GABAR: Graph Attention-Based Action Ranking for Relational Policy Learning

Rajesh Mangannavar, Stefan Lee, Alan Fern, Prasad Tadepalli

Oregon State University, Corvallis, OR 97330, USA
{mangannr, leestef, alan.fern, prasad.tadepalli}@oregonstate.edu

Abstract

We propose a novel approach to learn relational policies for classical planning based on learning to rank actions. We introduce a new graph representation that explicitly captures action information and propose a Graph Neural Network architecture augmented with Gated Recurrent Units (GRUs) to learn action rankings. Our model is trained on small problem instances and generalizes to significantly larger instances where traditional planning becomes computationally expensive. Experimental results across standard planning benchmarks demonstrate that our action-ranking approach achieves generalization to significantly larger problems than those used in training.

Introduction

Classical planning tackles the problem of finding action sequences to achieve goals in deterministic environments. While traditional planners can solve small problems optimally using search and heuristics, they often struggle with scalability. This has motivated research into learning general relational policies from small solved problems, which can then be applied to significantly larger instances (Ståhlberg, Bonet, and Geffner 2022a). The key insight behind learning-based planning is that optimal solutions to small problems often reveal patterns that generalize to bigger problems. For example, in Blocks World, solutions to 4 block problems can teach a policy to stack blocks bottom-up that generalizes to problems with 20 or more blocks. The advantage of relational learning is its ability to capture compositional structure. This compositionality also enables strong generalization (Zambaldi et al. 2018), (Fern, Yoon, and Givan 2003) (Džeroski, De Raedt, and Driessens 2001).

Learning approaches in automated planning have traditionally focused on learning heuristic functions to guide search algorithms. These methods typically learn a value function that estimates the distance to the goal and integrate it within classical search algorithms like A* or greedy best-first search. The learned heuristics help focus the search but still require explicit search during plan execution.

Recently, several neural approaches have been proposed for learning domain-specific heuristics. (Toyer et al. 2020) learn neural heuristics that generalize across problem instances but still rely on search for planning. Similarly, (Shen, Trevizan, and Thiébaux 2020) use hypergraph networks to learn heuristics that can transfer across problems. Other approaches that learn value functions for planning include (Chen, Thiébaux, and Trevizan 2023), (Chen, Thiébaux, and Trevizan 2024). While these approaches show promise in learning useful search guidance, they suffer from the inherent computational overhead of the search process during planning.

An alternative line of work focuses on learning policies that can directly select actions without search. Value-based methods like (Ståhlberg, Bonet, and Geffner 2022b), (Ståhlberg, Bonet, and Geffner 2022a) learn a value function that induces a greedy policy by selecting actions leading to states with minimum estimated cost-to-go. While effective, these approaches face two key limitations: First, value functions are not only quite complex and challenging to learn, but also are unnecessary for action selection. Second, in domains where optimal planning is NP-hard (like the Blocks World), optimal value functions are not easily generalizable to larger problem sizes (Gupta and Nau 1992).

Learning to Rank approaches have shown promise in planning domains but have primarily focused on ranking states rather than actions. (Garrett, Kaelbling, and Lozano-Pérez 2016) pioneered this direction by using RankSVM to learn state rankings using hand-crafted features. More recently, (Chrestien et al. 2024) and (Hao et al. 2024) demonstrated that learning to rank states can be more effective than learning precise heuristic values, as the relative ordering of states is sufficient for guiding search. However, these approaches still fundamentally rely on search during execution, inheriting signficant computational overhead.

Our work is inspired by the effectiveness of graph neural networks (GNNs) to represent and learn general relational policies such as (Ståhlberg, Bonet, and Geffner 2022b, 2023, 2024; Chen, Thiébaux, and Trevizan 2024). We introduce a novel architecture GABAR (Graph Attention-Based Action Ranking), which directly learns to rank actions rather than estimating value functions. GABAR introduces (1) an action-centric graph representation that explicitly captures how objects participate in actions and (2) a GNN architecture with Gated Recurrent Units (GRUs) that learns how object representations update based on their role in actions.

