TransPath: Learning Heuristics For Grid-Based Pathfinding via Transformers

Daniil Kirilenko[1], **Anton Andreychuk**[2], **Aleksandr Panov**[1, 2], **Konstantin Yakovlev**[1, 2]

[1] Federal Research Center for Computer Science and Control of Russian Academy of Sciences, Moscow, Russia
[2] AIRI, Moscow, Russia
anedanman@gmail.com, andreychuk@airi.net, panov@airi.net, yakovlev@isa.ru

Abstract

Heuristic search algorithms, e.g. A*, are the commonly used tools for pathfinding on grids, i.e. graphs of regular structure that are widely employed to represent environments in robotics, video games, etc. Instance-independent heuristics for grid graphs, e.g. Manhattan distance, do not take the obstacles into account, and thus the search led by such heuristics performs poorly in obstacle-rich environments. To this end, we suggest learning the instance-dependent heuristic proxies that are supposed to notably increase the efficiency of the search. The first heuristic proxy we suggest to learn is the correction factor, i.e. the ratio between the instance-independent cost-to-go estimate and the perfect one (computed offline at the training phase). Unlike learning the absolute values of the cost-to-go heuristic function, which was known before, learning the correction factor utilizes the knowledge of the instance-independent heuristic. The second heuristic proxy is the path probability, which indicates how likely the grid cell is lying on the shortest path. This heuristic can be employed in the Focal Search framework as the secondary heuristic, allowing us to preserve the guarantees on the bounded suboptimality of the solution. We learn both suggested heuristics in a supervised fashion with the state-of-the-art neural networks containing attention blocks (transformers). We conduct a thorough empirical evaluation on a comprehensive dataset of planning tasks, showing that the suggested techniques *i*) reduce the computational effort of the A* up to a factor of 4x while producing the solutions, whose costs exceed those of the optimal solutions by less than 0.3% on average; *ii*) outperform the competitors, which include the conventional techniques from the heuristic search, i.e. weighted A*, as well as the state-of-the-art learnable planners.

The project web-page is: https://airi-institute.github.io/TransPath/.

Introduction

Path planning for a mobile agent in the static environment is a fundamental problem in AI that is often framed as a graph search problem. Within this approach, first, an agent's workspace is discretized to a graph. Second, a search algorithm is invoked on this graph to find a path from start to goal. Arguably, 2^k-connected grids (Rivera et al. 2020) are the most widely used graphs for path planning in a variety of applications (robotics, video games, etc.).

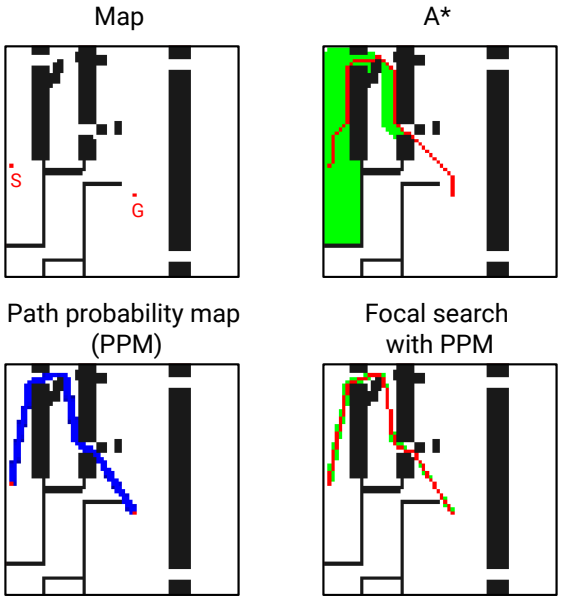

Figure 1: The difference between A* and our approach. The expanded nodes are shown in green, while the path is highlighted in red. Blue regions are predicted by the neural network to contain path cells with high probability.

Path planning on a grid is commonly accomplished by a heuristic search algorithm, e.g. A* (Hart, Nilsson, and Raphael 1968) or one of its numerous modifications. Performance of such algorithms is heavily dependent on the input heuristic that comes in the form of a function that estimates the cost of the path to the goal for each node of the graph (*cost-to-go heuristic*). If the heuristic is perfect, i.e., for every node its value equals the cost of the shortest path, a search algorithm explores only the nodes that lie on one of the minimum-cost paths. However, such a perfect heuristic is instance-dependent and cannot be encoded in the closed-loop form. In practice, instance-independent heuristics, e.g. Manhattan distance, are typically used for grid-based path planning. These heuristics do not take obstacles into account

and, consequently, perform poorly in obstacle-rich environments.

One of the recent and promising approaches to automated construction of the instance-dependent heuristics (and for path planning in general) is utilizing machine learning, specifically, deep learning (Speck et al. 2021; Janner et al. 2022). As grids can be viewed as the binary images, it is appealing to utilize the recent advances in convolutional neural networks (CNNs) (Ramachandran, Zoph, and Le 2017; Tan and Le 2019) to extract the informative features from the image representations of the pathfinding problems and embed these features into the heuristic search algorithm. For example, in (Takahashi et al. 2019) it was suggested to learn perfect cost-to-go heuristic in a supervised fashion. In a more recent study (Yonetani et al. 2021), a more involved approach was introduced when a matrix-based A* was proposed and used for learning. Consequently, the deep neural network model was trained end-to-end. That paper did not predict the conventional cost-to-go heuristic, but rather assigned the additional cost to each grid cell with the intuition that unpromising nodes would be assigned a high cost by the neural network. Thus, at the planning phase, the search would avoid the cells with the high costs.

