Learning Heuristics Through Pairwise Ranking

Primary Keywords: *(2) Learning;*

Abstract

Ranking

1 Introduction

Planning is a widely used technique across various fields. In general, planning involves finding a sequence of actions to achieve a specific objective. The classical planning problems are typically represented by a weighted directed graph, which can be solved using general path-searching algorithms (Geffner and Bonet 2022). To help the algorithm reach the goal more efficiently (by better efficiency, we mean both the optimality of the plan and minimising the number of nodes expanded), the search algorithm should bias towards the optimal path. To achieve this, the algorithms often need to retrieve information about the current state ~~toward~~s the goal as a guidance for the search direction (McDermott 1999). The standard approach to guiding the search algorithm towards the goal is by implementing a heuristic function, which estimates the cost to reach the goal (Bonet and Geffner 2001). Such heuristic functions are combined with the well-known Best-first search algorithms, which use an evaluation function to prioritize nodes ~~and always explore the node that minimises the evaluation function~~. The efficiency of the search vastly depends on the properties of the chosen heuristic function. Therefore, significant ~~research~~ efforts have been made to construct more informative heuristics(Bonet, Loerincs, and Geffner 1997; Hoffmann and Nebel 2001; Richter and Westphal 2010).

More recently, learning-based heuristics have caught increasing attention (Arfaee, Zilles, and Holte 2011; Bhardwaj, Choudhury, and Scherer 2017; Ha, An, and Kim 2022). Due to the success of deep learning in fields like reinforcement learning and imitation learning, many researchers have been trying to transfer the heuristic function into a neural network. However, most of these approaches formulate the heuristic function as a regression model either by directly predicting the real cost-to-goal and then combing with existing algorithms such as $A*$ (Garrett, Kaelbling, and Lozano-Pérez 2016; Takahashi et al. 2019; Shen, Trevizan, and Thiébaux 2020), or by imitating the $A*$ pipeline and implicitly replacing the heuristic function by a neural network(Ha, An, and Kim 2022; Agostinelli et al. 2021). The optimality and efficiency of these methods are based on the properties of the algorithms they were combined with or imitating. However, there are three problems with these approaches:

- It is important to note that although the real cost-to-goal is an admissible ~~and consistent~~ heuristic that makes the $A*$ algorithm optimal, it is not the only necessary condition for optimal efficiency in best-first search(Dechter and Pearl 1985). The main purpose of the heuristic function is to establish the node expansion priority queue in best-first search. Therefore, the rank of nodes in the queue, regardless of their numerical heuristic value, affects the outcome of the search.
- Training a neural network typically involves using state-value pairs that represent a state and its target heuristic value. However, computing the actual cost-to-goal can be expensive, particularly for large problems. As a result, it is challenging to create a dataset that is large enough to effectively train a deep neural network, as is commonly done in computer vision or natural language processing, where data acquisition and processing are relatively inexpensive.
- It is commonly observed that heuristic values provided by learning-based models will deviate from the optimal value when the instance is scaled up (Chen et al. 2023). This occurs because the training set typically includes state-value pairs from small instances, and the model has to learn how to scale input for larger instances. However, since the size of the problem can increase infinitely in theory, the target value also increases infinitely, which may exceed the neural network's capability.

On the other hand, learning to rank is an active research topic in relevance analysis and information retrieval. The major goal of ranking is to sort a list of elements under a certain query. The learning-to-rank approach solves the ranking problem by learning a ranking function that implicitly or explicitly assigns a score to each element and then sorts them based on the score. Intuitively, this is similar to the purpose of the heuristic function in best-first search.

In this paper, we frame the best-first search algorithm as a ranking problem. We first argue that using the heuristic function in best-first search is symmetric to the pointwise ranking function in the element ranking problem. Under such consideration, we then propose a novel architecture for learning the heuristic function through a pairwise rank-

ing approach. The proposed method can be integrated with any neural network-based heuristic learning model. We argue that such an approach imitates the priority queue that maximises the efficiency of best-first search, and hence resulting in better performance of the original model. Then, we conduct experiments by comparing our model by integrating with state-of-the-art heuristic learning models and analysing the usefulness of the architecture.