One of our key insights is that ranking actions that are applicable in the same state often turns out to be easier and more generalizable than ranking states by their distances to the goals. Through experiments on standard benchmarks, we show GABAR achieves generalization to significantly larger problems than those used for training.

Problem Setup

Classical planning deals with finding a sequence of actions that transform an initial state into a goal state. A classical planning problem is represented as a pair $P = \langle D, I \rangle$, where D represents a first-order domain and I contains instance-specific information. The domain D consists of a set of predicate symbols $\mathcal{P}$ with associated arities and a set of action schemas $\mathcal{A}$. Each action schema $a \in \mathcal{A}$ is defined by a set of parameters $\Delta(a)$ representing variables that can be instantiated, preconditions $pre(a)$, add effects $add(a)$, and delete effects $del(a)$. The instance information I is a tuple $\langle O, s_0, G \rangle$ where O is a finite set of objects, s_0 is the initial state represented as a set of ground atoms $p(o_1, ..., o_k)$ where $p \in \mathcal{P}$ and $o_i \in O$, and G is the goal condition also represented as a set of ground atoms.

A state s is a set of ground atoms that are true in that state. An action schema can be grounded by substituting its parameters with objects from O. A ground action a is applicable in state s if $pre(a) \subseteq s$, and results in successor state $s' = (s \setminus del(a)) \cup add(a)$. A solution or plan is a sequence of applicable ground actions that transform the initial state s_0 into a state satisfying the goal condition G. A relational policy maps a problem state to an action. The current paper addresses the following problem. Given a domain D and a set of training instances of different sizes and their solutions, learn a relational policy that leads to efficient solutions for larger test instances from the same domain.

Graph Attention-Based Action Ranking

Learning general policies for classical planning domains requires effectively processing variable-sized states and selecting appropriate actions that generalize across problem instances. We introduce GABAR (Graph Attention-based Decoder for Action Ranking), a novel architecture that directly learns to rank and select actions rather than learning value functions. GABAR consists of four key components: (1) a graph-based state representation that explicitly captures grounded action information, (2) a neural encoder that processes this rich representation, and (3) a GRU-based decoder that sequentially constructs complete actions. This section details each component and explains how they work together to enable effective action selection.

System Overview

Given a planning instance, **I** in PDDL format (Fox and Long 2003), along with the set of ground actions, GABAR operates by first converting this to a graph structure that makes explicit the relationships between objects, predicates, and potential actions. This graph is then processed through our neural architecture to rank actions(as described in Fig 1). Then, the highest-ranked applicable action is executed to reach the next state **I'**. This process is repeated until a goal state is reached. To ensure the execution terminates, the system maintains a history of visited states and avoids actions that would lead to previously visited states. The execution continues until either reaching a goal state or terminating if no unvisited successor states are available or if the maximum execution length (1000 in our experiments) is exceeded.

Graph Representation

We introduce a novel graph representation for classical planning tasks that captures the structural relationships between objects, predicates, and actions and the semantic information needed for learning action ranking effectively. Our representation $G = (V, E, X, R)$ consists of a set of nodes V, edges E, node features X, and edge features R.

The node set $V = O \cup P \cup A \cup \{g\}$ where O, P,and A represent sets of domain objects, grounded predicates, and action schemas respectively and g is a global node that aggregates graph-level information. The edge set $E = E_{\text{pred}} \cup E_{\text{act}}$, where E_{pred} is the set of edges between predicates and their argument objects and similarly E_{act} is the set of edges between action schemas and their argument objects.