In this work, we follow the described paradigm and further examine the ways of how heuristic search can benefit from state-of-the-art deep learning techniques in the context of the grid-based path planning. The distinguishable features of our work are as follows. We consider 8-connected grids with non-uniform costs (i.e., diagonal moves cost more than the cardinal ones), unlike the previous works that considered unit cost domains. We suggest learning the novel heuristic proxies for the problem at hand. Instead of learning to predict the values of the perfect cost-to-go heuristic, we suggest learning the *correction factor* of the heuristic function, which is the ratio between the instance-independent heuristic and the perfect heuristic. Thus, a correction factor embeds information about both of these heuristics. Our empirical evaluation confirms that learning the correction factor leads to a notably better performance than learning the absolute values of the conventional cost-to-go heuristic.

We also suggest learning the *path probability map*, which assigns to each grid cell the probability of belonging to the shortest path. This can be used as a secondary heuristic in the bounded sub-optimal search algorithm: Focal Search (Pearl and Kim 1982). Thus, we are able to preserve the theoretical guarantees on a sub-optimality bound of the constructed solution while speeding up the search, as our experiments show.

To learn the correction factor and path probabilities, we utilize supervised deep learning. In doing so, We employ a neural network model, that is a combination of the convolutional encoder-decoder with the attention blocks (Vaswani et al. 2017) (the so-called transformers). Such a combination allows the neural network to capture and "reason about" both the local features of a given map (corners of obstacles, passages etc.) and the relations between them, e.g. "there is a passage between the two regions of interest".

To evaluate the suggested techniques, a comprehensive dataset of the challenging planning tasks has been created that extends the dataset previously used in closely related works (Yonetani et al. 2021). We compare our approach with those of the competitors that include both the deep learning techniques and the traditional ones, and demonstrate its superiority in terms of the computational effort and solution cost. Overall, we have been able to reduce the computational effort compared to A* up to a factor of 4x while producing the solutions, which costs exceed the costs of the optimal solutions by less than 0.3% on average.

Related Work

Utilizing machine learning for graph search in general and grid-based pathfinding, in particular, has been getting increased attention recently. In (Pogančić et al. 2020) an approach was presented that allows one to combine a learnable module with a non-learnable (classic) solver of a combinatorial problem and train the pipeline end-to-end. The approach was evaluated on several problems including pathfinding on the grids, represented as images, where the transition costs are not known apriori. In (Li, Chen, and Koltun 2018), a learning-based approach for solving certain NP-hard problems was presented that exploited a graph convolutional network to estimate the likelihood of whether a certain vertex of the graph is a part of the optimal solution. In (Pándy et al. 2022), a framework was proposed that suggests imitation learning-based heuristic search paradigm with a learnable explored graph memory. In brief, it learns a representation that captures the structure of the so far explored graph, so that it can then better select what node to explore next. Such an approach can be viewed as solving a sequential decision-making problem. Similar approaches were introduced in (Tamar et al. 2016; Bhardwaj, Choudhury, and Scherer 2017; Panov, Yakovlev, and Suvorov 2018). Special care to the properties of the learned heuristics, i.e. admissibility, is given in (Li et al. 2022). Additionally, this work introduces a version of A* search (Hart, Nilsson, and Raphael 1968) that leverages parallel execution on graphical processing units (GPUs) that are widespread in machine learning computations. Analogous batch-handling techniques for heuristic search were considered in (Greco et al. 2022).

The papers that are especially relevant to this one are (Soboleva and Yakovlev 2019; Takahashi et al. 2019; Yonetani et al. 2021) as they all suggest specific machine learning techniques tailored to grid-based pathfinidng. In the former a generative adversarial (neural) network is proposed to generate the solutions of the pathfinding instances. In (Takahashi et al. 2019) a convolutional neural network is used to predict the values of the cost-to-go heuristic. In (Yonetani et al. 2021) Neural A* is introduced, which is a combination of the encoder-decoder predictor and a differentiable module that imitates A* search on grids. The predictor is a neural network, which estimates the transition costs on the grid with the intuition that transitions to unpromising parts of the map should cost more. The presence of the differentiable A* module allows training the pipeline end-to-end. Neural A* was empirically shown to consistently outperform a range of competitors for grid-based pathfinding. In this work, we use Neural A* as a baseline to compare with.

Background

Pathfinding Problem

Consider a grid, Gr, composed of the blocked and free cells and two distinct free grid cells, $start$ and $goal$. Being at any free cell, an agent is allowed to move to one of its cardinally- or diagonally-adjacent neighboring cells, provided the latter is free. The cardinal moves incur the cost of 1, while the diagonal ones incur the cost of $\sqrt{2}$. This setting can be referred to as the 8-connected grid with non-uniform costs.

A path, $\pi(start, goal)$, is a sequence of the adjacent cells, starting with $start$ and ending with $goal$: $\pi = (c_0 = start, c_1, c_2, \ldots, c_n = goal)$. A path is valid *iff* all the cells forming this path are free. The cost of the valid path is the sum of costs associated with the transitions between the cells comprising the path: $cost(\pi) = \sum_{i=0}^{i=n-1} cost(c_i, c_{i+1})$.