The main contributions of the paper are:

1. GBFS as ranking
2. pairwise ranking-based architecture

2 Preliminaries

2.1 Planning

using f here means you will have a notation clash when defining the f-value of a node. Best to use $a(s)$ or $a[[s]]$ to denote the result of applying action a in s.solved

We consider the classical planning problem model (Geffner and Bonet 2022) $\mathcal{S} = \langle S, s_0, S_G, A, a \rangle, c$ where S is a finite and discrete set of states, $s_0 \in S$ is the initial state, $s_G \subseteq S$ is the set of goal states, $A(s) \subseteq A$ is the set of actions applicable in state s, and the action $a \in A(s)$ is a deterministic transition function, which takes the state s as input and output a successor state s' if a is applicable in s. The cost function $c(a, s)$ returns the non-negative numerical cost of taking action a in state s. The instance has unit costs if all actions have cost 1. A plan or a solution of the model π is a sequence of actions $a_1, ..., a_l$ that generates a sequence of states $s_0, s_1, ..., s_l$ where $a_{i-1} \in A(s_{i-1}), s_i = a_{i-1}(s_{i-1})$ for $i \in \{1, \dots, l\}$, and $s_l \in S_G$. In other words, executing the plan will lead to a final state starting from the initial state. Because we only consider unit cost, the plan's cost equals the length. A plan is optimal if the plan's cost is minimal, denoted by π^*.

The classical planning model (with unit costs) can be represented by a directed graph $G = \langle \mathcal{V}, \mathcal{E}, W \rangle$ where each node $v_i \in \mathcal{V}$ corresponds to a state $s_i \in S$ and the edges between nodes $e = (v_i, v_j) \in \mathcal{E}$ represents the action a applicable in s_i and $a(s_i) = s_j$ with cost $W(e)$ (Cormen et al. 2009). The set $Succ(v_i) = \{v_j | \exists (v_i, v_j) \in \mathcal{E}\}$ represents the set of successors of node v_i. Then a plan in the graph is a path $[(v_0, v_1), (v_1, v_2), ..., (v_{l-1}, v_l)]$ starting from the initial state and ending in one of the goal states. Such correspondence reduces solving the planning problem to a path-finding problem into a weighted directed graph.

2.2 Standard Best-First Search

One major category of path-finding algorithms is the forward search algorithms, where the planner searches for the goal nodes $v_G \in \mathcal{V}$ by selecting and expanding nodes starting from the initial node v_0. Table 1 shows the generic forward search schema. Many search algorithms share the same structure but differ in implementations of SelectNode and ExpandNode.

The Best-First-Search Algorithm (BFS) with evaluation function f always expand the node with the minimal score. As an example, the A^* algorithm is a BFS node selection

Algorithm 1: Generic Forward Search Schema

Input: $\mathcal{S}, G$
Output: Plan π or *unsovable*

```
Let Nodes := {v_0}
while Nodes ≠ φ do
    Let n := SelectNode(Nodes)
    Let Nodes := Nodes \ {n}
    if n ∈ v_G then
        return Extract-Solution(n)
    else
        Let Successors := {n' | ∃(n, n') ∈ V}
        Set Nodes := AddNode(Nodes, Successors)
    end if
end while
```

strategy is implemented by prioritizing paths that lead to the goal through the use of $f(v) = g(v) + h(v)$ (Hart, Nilsson, and Raphael 1968), where $g(v)$ is the cost to v from the initial node, and $h(v)$ is a heuristic function estimating the cost-to-goal from v. Further, we use $h^*(v)$ to denote the real optimal cost-to-goal of node v. It is also common to guide the BFS purely on the heuristic $f(v) = h(v)$, and the search algorithm is generally referred to as the Greedy-Best-First Search (GBFS) (Bonet and Geffner 2001). To prevent looping expansion, a common trick is to have two lists (*Open list*$\mathcal{O}$ and *Close list*$\mathcal{C}$) to keep track of visited nodes and avoid loops. Visited nodes are stored in the closed list until the node enters the expansion frontier again with lower g, in which the node will be put back to the open list again.