Node Features The node feature function $X : V \to \mathbb{R}^d$ maps each node to a feature vector that encodes type and semantic information. The feature vector is constructed by concatenating several one-hot encoded segments:

For any node $v \in V$: $X(v) = [X_{\text{type}}(v) \parallel X_{\text{act}}(v) \parallel X_{\text{pred}}(v) \parallel X_{\text{obj}}(v)]$, where:

- $X_{\text{type}} \in \{0,1\}^3$: One-hot encoding of node type (object, predicate, or action)
- $X_{\text{act}} \in \{0,1\}^{|A|}$: One-hot encoding of action type (if v is an action node)
- $X_{\text{pred}} \in \{0,1\}^{2|P|}$: Encoding for predicates, where first $|P|$ bits indicate predicate type and next $|P|$ bits indicate goal predicates ((if v is a predicate node))
- $X_{\text{obj}} \in \{0,1\}^{|T|}$: One-hot encoding of object type (if v is an object node), where T is the set of object types

Edge Features The edge feature function $R : E \to \mathbb{R}^k$ maps each edge to a feature vector encoding edge type and role information:

For any edge $e \in E$: $R(e) = [R_{\text{type}}(e) \parallel R_{\text{pred}}(e) \parallel R_{\text{act}}(e)]$, where:

- $R_{\text{type}} \in \{0,1\}^2$: One-hot encoding of edge type (predicate-object or action-object)
- $R_{\text{pred}} \in \{0,1\}^m$: For predicate-object edges, one-hot encoding of argument position (m is max predicate arity)
- $R_{\text{act}} \in \{0,1\}^{(m+|P|)}$: For action-object edges, concatenation of:
  - One-hot encoding of parameter position in action schema (m bits)
  - Binary vector indicating which predicates are satisfied by the object in the grounded action ($|P|$ bits)

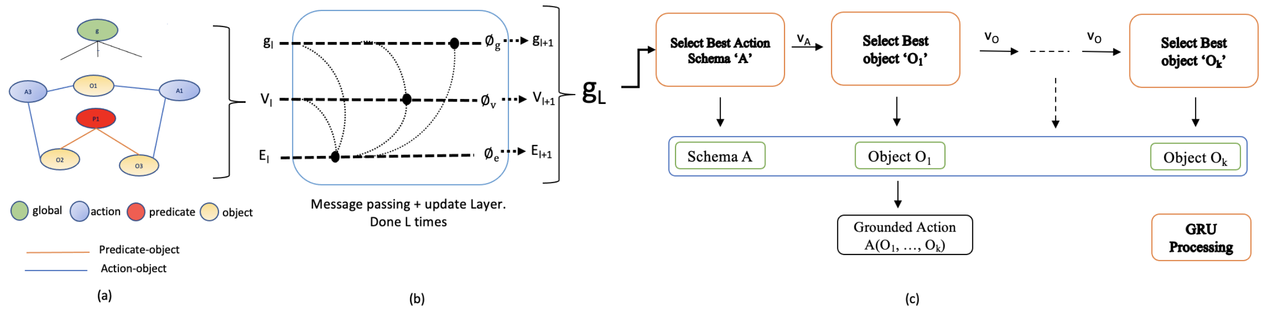

Figure 1: GABAR's architecture for action extraction. (a) Graph representation: The input PDDL problem is converted into a graph with four types of nodes (predicate, object, action schema, and global) connected by predicate-object and action-object edges that encode state and grounded action information. (b) GNN encoder: Processes the graph through L rounds of message passing where edge, node, and global representations are sequentially updated based on their interactions. (c) Action decoder: Uses the final global embedding to construct a grounded action through a GRU-based decoder sequentially - first selecting an action schema, then iteratively choosing objects for each parameter position until a complete grounded action is formed.

Global Features The global node g is initialized with a zero vector in $\mathbb{R}^h$ where h is the chosen hidden representation dimension. This node can be used to aggregate and propagate graph-level information during message passing.

This graph representation captures both the structural and semantic information necessary for learning planning heuristics while maintaining a bounded feature dimension independent of problem size. The node features encode type and semantic information, while the edge features capture the relationships between objects, predicates, and actions in the planning domain.