Denote a set of all valid paths connecting $start$ and $goal$ as Π. The least cost (shortest) path from $start$ to $goal$ is $\pi^* \in \Pi$, s.t. $\forall \pi \in \Pi : cost(\pi*) \leq cost(\pi)$.

The pathfinding problem is a tuple $(Gr, start, goal)$, which asks to find a *valid* path from $start$ to $goal$ on Gr. The *shortest path* is said to be the *optimal solution*. Given a positive real number, $w > 1$, the bounded sub-optimal solution is a valid path whose cost exceeds that of the shortest path by no more than a factor of w: $cost(\pi^w) \leq w \cdot cost(\pi^*)$.

In this work, we are specifically interested in obtaining *i*) valid paths; *ii*) bounded sub-optimal paths. The problem of obtaining optimal solutions is beyond the scope of this paper.

A* Search

A* is a heuristic search algorithm with strong theoretical guarantees that is widely used to solve the pathfinding problems stated above. A* incrementally builds a search tree of nodes, where each node corresponds to a grid cell and bears the additional search-related data. This data includes the g-value of the node, which is the cost of the path to the node from the root of the tree. h-value of the node is the heuristic estimate of the cost of the path from the current node to the goal one. The sum of g- and h-values is called the f-value of the node.

Nodes are generated and added to the A* search tree via the iterative *expansions*. To expand a node means to generate all of its valid successors, i.e., the successors that correspond to the valid moves on a grid, to compute their g-values (as the sum of the g-value of the expanded node plus the transition cost), and to add certain successors to the tree. A successor is added to the tree only if it is not yet present in the tree or, alternatively, if the same node (i.e. the one corresponding to the same grid cell) exists, but its g-value is greater than the newly computed one.

A* performs expansions in the systematic fashion (starting with the $start$ node). It maintains a list of nodes that have been generated but not yet expanded. This list is typically referred to as $OPEN$, while the list of the expanded nodes is designated as $CLOSED$. At each iteration, a node with the minimal f-value is chosen from $OPEN$ for the expansion. A* stops when the goal node is extracted from $OPEN$. At this point, the sought path can be reconstructed using the backpointers in the search tree.

The performance of the algorithm, i.e. the number of the iterations before termination and the guarantees on the cost of the found path, is largely dependent on the used heuristic.

Heuristics The heuristic is called perfect, denoted as h^*, if for every node, its value equals the true cost-to-go: $h^*(n) = cost(\pi^*(n, goal))$. The heuristic is called *admissible* if it never overestimates the true cost-to-go: $h(n) \leq h^*(n)$. The heuristic is said to be *consistent* or *monotone* if $\forall n, n' : h(n) \leq h(n') + cost(\pi^*(n, n'))$.

A range of consistent and admissible instance-independent heuristics are known for the 8-connected grids, e.g. Chebyshev distance, Euclidean distance, or Octile distance. They all can be efficiently computed in the closed-loop form for any grid cell. Without the loss of generality, in this work, we assume that the Octile distance is used as the heuristic function.

It is known that A* with an admissible heuristic is guaranteed to find the optimal solution. Moreover, if the heuristic is *consistent* (as is in our case) it is not possible to find a better path to any of the expanded nodes, which infers that no node can be expanded more than once. Still, the number of such expansions can be significantly large as depicted in Fig. 1. The reason is that the Octile distance, being an instance-independent heuristic, is unaware of the blocked cells and drives the search toward the obstacle via the low f-values of the nodes residing in its vicinity.

Weighted A* and Focal Search

Weighted A* One of the widespread ways to trade off optimality for the computational efficiency in grid-based pathfinding is to employ a *weighted heuristic*, i.e. to order nodes in $OPEN$ not by their $g + h$ values, but rather by $g + w \cdot h$ values, where $w \geq 1$. Such a modification of A*, typically referred to as WA* (Weighted A*), is known to provide bounded sup-optimal solutions w.r.t. w.

Focal Search Focal Search (FS) (Pearl and Kim 1982) is another technique tailored to lower the number of search iterations while providing the bound on the optimally of the resultant solution. In FS, an additional list of nodes is maintained called $FOCAL$. It is formed of the nodes residing in $OPEN$, whose f-values do not exceed the minimum f-value in $OPEN$, f_{min}, by a factor of w (given the sub-optimality bound). $FOCAL$ is ordered in accordance with the secondary heuristic, h_{FOCAL}, which does not have to be consistent or even admissible. The node to be expanded is chosen from $FOCAL$ in accordance with the ordering imposed by h_{FOCAL} (and removed from $OPEN$ as well). In case $OPEN$ is updated as a result of the expansion, $FOCAL$ is modified accordingly. The stop criterion is the same as in A*. FS is guaranteed to obtain bounded sup-optimal solutions. Indeed, the number of search iterations and, thus, the computational efficiency of FS is strongly dependent on h_{FOCAL}.

Fig. 2 shows the pseudocode of a generic heuristic search algorithm. Different colors correspond to different variants of the algorithm as explained in the caption.