2.3 Learning to Rank

Now we provide a formal definition of the ranking task. For consistency of the symbols, we only consider regression tasks in this paper. Given a set of queries $\mathcal{Q} = \{q_1, q_2, ..., q_m\}$, and a set of objects $\mathcal{B} = \{b_1, b_2, ..., b_n\}$, we aim to learn a partial order $\succeq$ over object space $\mathcal{F}$, such that $b_{i1} \succeq b_{i2} \succeq ... \succeq b_{in}$, where $i1, ..., in$ is a permutation of $1, ..., n$. We define the following properties of the order $\succeq$:

Definition 1 *Let $\succeq$ be a partial order on the feature space $\mathcal{F}$. Then $\forall a, b, c, \in \mathcal{F}$, the order has:*

- *Refleivity, if $a \succeq a$;*
- *Antisymmetry, if $a \not\succeq b \Rightarrow b \succeq a$;*
- *Transitivity, if $a \succeq b \wedge b \succeq c \Rightarrow a \succeq c$;*

Such ranking is generally represented by a ranking function $F(q, \mathcal{B}') : \mathcal{Q} \times \mathcal{B}^k \rightarrow \mathbf{R}^k$, where $\mathcal{B}' \subseteq \mathcal{B}$ and $k = |\mathcal{B}'|$. The function takes in a query and a subset of objects and returns a vector $S_{\mathcal{B}'}$ representing the scores of each element in $\mathcal{B}'$ (Li 2022). This function is called the total ranking function (Li 2022). Instead, it is also common to use the local ranking function $r(q, b) : \mathcal{Q} \times \mathcal{B} \rightarrow \mathbf{R}$, which takes in one element and returns a numerical score. Then the ranking η is obtained by an *argsort* function over $\mathcal{B}'$: $\eta = sort_{S_{\mathcal{B}'}}(B')$.

The learning-to-rank method obtains the ranking function through machine learning approaches (Liu et al. 2009). Given the set of queries and objects (divided into training

and testing sets), the goal during training is to learn the local ranking function $s = r(q, b)$. Then in the testing phase, the ranking function returns a numeric value for each object, and all elements are sorted by their corresponding values to obtain the final rank.

Based on how the ranker is trained, the family of regression learning-to-rank pipelines can be roughly divided into three groups: *pointwise, pairwise and listwise* approaches (Köppel et al. 2020). The pointwise approach takes an ⟨object, value⟩ pair as input. Then the model aims to learn the scoring function through a regression model. As we will show in Section 3, classical heuristic functions are typical pointwise rank functions with different queries.

Pairwise rankers take a pair of elements as input and output the pairwise difference between the scores of the two objects. The goal is to decide which object is more relevant to the query. The pairwise approach eliminates the need for trainers to provide numerical scores for objects; only the pairwise order is required. Therefore, the model is free to decide the range and distribution of numeric values assigned to each object.

Finally, a listwise ranker takes the complete list of elements as input and outputs a sorted list based on the ranking. The ranker is trained in an end-to-end manner. We leave the exploration of listwise ranking models as future works.

3 Ranking-based Best-First Search

We first provide a more generalised definition of the Best-First Search based on ranking.

Definition 2 (Ranking-based Best-First Search) *Given problem $\mathcal{S}$ and corresponding directed graph G, the Best-First Search algorithm takes a partial order $\succeq$ on the node space V such that the order satisfies reflexivity, antisymmetry and transitivity. Then the search algorithm proceeds as shown in 2.*

The only difference with the standard BFS is the algorithm chooses and expands the node v_0 in the open list $\mathcal{O}$ such that $\forall v \in \mathcal{O}, v \succeq v_0$ with some break-even method. Therefore, the standard Best-First Search algorithm is a special case of the above definition by representing the partial order with a numerical function:

Theorem 1 *Given deterministic evaluation function $f(v) : V \rightarrow \boldsymbol{R}$, the standard BFS with f is equivalent to the ranking-based BFS where the partial order $\succeq$ is implemented by $f(v)$, such that $v_i \succeq v_j \Leftrightarrow f(v_i) \geq f(v_j)$.*

Proof Consider a standard BFS algorithm with input problem $\mathcal{S}$, graph G and deterministic evaluation function f. For arbitrary nodes $v_i, v_j, v_z \in G$, we have a partial order represented by the real number ordering $\geq$, which satisfies reflexivity, anti-symmetry and transitivity:

- $f(v_i) = f(v_i) \rightarrow f(v_i) \geq f(v_i)$
- $f(v_i) \geq f(v_j) \Rightarrow -f(v_i) \leq -f(v_j)$
- $f(v_i) \geq f(v_j) \cap f(v_j) \geq f(v_z) \Rightarrow f(v_i) \geq f(v_j) \geq f(v_z)$