Neural Architecture

Our neural architecture processes this graph representation through multiple components designed to handle the challenges of processing variable-sized inputs, capturing long-range dependencies between objects and actions, and making sequential decisions to construct complete actions. We detail each component below:

Graph Neural Network Encoder Graph Neural Networks (GNNs) (Scarselli et al. 2008) are particularly well-suited for encoding states in planning problems as they can naturally process relational structures while being invariant to permutations and handling varying input sizes. This allows them to learn patterns that generalize across different problem instances within the same domain, regardless of the number of objects involved. The key insight is that planning states are inherently relational - objects interact through predicates and actions - and GNNs can capture these relationships through message passing between nodes and edges.

Our GNN encoder processes the input graph through L layers of message passing between nodes, edges, and a global node. At each layer l, the updates proceed as follows:

First, edge embeddings are updated based on their incident nodes and the global context:

$$\mathbf{e}_{ij}^{l+1} = \phi_e([\mathbf{e}_{ij}^l; \mathbf{v}_i^l; \mathbf{v}_j^l; \mathbf{g}^l]) \tag{1}$$

where $\mathbf{e}_{ij}^l$ is the embedding of edge (i, j) at layer l, $\mathbf{v}_i^l$ and $\mathbf{v}_j^l$ are the embeddings of its incident nodes, $\mathbf{g}^l$ is the global node embedding, $[;]$ denotes concatenation, and ϕ_e is a learnable neural network.

Next, node embeddings are updated using information from their connected edges and the global context:

$$\mathbf{v}_i^{l+1} = \phi_v([\mathbf{v}_i^l; \text{AGG}(\{\mathbf{e}_{ij}^{l+1} | j \in \mathcal{N}(i)\}); \mathbf{g}^l]) \tag{2}$$

Where AGG is a permutation-invariant aggregation function (we use attention weighted sum), $\mathcal{N}(i)$ denotes the neighbors of node i, and ϕ_h is a learnable network.

Finally, the global node is updated by aggregating information from all nodes and edges:

$$\mathbf{g}^{l+1} = \phi_g([\mathbf{g}^l; \text{AGG}(\{\mathbf{v}_i^{l+1} | i \in \mathcal{V}\}); \text{AGG}(\{\mathbf{e}_{ij}^{l+1} | (i, j) \in \mathcal{E}\})]) \tag{3}$$

where ϕ_g is a learnable network that combines the previous global state with aggregated node and edge information.

This architecture makes several key design choices motivated by the planning domain. First, we explicitly model and update edge representations because edges in our graph capture crucial information about action applicability. This edge information helps guide the model toward selecting valid and effective actions. Second, we include a global node that can rapidly propagate information across the graph. This is particularly important as planning problems scale up - without the global node, information would need to flow through many message-passing steps to reach distant parts of the graph. The global node acts as a shortcut, allowing the model to maintain a comprehensive view of the planning state even as the number of objects and relations grows. After L rounds of message passing have been performed, the final global node embedding g^{l+1} captures the relevant planning context needed for action selection.

GRU-based Action Decoder

One of the key novelties of our architecture is a GRU-based action decoder that supports actions with an arbitrary number of parameters. The decoder transforms the final global graph representation g_{l+1} into a sequence of parameter decisions that fully specify an action. This sequential process is managed by a Gated Recurrent Unit (GRU) that maintains a hidden state h_i representing the partial action construction:

$$v_1 = \text{GRU}(g_{v+1}, 0) \quad \text{// Initialize with global state}$$
$$v_{i+1} = \text{GRU}(v_i, e_i) \quad \text{// Update with each decision } e_i$$

The GRU architecture allows the decoder to maintain relevant context while making a sequence of interdependent decisions. This is crucial for ensuring that selected objects form valid and effective action instantiations.