Input: Grid Gr, *start* node, *goal* node, heuristic function h, sub-optimality factor w, h_{FOCAL} – secondary heuristic for Focal Search
Output: path π

$g(start) := 0$; $\forall n \neq start\ g(n) := \infty$
$OPEN := \{start\}$; $CLOSED := \emptyset$
while $OPEN \neq \emptyset$ **do**
  $n := GetBestNode(OPEN, FOCAL, h_{FOCAL})$
  remove n from $OPEN$and $FOCAL$
  insert n into $CLOSED$
  if f_{min} *has changed* **then**
    update $FOCAL$
  if n *is goal* **then**
    return $ReconstructPath(n)$
  for *each* n' *in* $GetSuccessors(Gr,n)$ **do**
    if $g(n') > g(n) + cost(n, n')$ **then**
      $g(n') := g(n) + cost(n, n')$
      $f(n') := g(n') + w \cdot h(n')/w(n')$
      update or insert n' in $OPEN$
      if $f(n') \leq w \cdot f_{min}$ **then**
        update or insert n' in $FOCAL$
return *path not found*

Figure 2: A generic search algorithm. A* executes black parts only. WA* is shown in black and red, Focal Search in black and purple, our variant of WA* in black and blue.

Method

Recall that we are interested in two variants of the pathfinding problem. The first variant asks to find a valid path on a grid, without specifying any constraints on the cost of the path, VP-PROBLEM. The second variant assumes that a sub-optimality bound, $w \geq 1$, is specified and the task is to find a path whose cost does not exceed the cost of the optimal path by more than a factor of w, BSP-PROBLEM.

The solvers that we suggest for both problems share their structure. Each of them is composed of the two building blocks. First, a deep neural network is used to process the input grid and to predict the values of the heuristic function that will be used later. Second, a heuristic search algorithm is invoked that utilizes the heuristic data from the neural network. The neural network used for VP-PROBLEM and BSP-PROBLEM has the same architecture; however, in each case, the output heuristic is different. The heuristic search algorithm is also different. For solving VP-PROBLEM, we utilize WA*, while for BSP-PROBLEM, Focal Search (FS) is used.

Heuristics Learned

The first type of the heuristic is the *correction factor* (cf), which is defined as the ratio of the value of the available instance-independent heuristicto the value of the perfect heuristic: $cf(n) = h(n)/h^*(n)$. We suggest plugging the predicted cf-values into the WA* algorithm as shown in Fig. 2 (black + blue code fragments). I.e., the f-value of a node is computed as $f(n) = g(n) + h(n)/cf(n)$. This can be thought of as running WA* that uses individual weights for the search nodes. As there is no theoretical bound on the error of predicting cf-values, the resultant search algorithm provides no guarantees on the resultant cost.

In general, predicting cf-values may seem similar to predicting the values of the perfect cost-to-go heuristic as was proposed in (Takahashi et al. 2019). However, there exists a crucial difference, which is twofold. First, when learning the cost-to-go heuristic, an additional technical step is needed that transfers the range of the heuristic to the range typically employed in deep learning, e.g. $[0, 1]$. Meanwhile, the range of the introduced correction factor is $[0, 1]$ by design; thus, no auxiliary transformations are required. Second, the correction factor encompasses more heuristic data, as it is a combination of both instance-dependent and instance-independent heuristics. As confirmed by our experiments, learning cf-values instead of h^*-values leads to a notable boost in the performance.

The second suggested heuristic is tailored to serve as the secondary heuristic for the FS, h_{FOCAL}, which is employed to solve the BSP-PROBLEM. Intuitively, we want from h_{FOCAL} to distinguish the nodes that are likely to yield rapid progress toward the goal. To this end, we suggest assigning (and learning to predict) a value to each grid cell that tells us how likely it is that this cell lies on the shortest path between $start$ and $goal$. We call this value a *path probability*, pp-value, and, by design, its range is within $[0, 1]$. Learning to correctly predict $pp - values$ may be thought of as attempting to learn to solve the pathfinding queries directly. I.e., if we were able to obtain such predictions for where pp-values of 1 were assigned to the cells lying on the shortest path, while the other cells were assigned pp-values of 0, then we would not need to run the search algorithm at all. However, in practice, this is not realistic, and thus, we use the predicted pp-values as h_{FOCAL} values in the FS.

Learning Supervision

An evident approach to learning the suggested heuristics is to create a rich dataset of pathfinding instances with the annotated ground-truth cf- and pp-values and to train the neural network to minimize the error between its predictions and the ground-truth values. Using the techniques introduced in (Pogančić et al. 2020; Yonetani et al. 2021), one might consider another option of learning, i.e. including the search algorithm in the learning pipeline and to back-propagating the search error through it. This option is especially useful when it is hard or impossible to create ground-truth samples (like in planning on images when the cost of the transition is unknown). We have experimented with both types of learning and found that for our setting, the first option is preferable for the following reasons. First, there is no problem to create ground-truth samples for cf- and pp-values (technical details on this will follow shortly). Second, learning without differentiable planner is much faster (up to 4x in our setup). Third and not least of all, our experiments have shown that learning the suggested model with the supervision from the ground-truth values leads to a consistently better performance.