A critical observation in standard BFS is that the heuristic value does not affect the algorithm's outcome as long as the node order is maintained. For example, consider a situation where the open list only contains two states (with corresponding nodes n_1, n_2). And their heuristic value return by two different heuristic functions are $(10, 1)$ and $(100, 99)$, respectively. Because the GBFS algorithm always expands the node with the smallest heuristic value, regardless of which heuristic function is used, the GBFS algorithm will always expand n_2 before n_1. To bias the optimal path, the numeric value is not important as long as the heuristic assigns lower values to nodes on the path compared to other nodes on sub-optimal paths. Therefore, generalizing the evaluation function to the partial order allows us to optimise the algorithm performance directly.

Furthermore, we define the optimality and completeness of an order w.r.t the BFS.

Definition 3 (Complete Ranking) *The partial order $\succeq$ is complete if, for any solvable problem $\mathcal{S}$, the BFS with $\succeq$ will always terminate and return a plan.*

Definition 4 (Optimal Ranking) *The partial order $\succeq$ is optimal if, for any solvable problem $\mathcal{S}$, the BFS with $\succeq$ is complete and returns the optimal plan.*

3.1 Perfect Ranking

Therefore, we need to optimise the partial order $\succeq$ to improve the performance of the Ranking-based Best-First Search algorithm. In the perfect situation, the GBFS algorithm should only expand nodes on the optimal plan (Edelkamp and Schrödl 2011). In other words, the node on the optimal plan always has a lower heuristic value than its

Algorithm 2: Ranking-based BFS Algorithm with Reopening

Input: $\mathcal{S}, G, \succeq$
Output: Plan π or *unsovable*

```
Let Openlist𝒪 := {v_0}
Let g(v_0) = 0
Let Closelist𝒞 := ϕ
while 𝒪 ≠ ϕ do
    Choose n s.t. ∀v ∈ 𝒪, v ⪰ n
    Let 𝒪 := 𝒪 \ {n}
    if n ∈ v_G then
        return Extract-Solution(n)
    else
        Let 𝒞 := 𝒞 ∪ n
        Let succ := {n' | ∃(n, n') ∈ 𝒱}
        for all n' ∈ succ do
            Let g(n') = g(n) + 1
            if n' ∈ 𝒞 as n'_𝒞 and g(n') < g(n'_𝒞) then
                Let 𝒞 := 𝒞 \ {n'_𝒞}
                Let 𝒪 := 𝒪 ∪ {n'}
            else if n' ∈ 𝒪 as n'_𝒪 and g(n') < g(n'_𝒪) then
                Update g(n'_𝒪) = g(n')
            end if
        end for
    end if
end while
```

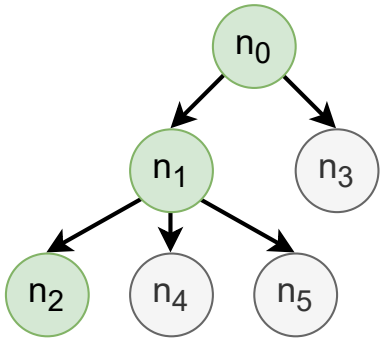

Figure 1: A example graph representation of part of the problem. The optimal path is $[n_0, n_1, n_2]$

siblings and siblings of other expanded nodes in the optimal path. We use a similar definition of strictly optimal efficiency as defined in (Chrestien et al. 2023).

Definition 5 (Perfect Ranking) *For a solvable planning problem $\mathcal{S}$ and corresponding graph G, a partial order $\succeq^*$ is a perfect ranking if:*

- *It is reflexivity, anti-symmetry and transitivity.*
- *There exists an optimal path $\pi = [v_0, v_1, ..., v_m]$ such that:*
  - *While $\mathcal{O} \neq \phi$, there is one and only one node in the optimal path such that the node is in the open list: $|\pi \cap \mathcal{O}| = 1$.*
  - *For any nodes in the optimal path $v_i \in \pi$, it satisfies $\forall v_j \in \cup_{k=0}^{i-1} succ(v_k), v_j \succeq^* v_i$.*

As an example, Figure 1 is a graph representation of an instance with six states and unit costs. The optimal path is $[n_0, n_1, n_2]$ while $\{n_3, n_4, n_5\}$ are siblings along the optimal path. Assume there exists a perfect ranking $\succeq^*$ and initially the open list $\mathcal{O}$ contains the initial node n_0, then the BFS algorithm will remove and expand n_0 and add n_1, n_3 to the open list with $n_3 \succeq^* n_1$. Then in the next step n_1 will be expanded and n_2, n_4, n_5 are added to the open list with $n_3 \succeq^* n_2 \cap n_4 \succeq^* n_2 \cap n_5 \succeq^* n_2$. Notice that the orders among n_3, n_4 and n_5 are not specified as they will not affect the outcome of BFS.

