000× acceleration over traditional A* search. This performance gain comes from three interconnected factors: the replacement of sequential state exploration with parallel graph processing, utilization of fixed computational graph structures, and implementation of parametric heuristics that eliminate computation-intensive heuristic functions. Also, the size adapter module enables multi-size unified processing, allowing a single model architecture to handle puzzles ranging from 3×3 to 7×7. This design reduces deployment complexity and maintains size-specific performance characteristics.

4.2.2. Core limitations

Although PuzzleGNN has these advances, three fundamental limitations constrain current applicability. The most significant is poor size generalization ability, shown by the fatal performance collapse on untrained 8×8 puzzles (R^2=-2.45). This failure comes from the hard-coded constraints in the size embedding space and insufficient training data coverage for extrapolation.

The second limitation concerns inaccurate long-path prediction, where errors increase significantly for solutions exceeding 50 steps. This comes from three compounding factors: sparse representation of long-path samples in training data, the limited receptive field of 4-layer graph convolutions, and insufficient modeling of complex state interactions in deep solution paths.

Finally, the approach exhibits strong data dependency requiring extensive precomputed optimal solutions via IDA*, creating scalability bottlenecks. This dependency fundamentally limits application to larger puzzles where generating optimal solutions becomes computationally prohibitive.

4.3. Future directions

The PuzzleGNN model establishes neural heuristics as effective for known puzzle sizes while revealing critical generalization boundaries.

Longer-term foundational work should establish theoretical boundaries for neural heuristics. This includes formalizing generalization limits through graph spectral analysis, developing admissibility guarantees via Lipschitz continuity constraints [14], and optimizing state space coverage through quasi-Monte Carlo sampling techniques. These directions address core challenges in scaling neural heuristics to more complex combinatorial domains.

DOI: 10.54254/2753-8818/2025.DL29641

5. Conclusion

This paper presents PuzzleGNN, a framework for sliding-tile puzzles that represents board states as graphs and employs a size adapter so a single model can predict across puzzle sizes from 3×3 to 7×7. Empirical results show that PuzzleGNN produces high-quality, low-latency estimates of optimal move counts within the training size range, demonstrating the GNN's ability to capture local and global structure and to generate parallelizable heuristic estimates.

However, PuzzleGNN's performance degrades sharply when extrapolated to unseen larger sizes (e.g., 8×8), indicating that the current size-adaptation and feature design do not guarantee robust cross-size generalization. In addition, long-solution instances are underrepresented in the training data, which increases error for long-horizon predictions. The model's reliance on labeled optimal solutions constrains scalability to large puzzle sizes where obtaining optimal labels is costly.

To address these issues, we propose several directions: (1) incorporate sizeinvariant relative position features and graph normalization to improve cross-size consistency; (2) apply curriculum learning that systematically increases solution complexity, hierarchical subgoal decomposition, or multi-scale graph representations to better handle long solution paths; and (3) explore neural–symbolic hybrid heuristics to retain optimality guarantees while leveraging learned speedups. The establishment of theoretical boundaries for neural heuristics facilitate the scaling of neural heuristics to more complex combinatorial domains.

In summary, PuzzleGNN demonstrates the great potential of applying graph neural networks to learned heuristics for combinatorial optimization: it achieves accurate, highly efficient move-count predictions on seen sizes and provides a clear roadmap for improving generalization, long-horizon modeling, and integration with provably admissible search methods. Although substantive work remains to improve generalization, long-path handling, and data scalability, combining representation learning with traditional search presents a promising path for tackling large-scale combinatorial search problems.

References

[1] Korf, R. E. (1985). Iterative-Deepening A*. Artificial Intelligence, 27(1), 97–109.
[2] Hansson, O., Mayer, A., & Valtorta, M. (1992). A new result on the complexity of heuristic estimates for the A* algorithm. Artificial Intelligence, 55(1), 129–143.
[3] Culberson, J. C., & Schaeffer, J. (1998). Pattern databases. Computational Intelligence, 14(3), 318–334.
[4] Felner, A., Korf, R. E., & Hanan, S. (2004). Additive pattern database heuristics. Journal of Artificial Intelligence Research, 22, 279–318.
[5] Zhou, R., & Hansen, E. A. (2012). Parallel structured duplicate detection for A* graph search. AAAI Conference on Artificial Intelligence.
[6] Agostinelli, F., McAleer, S., Shmakov, A., & Baldi, P. (2019). Solving the Rubik's Cube with Deep Reinforcement Learning and Search. Nature Machine Intelligence, 1(8), 356–363.
[7] Kipf, T. N., & Welling, M. (2017). Semi-Supervised Classification with Graph Convolutional Networks. International Conference on Learning Representations (ICLR).
[8] Hart, P. E., Nilsson, N. J., & Raphael, B. (1968). A Formal Basis for the Heuristic Determination of Minimum Cost Paths. IEEE Transactions on Systems Science and Cybernetics, 4(2), 100–107.
[9] Korf, R. E., & Reid, M. (1997). Complexity Analysis of Admissible Heuristic Search. AAAI/IAAI, 305–310.
[10] Thayer, J. T., Dionne, A. J., & Ruml, W. (2009). Learning Inadmissible Heuristics During Search. ICAPS.
[11] Yang, F., Culberson, J., Holte, R., Zahavi, U., & Felner, A. (2011). A general theory of additive state space abstractions. Journal of Artificial Intelligence Research, 32, 631–662.
[12] Arora, S., et al. (2019). Theoretical limitations of self-attention in neural sequence models. Transactions of the Association for Computational Linguistics.
[13] Bershatsky, D., et al. (2022). Generalization in graph neural networks: Improved PAC-Bayesian bounds on graph diffusion. NeurIPS.

DOI: 10.54254/2753-8818/2025.DL29641

[14] Liu, J., et al. (2020). Certified robustness of graph neural networks against adversarial structural perturbation. KDD.