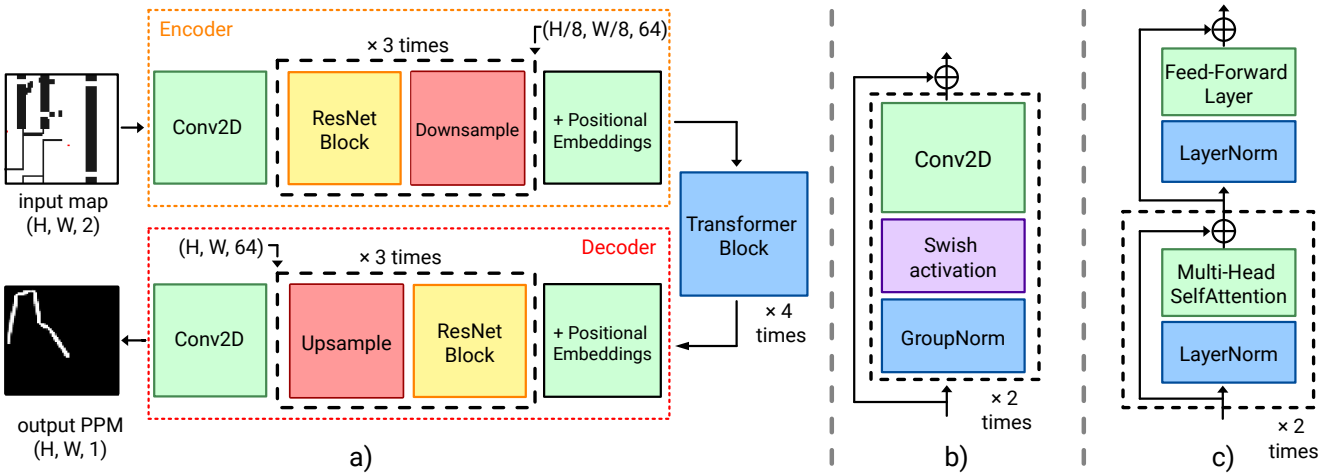

Figure 3: Overview of the neural network architecture. a) Design of the whole model. CNN encoder is used to produce local features which are further fed into the transformer blocks to catch the long-range dependencies between the features. The resulting representation is passed through the CNN decoder to produce output values. b) Architecture of the ResNet block. c) Architecture of the Transformer block.

To create ground-truth cf-values, we utilized uninformed search that starts backwards from the goal and computes true distances to it from any cell (which are straightforwardly converted to the cf-values). Creating the ground-truth samples of pp-values (we refer to such samples as path probability maps, or PPMs) is more involved. Recall that in PPMs, we need to have values of 1 for the cells lying on the shortest path while all other cells should have smaller values. However, numerous shortest paths on 8-connected grids might exist, which differ only in the order of the cardinal/diagonal moves. To break the symmetry we ran Theta* (Nash et al. 2007), an any-angle search algorithm that can be thought of as A* with online path smoothing. Theta* paths are formed of the way points (cells) located at the corners of the obstacles and cells that lie on the straight-line segments connecting the way points. In the resultant PPM, we assigned the values of 1 for such paths. For all other cells, we computed the value that tells us how close the cost of the path through cell n to the one of Theta* is: $cost(\pi(s,g))/(cost(\pi(s,n))+cost(\pi(n,g))$. If this value was greater than or equal to 0.95, we used it as the pp-value; if not, the pp-value was set to 0. As a result, we obtained PPMs that contain paths from $start$ to $goal$ and narrow tunnels (with lower pp-values) around them (see Fig. 1).

Neural Network Architecture

The neural network for learning cf-values and pp-values has the same architecture; however, the input is slightly different. For pp-values, the input contains the grid (as binary image) and the start-goal matrix of the same dimensions, which contains the values of 1 only for start and goal, while all other pixels are zeroes. For cf-values, this matrix contains only one non-zero element: the goal one.

The architecture has three main blocks (see Fig. 3): a convolutional encoder, a spatial transformer, and a convolutional decoder. The convolutional encoder utilizes the well-known ResNet blocks (He et al. 2016) and is aimed at extracting the local features of the pathfidning instance such as corners of the obstacles, narrow passages etc. The transformer leverages the mechanism of self-attention (Vaswani et al. 2017) to establish the global relations between these features (how important is one feature w.r.t. the other). An example may be how important it is that there is a narrow passage in between start and goal. Transformers were originally suggested for text sequences that lack 2D structure. However, in the considered case, this structure is important. To this end, we utilize the positional embedding technique from Visual Transformers (Dosovitskiy et al. 2021; Han et al. 2022). This technique re-arranges 2D feature maps into vectors (before the transformer block) and vice versa (afterward), while preserving the spatial structure. Finally, the transformed feature maps are processed by the convolutional decoder, which provides the final output.

Training and Evaluation

Dataset

We have adopted the TMP (Tiled Motion Planning) dataset that was used in (Yonetani et al. 2021) for empirical evaluation. This dataset is a modification of the MP dataset used in (Bhardwaj, Choudhury, and Scherer 2017). The latter consists of maps with various challenging topologies, such as bugtraps, gaps, etc. Each map in the TMP dataset is composed of the four MP maps. In total, $4,000$ maps of size 64×64 were present in TMP. We further increased the size of the dataset to $64,000$ maps via the augmentation by mirroring and rotating each of the four parts of the TMP maps. Examples are shown in Fig. 4. For each map, we generated 10 problem instances. The goal was chosen randomly, the start was chosen randomly out of the $1/3$ of the reachable nodes that have the highest cost of