Theorem 2 *The Perfect Ranking is a complete and optimal ranking.*

Proof Given a solvable problem $\mathcal{S}$ with one of the optimal plan $\pi^* = [v_0, v_1, ..., v_n]$, assume we have a perfect ranking $\succeq^*$. After BFS expands the initial node v_0, the open list must contain v_1. This is because π^* is an optimal and valid plan. Then, there must exist some action a_0 applicable at the initial state and $a_0(v_0) = v_1$. Given the perfect ranking, the next node expanded must be v_1. Similarly, v_2 is now in the open list, and in the next iteration, v_2 is expanded. Inductively, after n iterations, v_n is in the open list and in the next iteration, v_n is chosen and is the goal state because it is the last node in the optimal plan. Therefore, the algorithm will always terminate with the optimal plan π.

3.2 Learning the Perfect Order Ranking Function

We can implement a perfect order using a local ranking function:

Definition 6 (Perfect Order Ranking Function) *Given domain D, a local ranking function $r_D : V \to \mathbf{R}$ represents a perfect order $\succeq_D^*$ which satisfy the following conditions:*

- *r_D is reflexivity, anti-symmetry and transitivity.*
- *For a solvable problem $\mathcal{S}$ of domain D with graph G, there exists an optimal path $\pi = [v_0, v_1, ..., v_m]$, such that for $i \in \{0, 1, ..., m\}, \forall v_j \in \cup_{k=0}^{i-1} succ(v_k), r_D(v_j) \geq r(v_i)$.*

We could try to approximate such a function through neural networks based on the Universal approximation theorem. This leads us to the learn-to-rank method. To achieve this, we consider all three general training approaches: pointwise, pairwise and listwise. The pointwise approach generally requires state-value pairs as the training data, where the numerical values are sample scores of the target ranking. However, for the perfect ranking, we don't have any numerical score from the unknown perfect ranking function. Therefore, generating the dataset requires either human-engineered scores or some pre-knowledge of the perfect ranking of the given domain, which could be difficult to obtain.

On the other hand, the list-wise approach requires a sorted list of data as the training set. However, in the perfect ranking, we only specify the rankings of nodes in the optimal path compared to their siblings, and the order between siblings is undefined. We could specify a particular order for the sake of training. However, such unnecessary information could bias the model towards sub-optimal results. Besides, limited by our available training samples for planning tasks, the optimal path along is insufficient to well-train the model.

On the contrary, the pairwise approach does not require numerical scores or list-wise data ordering. Under the pairwise approach, only the pairwise order is used as input, which perfectly fits our situation as we only require pairwise orders between nodes in the optimal plan and their siblings. Therefore, in the following paragraph, we propose a pipeline based on the pairwise learning approach(Köppel et al. 2020).

3.3 Pairwise Approach Rank Learning

We first represent the pairwise perfect ranking by a polarity function $p : V \times V \to \mathbf{R}$ such that

$$\forall v_i, v_j \in V, v_i \succeq v_j \Leftrightarrow p(v_i, v_j) \geq 0 \quad (1)$$

The goal is to learn p at first and then extract the local ranking function r. We construct a neural network model to represent p based on the representative pairwise ranking model DirectRanker(Köppel et al. 2020). The schema is shown in Figure 2. The model consists of two parts. Given a pair of nodes (v_i, v_j) with pairwise order $v_i \succeq v_j$, the first part nn takes each node as input and outputs an embedding vector of shape $\mathbf{R}^n$.

$$emb_v = nn(v), v = \{v_i, v_j\} \quad (2)$$

The second part of the model takes the difference of the vectors and passes it through a single neuron τ without bias

and with an anti-symmetric and monotonically increasing activation function $\sigma : \mathbf{R} \to \mathbf{R}$. The output p is expected to be non-negative. And if $v_j \succeq v_i$ then p should be non-positive.