During training, action selection proceeds greedily in two phases: First, the decoder selects an action schema using the initial hidden state:

$$\text{score}(a) = \text{MLP}([v_1; a]) \quad \text{for each action schema } a$$
$$a^* = \arg\max_{a \in A} \text{softmax}(\text{score}(a))$$

Then, it iteratively selects the action parameters:

$$\text{score}(o) = \text{MLP}([v_i; o]) \quad \text{for each candidate object } o$$
$$o^* = \arg\max_{o \in O} \text{softmax}(\text{score}(o))$$
$$v_{i+1} = \text{GRU}(v_i, \text{embed}(o^*))$$

Since greedy parameter selection is often too myopic, we employ beam search to explore multiple choices in parallel. For a beam width k, at each step, we maintain the k highest-scoring partial sequences. The final output is a ranked list of k action groundings $(a, o_1, \ldots, o_n)$ along with their accumulated scores. This aligns with our goal of learning to rank actions rather than just selecting individual best choices - while training optimizes for selecting the optimal action, the learned model provides a ranking of top k grounded actions during execution. The planner can then use this ranking to make more informed decisions, such as incorporating additional criteria such as cycle avoidance.

Data Generation and Training

For each planning domain, we generate training data by solving a set of small problem instances using an optimal planner. Each training example consists of a planning state s, goal specification G, and the first action a^* from the optimal plan from s to G. For states with multiple optimal actions, we randomly select one to avoid biasing the model.

The state-goal pairs are converted into our graph representation $G = (V, E, X, R)$ as described in the graph representation section. For each action a^* in the training data, we create supervision signals in the form of:

- y_a: A one-hot vector over the action schema space indicating the correct action type
- $y_o = \{y_{o1}, ..., y_{ok}\}$: A sequence of k one-hot vectors over the object space, where k is the maximum number of parameters any action can take, indicating the correct objects for each parameter position

For action schema selection: the model needs to learn to assign the highest score to the correct action schema among all possible schemas. For object selection: for each parameter position, the model needs to learn to assign the highest score to the correct object among all candidate objects. This is done using the following loss function.

Loss Function Given a training instance (G, y_a, y_o), GABAR computes action scores s_a for all possible action schemas and object scores s_{o_i} for each parameter position i. The total loss is computed as $L = L_{action} + L_{objects}$, where, $L_{objects}$ is the sum of cross-entropy losses between softmax(s_{o_i}) and y_{o_i} for each parameter position i.

Training Procedure We train the model using the Adam optimizer with a learning rate of 0.0005, 9 rounds of GNN, and batch size of 16. Training proceeds for a maximum of 500 epochs, and we select the model checkpoint that achieves the lowest loss on the validation set for evaluation.

Experiments

We evaluate GABAR's performance across a diverse set of classical planning domains. Our experiments aim to assess both the quality of learned policies and their ability to generalize to significantly larger problems than those in training.

Domains. We selected six standard planning domains that present different types of structural complexity and scaling dimensions.

Blocks World involves manipulating blocks to achieve specific tower configurations. The domain's complexity scales with the number of blocks (6-9 blocks for training/validation, 10-40 blocks for testing).

Gripper requires a robot with two grippers to transport balls between rooms. The domain scales primarily with the number of balls to be moved (5-17 balls for training/validation, up to 100 balls for testing).

Miconic involves controlling an elevator to transport passengers between floors. The domain complexity increases along two dimensions: the number of passengers (1-10 for training/validation, 20-100 for testing) and the number of floors (2-20 for training/validation, 11-30 for testing).

Logistics involves transporting packages between locations using trucks (for intra-city transport) and airplanes (for intercity transport). The domain scales with both the number of cities (4-8 for training/validation, 15-30 for testing) and packages (3-10 for training/validation, 9-24 for testing).

Visitall requires an agent to visit all cells in a grid. The domain scales with grid size (9-49 cells for training/validation, up to 400 cells for testing - testing problems 8 times larger than the training dataset).

Grid involves navigating through a grid where certain doors are locked and require specific keys to open. The number of locks (3) and keys (5) remain the same across training and testing while varying the size of the grid (7×9 for training/validation to 11×14 for testing - a 150% increase in cells).

