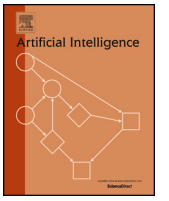
Artificial Intelligence 338 (2025) 104238

Contents lists available at ScienceDirect

Artificial Intelligence

journal homepage: www.elsevier.com/locate/artint



Generative models for grid-based and image-based pathfinding

Daniil Kirilenko [d], Anton Andreychuk [b], Aleksandr I. Panov [a,b,c], Konstantin Yakovlev [a,b,*]

[a] FRC CSC RAS, Vavilova st., 44-2, Moscow, Russia
[b] AIRI, Kutuzovsky pr. 32-1, Moscow, Russia
[c] MIPT, Institutskiy per. 9, Dolgoprudny, Russia
[d] Università della Svizzera italiana, Via Buffi 13, Lugano, Switzerland

ARTICLE INFO

Keywords:
Path planning
Pathfinding
A*
Transformer
Learning heuristics
Planning on images

ABSTRACT

Pathfinding is a challenging problem which generally asks to find a sequence of valid moves for an agent provided with a representation of the environment, i.e. a map, in which it operates. In this work, we consider pathfinding on binary grids and on image representations of the digital elevation models. In the former case, the transition costs are known, while in latter scenario, they are not. A widespread method to solve the first problem is to utilize a search algorithm that systematically explores the search space to obtain a solution. Ideally, the search should also be complemented with an informative heuristic to focus on the goal and prune the unpromising regions of the search space, thus decreasing the number of search iterations. Unfortunately, the widespread heuristic functions for grid-based pathfinding, such as Manhattan distance or Chebyshev distance, do not take the obstacles into account and in obstacle-rich environments demonstrate inefficient performance. As for pathfinding with image inputs, the heuristic search cannot be applied straightforwardly as the transition costs, i.e. the costs of moving from one pixel to the other, are not known. To tackle both challenges, we suggest utilizing modern deep neural networks to infer the instance-dependent heuristic functions at the pre-processing step and further use them for pathfinding with standard heuristic search algorithms. The principal heuristic function that we suggest learning is the path probability, which indicates how likely the grid cell (pixel) is lying on the shortest path (for binary grids with known transition costs, we also suggest another variant of the heuristic function that can speed up the search). Learning is performed in a supervised fashion (while we have also explored the possibilities of end-to-end learning that includes a planner in the learning pipeline). At the test time, path probability is used as the secondary heuristic for the Focal Search, a specific heuristic search algorithm that provides the theoretical guarantees on the cost bound of the resultant solution. Empirically, we show that the suggested approach significantly outperforms state-of-the-art competitors in a variety of different tasks (including out-of-the distribution instances).

* Corresponding author.
E-mail address: yakovlev@isa.ru (K. Yakovlev).

https://doi.org/10.1016/j.artint.2024.104238
Received 1 September 2023; Received in revised form 1 July 2024; Accepted 18 October 2024

Available online 8 November 2024
0004-3702/© 2024 Elsevier B.V. All rights are reserved, including those for text and data mining, AI training, and similar technologies.

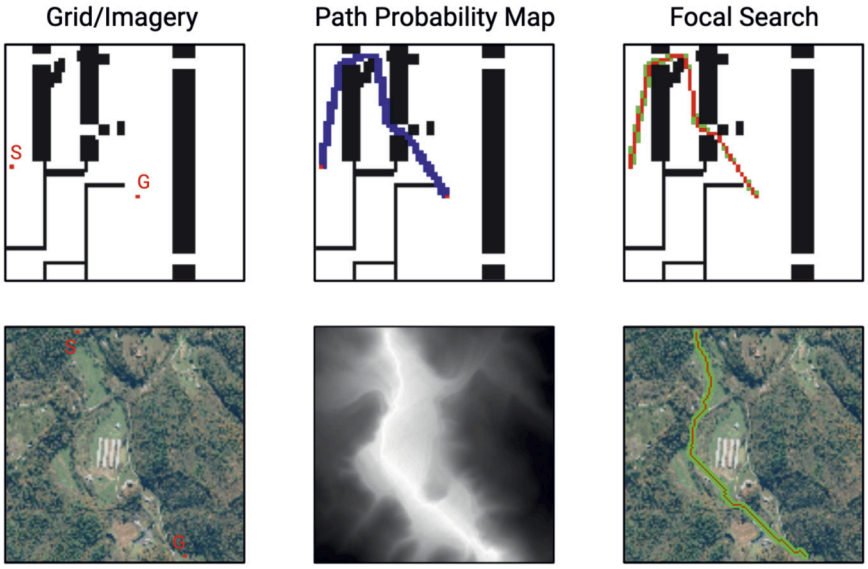


Fig. 1. Rows show two different problem setups investigated in this work: path planning on binary grids with known transition costs (top row) and path planning on images representing uneven terrain, when the transition costs are not known and convectional planning methods are not straightforwardly applicable (bottom row). Columns show the steps suggested to solve the problem. The left column depicts the input, where 'S', 'G' stands for the start and goal location, respectively. This input is processed with the transformer-based neural network and a path probability map is predicted (middle column). The latter is used within a heuristic search algorithm to find a resultant path (right column).