$$p = \tau(emb_{v_i} - emb_{v_j}) \tag{3}$$

Theorem 3 *Given instance $\mathcal{S}$ from domain D, after training the polarity function $p : V \times V \to \mathbf{R}$ represented by the neural network shown in Figure 2 with perfect ranking $\succeq_D^*$, we can extract a local ranking function $r : V \to \mathbf{R}$ representing $\succeq_D^*$.*

Proof We first decompose the local ranking function $r : V \to \mathbf{R}$ into two parts: the embedding function $l : V \to \mathbf{R}^k$ and aggregation function $t : \mathbf{R}^n \to \mathbf{R}$. The embedding function takes one node as input and outputs an embedding vector of dimension k. The embedding vector is then passed to the aggregation function that aggregates the vector into a single value. Given node space V, the order $\succeq_D^*$ is now represented by $r(v) = t(l(v)) : V \to \mathbf{R}$.

Given two nodes $v_i, v_j \in V_D$, without loss of generality, we assume pairwise ranking $v_i \succeq_D^* v_j$. Then from Equation 2 and 3 of the model, we have:

$$p = \tau(nn(v_i) - nn(v_j)) \geq 0 \tag{4}$$

Let the neuron $\tau(x) = \sigma(w(x))$, where σ is the activation function and w is the weight of the neuron. Then the anti-symmetry and monotonically increasing of σ means:

$$p \geq 0 \Leftrightarrow w(nn(v_i) - nn(v_j)) \geq 0 \tag{5}$$

Because the neuron has no bias, the weight is simply a linear function:

$$\begin{aligned} & w(nn(v_i) - nn(v_j)) \geq 0 \\ \Leftrightarrow\ & w(nn(v_i) - w(nn(v_j) \geq 0 \\ \Leftrightarrow\ & w(nn(v_i) \geq w(nn(v_j)) \end{aligned}$$

Now let the embedding function $l := nn$, aggregation function $t = w$, clearly we have:

$$v_i \succeq_D^* v_j \Leftrightarrow r(v_i) \geq r(v_j) \tag{6}$$

Therefore, we can extract local ranking function $r(v) = w(nn(v))$ representing the perfect order $\succeq_D^*$. And from Theorem 1 we have proved that we can use $r(v)$ as the evaluation function for the BFS.

3.4 Batched Evaluation

We can improve the computation efficiency of the ranking model by batching up the nodes with pairwise orders (Damke and Hüllermeier 2021). Consider a list of nodes $IN = [v_1, v_2, v_3, ...v_m]$ with a set of pairwise orders $OD = \{v_i \succeq v_j | v_i, v_j \in IN\}$, let the indices of nodes at the left-hand side of each order be o_l, and the indices for the right-hand sides be o_r. The first part of the model takes IN as input and outputs a matrix $\mathbf{E}$ of shape $\mathbf{R}^{m \times n}$ where the i-the row $\mathbf{E}_i$ is a vector embedding of dimension $\mathbf{R}^n$ for the i-th node in IN.

$$\mathbf{E} = \begin{bmatrix} nn(v_1) \\ nn(v_2) \\ ... \\ nn(v_m) \end{bmatrix} \tag{7}$$

The second part of the model takes the set of pairwise orders OD and the matrix from the first part as input. Then we extract the rows by indices from $\mathbf{E}$ corresponding to the nodes at both sides of the orders in OD, and concatenate them to separated matrices $\mathbf{E}_l, \mathbf{E}_r$. Then the second part of the model computes

$$p = \tau(\mathbf{E}_l - \mathbf{E}_r) \tag{8}$$

Here τ, as before, is a single neuron without bias and with an anti-symmetric monotonically increasing activation function. The output p is expected to be a vector of shape $\mathbf{R}^{|OD|}$ with all entries non-negative.

The batched model reduces repeated computation of node embeddings. Consider three nodes $v_i \succeq v_j \succeq v_k$. In the original model, it first takes the pair (v_i, v_j) and then takes (v_j, v_k). As a result, the embedding of v_j is computed twice. However, we put all three nodes into one list using the batching approach and compute their embeddings once. In the experiment, the batched model significantly improved training time compared to the original model.