Evaluation Metrics. We evaluate GABAR using three primary metrics:

- **Coverage (Cov):** The percentage of test instances successfully solved within a 1000-step limit.

Table 1: Performance of both GABAR and GABAR-G methods across domains(number of problems shown in parenthesis(#)). Coverage indicates the percentage of test instances solved within 1000 steps. Plan lengths are reported only for solved instances.

	GABAR				GABAR-G			
Domain	Cov.(%) ↑	Mean PL ↓	Median PL ↓	PQR ↑	Cov.(%) ↑	Mean PL ↓	Median PL ↓	PQR ↑
Blocks World (100)	**100**	53.1	46	**1.42**	84	52.6	54.7	1.1
Gripper (100)	**100**	147.1	133	**0.99**	89	146.7	137	0.95
Miconic (50)	**100**	181.7	181	**0.97**	75	192.4	189	0.87
Logistics (50)	**76**	155.2	132	**0.65**	12	238.5	220	0.33
Visitall (50)	**88**	207.9	190	**1.07**	63	205.7	183	0.86
Grid (50)	**96**	72.1	62	**0.81**	68	70.2	68	0.73

- **Plan Length (PL):** Measured through both mean and median plan lengths across solved instances.
- **Plan Quality Ratio(PQR):** Ratio of plan length produced by Fast Downward (FD) planner to the plan length produced by the learned policy. FD is run with `fd-lama-first` setting. We chose this satisficing configuration over optimal planners since optimal planners fail to solve most test problems within a reasonable time. While this means we cannot guarantee the optimality of the reference plans, it provides a practical baseline for assessing solution quality across our test suite.

High coverage on larger instances demonstrates the model's ability to learn robust action selection strategies, while plan length and quality metrics reveal whether these strategies remain efficient as problems scale up.

Ablation: To understand the importance of different architectural components, we conduct an ablation study focusing on the design choice of using global nodes in the graph **(GABAR-G)**. This variant of GABAR is without the global node, which helps assess its role in information propagation and helps quantify how the global node affects coverage, plan quality, and length.

Results and Discussions

Table 1 shows the results of our learning system across 6 domains. We can see that the proposed method (OURS) generalizes very well on three domains - Blocksworld, Gripper, and Miconic. We can also see that our method generalizes well in the other three domains. This result, while not as good as others, shows an important capability of the proposed system - the ability to handle complex domains.

Both Grid and Logistics are more complex domains (Ståhlberg, Bonet, and Geffner 2022b), as expressing properties of the environment requires multiple relations (Ex, the feature expressing that a package is in a city while possibly within a vehicle needs multiple relations to express). We can see that the proposed method can learn these complex features that are required for solving the planning problem as well as generalize to larger problems in these settings (success rate of 76% and 96% in them). While the success rate of Visitall is not as impressive as Blocksworld, the generalization it achieves is impressive - it is solving problems up to eight times larger than what it trained on with an 88% success rate.

The ablation study comparing GABAR with and without the global node (GABAR vs. GABAR-G) reveals that removing the global node leads to degradation in performance.

In domains that primarily require local reasoning, like Blocks World, the impact is moderate but still notable - coverage drops from 100% to 84%, and the plan quality ratio decreases from 1.42 to 1.1. This suggests that while local messages passing between objects and actions can capture basic patterns, the global node helps maintain a broader context. This degradation is even more dramatic in Logistics, where coverage plummets from 76% to just 12%. The stark difference can be attributed to the need for long-range planning, as logistics requires orchestrating multiple vehicles across different cities.

Plan quality also suffers significantly without the global node. In Logistics, the plan quality ratio drops from 0.65 to 0.33, indicating that solutions become nearly twice as inefficient. This degradation in quality suggests that the global node plays a crucial role in helping the model learn strategic action selection rather than just locally reasonable choices.

These findings validate our architectural choice of including a global node in GABAR. The global node proves essential for facilitating long-range information sharing that helps coordinate actions across distant entities.