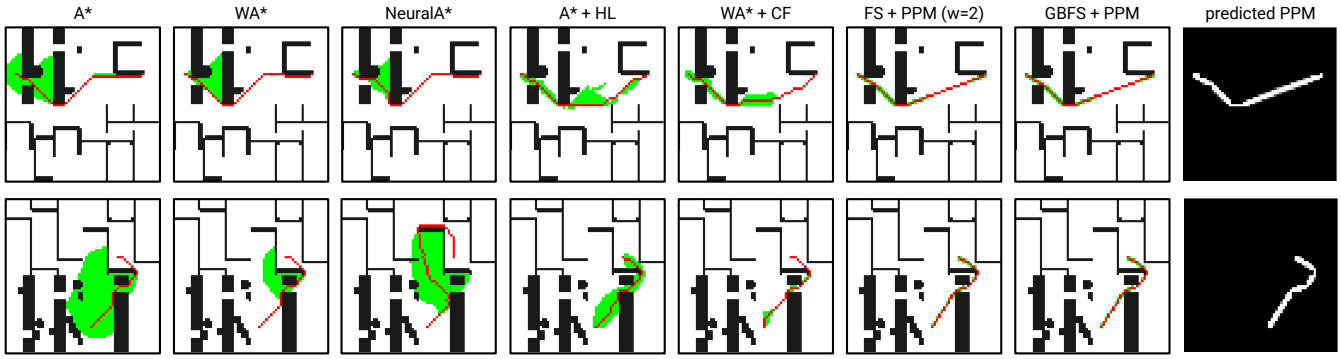

Figure 4: Several examples of the pathfinding results. The expanded nodes are shown in green, and the path in red. The last column shows the predicted PPMs.

the path from the goal. Overall, we generated $640,000$ instances. They were divided in the proportion of 8:1:1 for train, validation, and test subsets in such a way that all augmented versions of the same map were presented only in one of the subsets. Similarly to (Takahashi et al. 2019), we have excluded from the test part of the dataset the instances that are extremely easy to solve, formally, the ones that have hardness less than 1.05. Here hardness is defined as $cost(\pi^*(start, goal))/h(start)$, where h is the conventional cost-to-go heuristic. The closer this value is to 1.0, the easier the instance is, meaning that there is almost no need to bypass the obstacles and the path resembles a straight line.

Planners

We denote the planners[1] proposed in this work as WA*+CF (Weighted A* with the correction factor), FS+PPM (Focal Search with Path Probability Map). We also evaluated a combination of the Greedy Best First Search with PPM, denoted as GBPS+PPM. This planner greedily selects nodes by their pp-values (preferring the ones with the smaller f-values to break ties) and, thus, does not guarantee bounded supoptimality of the solution.

The baselines that we compare against include both standard heuristic search algorithms, A* and WA*, as well as the learnable ones. The latter are represented by the two planners. The first one is Neural A* (Yonetani et al. 2021), the state-of-the-art planner that was shown to notably outperform a range of competitors including the approaches presented in (Bhardwaj, Choudhury, and Scherer 2017; Pogančić et al. 2020) The second is the planner from (Takahashi et al. 2019), which predicts the perfect cost-to-go heuristic and use it in A*. We denote it as A*+HL.

We used the official code of Neural A* and modified it to handle non-uniform move costs. Moreover, we employed our neural network model in Neural A* to provide a fairer comparison (the performance of Neural A* with the original neural network was significantly worse). Similarly, we used our neural network for predicting cost-to-go heuristic in A*+HL. For bounded sub-optimal planners, i.e. WA* and FS+PPM, we set the sub-optimality factor as $w = 2$, as this value provided the best trade-off between path length and computation time for WA*.

[1]The source code of our planners is publicly available at https://www.github.com/AIRI-Institute/TransPath

	Optimal Found Ratio (%) ↑	Cost Ratio (%) ↓	Expansions Ratio (%) ↓
A*	100	100	100
WA*	40.66	103.52 ±4.85	44.43 ±25.92
Neural A*	29.82	104.90 ±6.56	52.30 ±30.47
A*+HL	79.11	100.27 ±0.62	80.50 ±74.40
WA*+CF	**85.40**	100.25 ±1.13	36.98 ±21.18
FS+PPM	82.97	**100.24 ±0.74**	26.36 ±21.08
GBFS+PPM	83.02	100.25 ±0.90	**23.60 ±18.34**

Table 1: Experimental results. Values before ± indicate the average, while after ± – the standard deviation.

Training Setup

To train the neural networks predicting cf-values, pp-values and cost-to-go estimates (for A*+HL), we use the same setup. We train each model using Adam optimizer (Kingma and Ba 2014) for 35 epochs with a batch size of 512 and OneCycleLR learning rate scheduler (Smith and Topin 2018) at a maximum learning rate of 4×10^{-4}. We use L_2 loss for cf-values, pp-values and L_1 loss for the cost-to-go estimates following (Takahashi et al. 2019). It took us 3.5 hours to train each model on NVIDIA A100 80GB GPU.

We trained Neural A* on our training data with the same training setup as in the original work. It took us 16 hours to learn the model on our hardware, four times more compared to cf-/pp-values. This is expected, as Neural A* is trained with the differentiable A* in the loop.

Results

We were primarily interested in the following performance metrics: Expansions Ratio – the ratio of the number of expansions performed by the planner to the number of A* expansions; Cost Ratio – the same ratio but for the solution cost; and Optimal Found Ratio – the ratio of instances optimally solved by the planner.