4 Experiments

To validate the effectiveness of the proposed method, we implemented the ranker with state-of-the-art neural-network-based heuristic learning models. We then compare the model's performance with the original ones and analyse the results.

We utilized GOOSE(Chen et al. 2023), a heuristic learning architecture based on Graph Neural Networks (GNN), to implement the ranker. The model offers various domain-independent node encoding methods to transform search nodes into graphs. The graph representation is then fed into a GNN, which outputs a numerical value to predict the node's heuristic. The architecture has two main benefits. Firstly, it allows for domain-independent encoding and training pipelines, which means the ranker can learn heuristics that apply to any domain. Secondly, the model has been thoroughly tested using various datasets and experiments, serving as a reliable benchmark for our model.

Probably more experiments?

4.1 Dataset

4.2 Training Methods

4.3 Evaluations

4.4 Results

5 Related Works

6 Conclusion

References

Agostinelli, F.; Shmakov, A.; McAleer, S.; Fox, R.; and Baldi, P. 2021. A* search without expansions: Learning heuristic functions with deep q-networks. *arXiv preprint arXiv:2102.04518*.

Arfaee, S. J.; Zilles, S.; and Holte, R. C. 2011. Learning heuristic functions for large state spaces. *Artificial Intelligence*, 175(16-17): 2075–2098.

domain	ff	llg(max)	llg(max)+ranker	slg	slg+ranker	llg(mean)	llg(mean)+ranker
blocks	22	25	9	8	21	21	**55**
ferry	**90**	73	**90**	23	40	70	**90**
gripper	**18**	14	11	5	7	**18**	**18**
n-puzzle	**33**	0	0	0	3	0	0
sokoban	**86**	17	12	8	25	39	36
spanner	0	15	46	0	0	46	**49**
visitall	5	31	25	27	35	**30**	**30**
visitsome	24	20	20	41	40	**55**	47

Table 1: Coverage by domain for different models

Bhardwaj, M.; Choudhury, S.; and Scherer, S. 2017. Learning heuristic search via imitation. In *Conference on Robot Learning*, 271–280. PMLR.

Bonet, B.; and Geffner, H. 2001. Planning as heuristic search. *Artificial Intelligence*, 129(1-2): 5–33.

Bonet, B.; Loerincs, G.; and Geffner, H. 1997. A robust and fast action selection mechanism for planning. In *Proceedings of the fourteenth national conference on artificial intelligence and ninth conference on Innovative applications of artificial intelligence*, 714–719.

Chen, D. Z.; et al. 2023. GOOSE: Learning Heuristics and Parallelising Search for Grounded and Lifted Planning.

Chrestien, L.; Pevnỳ, T.; Edelkamp, S.; and Komenda, A. 2023. Optimize Planning Heuristics to Rank, not to Estimate Cost-to-Goal. *arXiv preprint arXiv:2310.19463*.

Cormen, T. H.; Leiserson, C. E.; Rivest, R.; and Stein, C. 2009. Introduction to algorithms. Massachusetts. *London: MIT Press. Section*, 25: 636–40.

Damke, C.; and Hüllermeier, E. 2021. Ranking structured objects with graph neural networks. In *Discovery Science: 24th International Conference, DS 2021, Halifax, NS, Canada, October 11–13, 2021, Proceedings 24*, 166–180. Springer.

Dechter, R.; and Pearl, J. 1985. Generalized Best-First Search Strategies and the Optimality of A*. *J. ACM*, 32(3): 505–536.

Edelkamp, S.; and Schrödl, S. 2011. *Heuristic search: theory and applications*. Elsevier.

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

Geffner, H.; and Bonet, B. 2022. *A concise introduction to models and methods for automated planning*. Springer Nature.

Ha, J.; An, B.; and Kim, S. 2022. Reinforcement Learning Heuristic A. *IEEE Transactions on Industrial Informatics*, 19(3): 2307–2316.

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.

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

Köppel, M.; Segner, A.; Wagener, M.; Pensel, L.; Karwath, A.; and Kramer, S. 2020. Pairwise learning to rank by neural networks revisited: Reconstruction, theoretical analysis and practical performance. In *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2019, Würzburg, Germany, September 16–20, 2019, Proceedings, Part III*, 237–252. Springer.

Li, H. 2022. *Learning to rank for information retrieval and natural language processing*. Springer Nature.