Conclusion and Future Work

We presented **GABAR**, a novel graph-based architecture for learning generalized policies in classical planning through action ranking. Our key contributions include (1) an action-centric graph representation that explicitly captures action-object relationships, (2) a GNN architecture augmented with global nodes and GRUs for effective information propagation, and (3) a sequential decoder that learns to construct complete grounded actions.

Our experimental results demonstrate strong generalization capabilities across multiple planning domains. GABAR achieves great coverage on Blocksworld, Gripper, Visitall, and Miconic domains when scaling to problems that are many times larger than those used in training. The architecture also shows promising results in more complex domains like Grid and Logistics. The ablation study removing global nodes highlights their critical role in enabling effective information propagation.

The primary scaling challenge lies in the graph representation's growth rate, particularly for domains with high-arity actions or predicates. Future work could explore more compact representations while maintaining expressiveness and investigate ways for pruning irrelevant actions in the graph.

References

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.

Chrestien, L.; Edelkamp, S.; Komenda, A.; and Pevny, T. 2024. Optimize planning heuristics to rank, not to estimate cost-to-goal. *Advances in Neural Information Processing Systems*, 36.

Džeroski, S.; De Raedt, L.; and Driessens, K. 2001. Relational reinforcement learning. *Machine learning*, 43: 7–52.

Fern, A.; Yoon, S.; and Givan, R. 2003. Approximate policy iteration with a policy language bias. *Advances in neural information processing systems*, 16.

Fox, M.; and Long, D. 2003. PDDL2. 1: An extension to PDDL for expressing temporal planning domains. *Journal of artificial intelligence research*, 20: 61–124.

Garrett, C. R.; Kaelbling, L. P.; and Lozano-Pérez, T. 2016. Learning to rank for synthesizing planning heuristics. *arXiv preprint arXiv:1608.01302*.

Gupta, N.; and Nau, D. S. 1992. On the complexity of blocks-world planning. *Artificial intelligence*, 56(2-3): 223–254.

Hao, M.; Trevizan, F.; Thiébaux, S.; Ferber, P.; and Hoffmann, J. 2024. Learned Pairwise Rankings for Greedy Best-First Search. In *Proc. ICAPS Workshop on Reliable Data-Driven Planning and Scheduling*.

Scarselli, F.; Gori, M.; Tsoi, A. C.; Hagenbuchner, M.; and Monfardini, G. 2008. The graph neural network model. *IEEE transactions on neural networks*, 20(1): 61–80.

Shen, W.; Trevizan, F.; and Thiébaux, S. 2020. Learning domain-independent planning heuristics with hypergraph networks. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 30, 574–584.

Ståhlberg, S.; Bonet, B.; and Geffner, H. 2022a. Learning general optimal policies with graph neural networks: Expressive power, transparency, and limits. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 32, 629–637.

Ståhlberg, S.; Bonet, B.; and Geffner, H. 2022b. Learning generalized policies without supervision using gnns. *arXiv preprint arXiv:2205.06002*.

Ståhlberg, S.; Bonet, B.; and Geffner, H. 2023. Learning general policies with policy gradient methods. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning*, volume 19, 647–657.

Ståhlberg, S.; Bonet, B.; and Geffner, H. 2024. Learning General Policies for Classical Planning Domains: Getting Beyond C _2. *arXiv preprint arXiv:2403.11734*.

Toyer, S.; Thiébaux, S.; Trevizan, F.; and Xie, L. 2020. Asnets: Deep learning for generalised planning. *Journal of Artificial Intelligence Research*, 68: 1–68.

Zambaldi, V.; Raposo, D.; Santoro, A.; Bapst, V.; Li, Y.; Babuschkin, I.; Tuyls, K.; Reichert, D.; Lillicrap, T.; Lockhart, E.; et al. 2018. Relational deep reinforcement learning. *arXiv preprint arXiv:1806.01830*.