Table 1 shows the average values and standard error of these indicators for the test dataset, while Fig. 4 highlights

FS+PPM ($w = 4$)	w/ Trans	w/o Trans
Optimal Found Ratio (%)	85.22	61.74
Average Cost Ratio (%)	100.31 $\pm$ 1.58	101.12 $\pm$ 2.19
Average Expansions Ratio (%)	16.06 $\pm$ 11.57	19.65 $\pm$ 17.03
MSE $\times 10^{-3}$	**3.2**	5.3

Table 2: Ablation study results. Values before $\pm$ indicate the average, while after $\pm$ – the standard deviation.

several test instances with the solutions obtained by the evaluated algorithms and the nodes they expand. Clearly, all the learning-based planners are able to generalize to the unseen instances solving them near-optimally while reducing the search effort. In terms of Cost Ratio, the best results were demonstrated by FS+PPM, while the other our planner, WA*+CF, turned out to outperform the competitors in terms of number of the instances solved optimally. The number of reduced expansions varied significantly for all algorithms (see the third column after the $\pm$ sign), and, evidently, in certain cases one of the learnable planners, i.e. A*+HL, managed to expand more nodes than A*. Still, the techniques suggested in this work, in particular, predicting *pp*-values in combination with FS and GBFS, managed to reduce the number of the expansions significantly (up to four times approximately) in numerous cases, as the average value of the Expansions Ratio tells us.

Runtime breakdown We measured the runtime of the compared methods, though it is heavily dependent on the implementation and the hardware. E.g. Neural A* is fully implemented in Python, while our planners feature both Python for neural networks' machinery and C++ for the search. Thus, it is not correct to directly compare their runtimes. To this end, we do not report the runtime of Neural A*. As for the other methods (implemented solely by us), the breakdown of their runtimes are as follows. The prediction step for the batch size of 64 and the native torch float32 type required 9.5ms on Tesla A100 GPU (and 40ms on GTX 1660S). The average CPU time required for further solving this batch of 64 tasks: A* – 155ms, WA* – 77ms, WA*+CF – 60ms, A*+HL – 96ms, FS+PPM – 37ms, GBFS+PPM – 31ms.

Evaluation on Out-of-the-Distribution Dataset

We additionally evaluated all the considered planners on a range of maps that differ significantly in topology and size from the maps of TMP dataset (without any additional training). For extra details and results of this experiment, see the arXiv version of the paper[2]. Overall, the results are similar to the ones reported above: our methods outperform the competitors.

Ablation Study

To demonstrate the importance of using the Transformer block in the neural network, we created a version of the latter that omits this block and is composed only of the convolutional layers (CNN model). We trained this neural network similarly to the baseline model. To compare them, we selected tasks with the hardness exceeding 1.5 as we hypothesize that utilizing transformer is especially useful for non-trivial instances. Quantitative results are presented in Table 2, while qualitative results are given in Fig. 5 (for the sake of space we demonstrate the results for FS+PPM only, as the results for WA*+CF are similar).

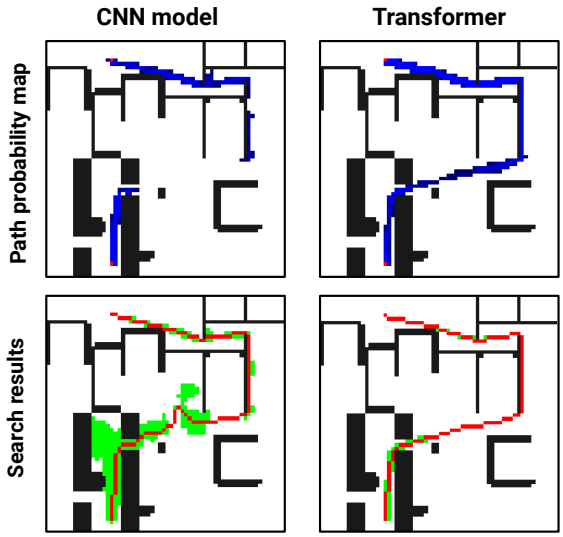

Figure 5: An example showing the difference between the CNN (only) model and the one with the Transformer block.

Clearly, the usage of Tansformer block notably increases the performance across all of the considered metrics, as Table 2 tells us. The last row reports the mean squared error (MSE) between the predictions of the neural network and the ground-truth values.

As Fig. 5 shows, transformer allows us to capture the long-range dependencies between the regions of interest on the map and, consequently, to form a complex and accurate PPM, which aids the search algorithm substantially. The PPM of the CNN model is, however, fragmented, and, as a result, a less natural-looking path is produced while the number of expansions is higher.

Conclusion

In this work, we have explored how state-of-the-art deep learning techniques may aid heuristic search planners in solving grid-based pathfinding problems. We have suggested utilizing a deep neural network composed of both convolutional and attention layers to predict several novel heuristics that can be used in combination with Weighed A* and Focal Search. We have shown empirically that the suggested planners were able to successfully solve challenging problems (unseen at the training phase) and outperform the competitors that include both traditional heuristic search techniques as well as the state-of-the-art learnable approaches.

The avenues for future research include, but are not limited to, planning in 3D, planning with kinodynamic constraints (including sample-based planning), etc.

[2]https://arxiv.org/abs/2212.11730

Acknowledgments

We would like to thank anonymous reviewers for their thoughtful comments that contributed to improving the paper. We would also like to thank Natalia Soboleva, Alexei Artemov, and Vasilii Davydov for their involvement in the preliminary studies on the topic of the paper.