Liu, T.-Y.; et al. 2009. Learning to rank for information retrieval. *Foundations and Trends® in Information Retrieval*, 3(3): 225–331.

McDermott, D. 1999. Using regression-match graphs to control search in planning. *Artificial Intelligence*, 109(1-2): 111–159.

Richter, S.; and Westphal, M. 2010. The LAMA planner: Guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research*, 39: 127–177.

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.

Takahashi, T.; Sun, H.; Tian, D.; and Wang, Y. 2019. Learning heuristic functions for mobile robot path planning using deep neural networks. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 29, 764–772.

NOT ARXIV

SHORTEN

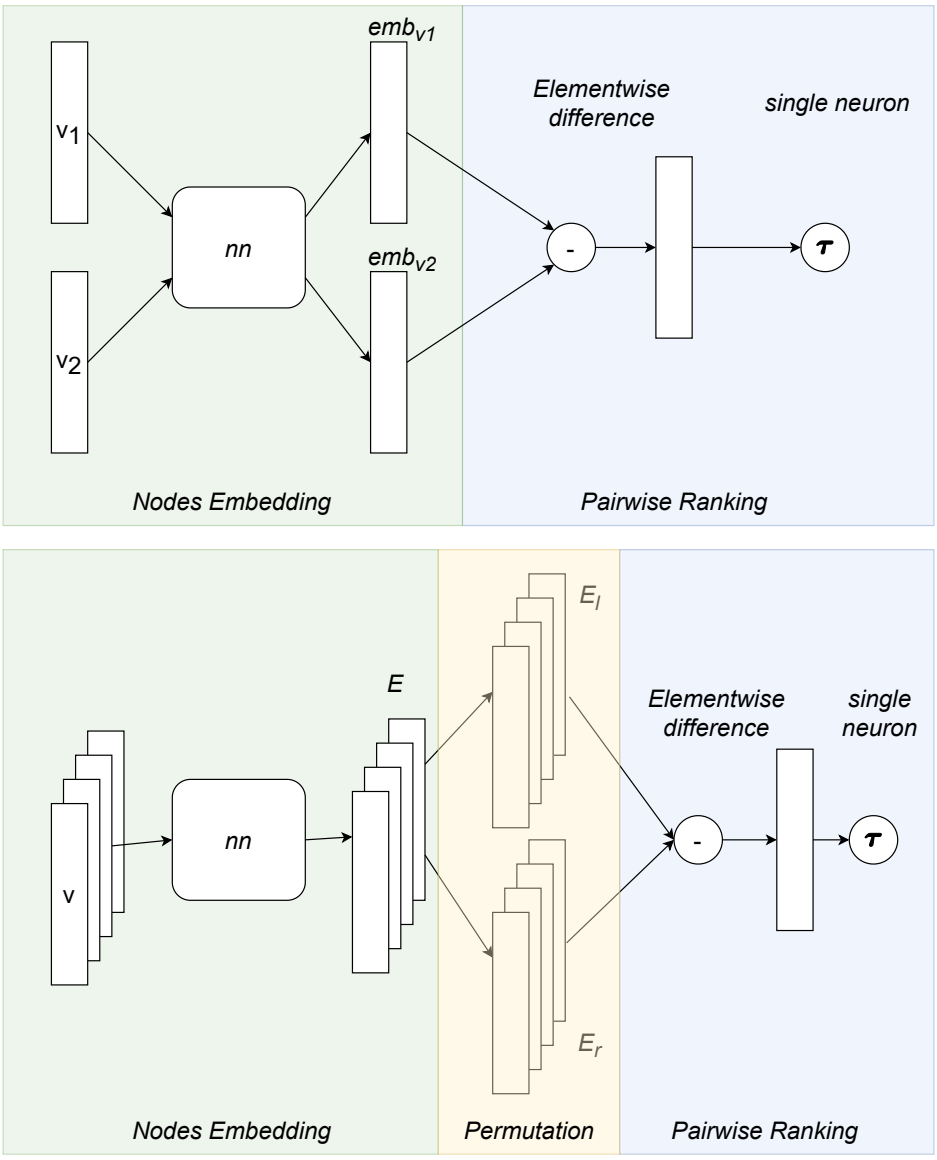

Figure 2: The structure of proposed model.