References

Bhardwaj, M.; Choudhury, S.; and Scherer, S. 2017. Learning heuristic search via imitation. In *Proceedings of The 1st Conference on Robot Learning (CoRL 2017)*, 271–280.

Dosovitskiy, A.; Beyer, L.; Kolesnikov, A.; Weissenborn, D.; Zhai, X.; Unterthiner, T.; Dehghani, M.; Minderer, M.; Heigold, G.; Gelly, S.; Uszkoreit, J.; and Houlsby, N. 2021. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. In *Proceedings of The 9th International Conference on Learning Representations (ICLR 2021)*.

Greco, M.; Toro, J.; Hernández-Ulloa, C.; and Baier, J. A. 2022. K-Focal Search for Slow Learned Heuristics. In *Proceedings of The 15th International Symposium on Combinatorial Search (SoCS 2015)*, 279–281.

Han, K.; Wang, Y.; Chen, H.; Chen, X.; Guo, J.; Liu, Z.; Tang, Y.; Xiao, A.; Xu, C.; Xu, Y.; Yang, Z.; Zhang, Y.; and Tao, D. 2022. A Survey on Vision Transformer. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1–1.

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 29th IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2016)*, 770–778.

Janner, M.; Du, Y.; Tenenbaum, J.; and Levine, S. 2022. Planning with Diffusion for Flexible Behavior Synthesis. In *Proceedings of The 39th International Conference on Machine Learning (ICML 2022)*, 9902–9915.

Kingma, D. P.; and Ba, J. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Li, T.; Chen, R.; Mavrin, B.; Sturtevant, N. R.; Nadav, D.; and Felner, A. 2022. Optimal Search with Neural Networks: Challenges and Approaches. In *Proceedings of The 15th International Symposium on Combinatorial Search (SoCS 2015)*, 109–117.

Li, Z.; Chen, Q.; and Koltun, V. 2018. Combinatorial Optimization with Graph Convolutional Networks and Guided Tree Search. *Proceedings of The 32nd Conference on Neural Information Processing System (NeurIPS 2018)*.

Nash, A.; Daniel, K.; Koenig, S.; and Felner, A. 2007. Theta*: Any-Angle Path Planning on Grids. In *Proceedings of The 22nd AAAI Conference on Artificial Intelligence (AAAI 2007)*, 1177–1183.

Pándy, M.; Qiu, W.; Corso, G.; Veličković, P.; Ying, Z.; Leskovec, J.; and Lio, P. 2022. Learning Graph Search Heuristics. In *Proceedings of The 1st Learning on Graphs Conference (LoG 2022)*, 10:1–10:13.

Panov, A. I.; Yakovlev, K. S.; and Suvorov, R. 2018. Grid path planning with deep reinforcement learning: Preliminary results. *Procedia computer science*, 123: 347–353.

Pearl, J.; and Kim, J. H. 1982. Studies in semi-admissible heuristics. *IEEE transactions on pattern analysis and machine intelligence*, (4): 392–399.

Pogančić, M. V.; Paulus, A.; Musil, V.; Martius, G.; and Rolinek, M. 2020. Differentiation of blackbox combinatorial solvers. In *Proceedings of The 8th International Conference on Learning Representations (ICLR 2020)*.

Ramachandran, P.; Zoph, B.; and Le, Q. V. 2017. Searching for activation functions. *arXiv preprint arXiv:1710.05941*.

Rivera, N.; Hernández, C.; Hormazábal, N.; and Baier, J. A. 2020. The 2ˆ k Neighborhoods for Grid Path Planning. *Journal of Artificial Intelligence Research*, 67: 81–113.

Smith, L. N.; and Topin, N. 2018. Super-Convergence: Very Fast Training of Residual Networks Using Large Learning Rates. *arXiv preprint arXiv:1708.07120*.

Soboleva, N.; and Yakovlev, K. 2019. GAN Path Finder: Preliminary Results. In *Proceedings of the 42nd German Conference on AI (KI 2019)*, 316–324.

Speck, D.; Biedenkapp, A.; Hutter, F.; Mattmüller, R.; and Lindauer, M. 2021. Learning Heuristic Selection with Dynamic Algorithm Configuration. In *Proceedings of The 31st International Conference on Automated Planning and Scheduling (ICAPS 2021)*, 597–605.

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 29th International Conference on Automated Planning and Scheduling (ICAPS 2019)*, 764–772.

Tamar, A.; Wu, Y.; Thomas, G.; Levine, S.; and Abbeel, P. 2016. Value iteration networks. In *Proceedings of The 30th International Conference on Neural Information Processing Systems (NeurIPS 2016)*, 2154–2162.

Tan, M.; and Le, Q. 2019. Efficientnet: Rethinking model scaling for convolutional neural networks. In *Proceedings of The 36th International conference on machine learning (ICML 2019)*, 6105–6114.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, Ł.; and Polosukhin, I. 2017. Attention is all you need. In *Proceedings of the 31st Conference on Neural Information Processing Systems (NeurIPS 2017)*.

Yonetani, R.; Taniai, T.; Barekatain, M.; Nishimura, M.; and Kanezaki, A. 2021. Path Planning Using Neural A* Search. In *Proceedings of The 38th International Conference on Machine Learning (ICML 2021)*, 12029–12039.