1. 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 [1] are the most widely used graphs for path planning in a variety of applications (robotics, video games, etc.) [2–7]. These grids are composed of the free cells and the occupied cells, corresponding to the obstacles. Indeed, it is allowed to move only from one free cell to the other and the cost of the move (edge in a graph) commonly equals (or is proportional to) the Euclidean distance between the (centers of the) cells.

Sometimes the available information about the initial map does not allow one to differentiate between the blocked and unblocked cells and in general to estimate the costs of the transitions between the grid elements. Consider, for example, a satellite image of the hilly outdoor terrain. Here each pixel (grid cell) corresponds to an area that lies at a certain elevation above the sea level. Thus, the transitions between the cells (pixels that form the map) of different heights should be penalized compared to when the move does not involve the change of elevation. Indeed, if the heights are known, which can be the case when not only a satellite image but also a digital elevation model is available, the costs can be assigned to being proportional to a change in elevation. However, if only the image is available, the costs cannot be assigned directly and path planning becomes challenging. Such or similar setups arise in costly planetary exploration missions [8], construction [9], search and rescue [10] and other applications (Fig. 1).

Generally, path planning on a grid (with known transition costs) is accomplished by a heuristic search algorithm, e.g. A* [11] 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 [12,13]. As grids can be viewed as the binary images, it is appealing to employ the recent advances in convolutional neural networks (CNNs) [14,15] to extract the informative features from the image representations of the pathfinding problems and embed these features into the heuristic search algorithm. For example, [16] suggests learning a perfect cost-to-go heuristic in a supervised fashion. In a more recent study [17], a more involved approach is introduced when a matrix-based A* is proposed and used for learning. Consequently, the deep neural network model is trained end-to-end. That paper does not predict the conventional cost-to-go heuristic, but rather assigns an 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 in which heuristic search can benefit from state-of-the-art deep learning techniques in the context of the grid-based path planning. We consider two settings: with known and unknown transition costs. To deal with both setups, we introduce a novel heuristic proxy, *path probability map*, that *i*) can be successfully

learned from data, *ii*) can be embedded into the powerful search framework of Focal Search [18], that is not only tailored at finding the valid solutions quicker (compared to regular heuristic search) but is also able to preserve strong theoretical guarantees on the cost of these solutions under certain conditions. Intuitively, path probability map tells us which grid cells are more likely to lie on the path from start to goal on the given grid. This information can be utilized as a secondary heuristic for the case when the cost transitions are known (complementing the conventional instance-independent heuristic function like Manhattan distance) or as a primary heuristic for planning in the absence of transition costs (planning on images). Additionally, for the case of path planning on grids with known costs we suggest another heuristic, i.e. the *correction factor*, which is the ratio between the instance-independent heuristic (Manhattan distance, Euclidean distance etc.) and the perfect heuristic. The correction factor can be embedded into the Weighted A* [19] framework to speed up the search.

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 [20] (the so-called transformer architecture). 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 distant relations between them, e.g. "there is a passage between the two regions of interest", keeping the number of trainable parameters relatively low. We also study how the more involved architectures (generative adversarial networks, diffusion models) perform in the context of image-based path planning. Indeed, utilizing these models leads to a better performance; however, this increase in performance is not substantial. The ablation study shows that the major factor that influences the performance of deep learning models is the presence of attention blocks, as without them, the model seems to lack reasoning about the relations of the local features of the input maps that are crucial to path planning.

To evaluate the suggested techniques, we create two comprehensive datasets of planning instances. For path planning with known costs, we extend the dataset previously used in closely related works [17]. For path planning on images, we create a novel dataset, which is composed of the satellite images and the corresponding digital elevation models (that contain the elevation data). Our dataset is based on the `NOAA: Data Access Viewer`[1] collection that contains up-to-date geospatial data on various regions of the Earth's surface. We believe, we are the first to compound and publicly release such kind of dataset (i.e. images with the corresponding elevation data) in a way that allows machine learning practitioners to readily use the data.

Using the datasets, we compare our approach to 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, for path planning on grids with costs, we have been able to reduce the computational effort compared to 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. This is notably better compared to state-of-the-art learnable competitor, i.e. Neural A* [17]. For image-based path planning, the difference between our method and Neural A* is also in our favor both in terms of the computational effort and solution cost.

This paper extends the previously published conference paper on that topic [21] in the following aspects. First, we conduct and report additional experiments for the grids-with-costs domain, i.e. evaluation on out-of-the-distribution dataset (which was only briefly mentioned in the conference paper). Second, we consider planning on images setup, which was not examined in [21] at all. Third, for this setup, we examine and evaluate three additional models that involve more complex training approaches, including Generative Adversarial Network (StyleGAN3 [22]) and Latent Diffusion Model [23]. Forth, we create and release a novel dataset for image-based pathfinding that is based on the real geospatial data and can be easily used by the AI community.

Finally, all source code, model weights, and data collection used in this work are publicly available at https://github.com/AIRI-Institute/TransPath.

2. Related work

The following lines of research can be distinguished that are especially relevant to this work: *i*) techniques that trade off optimality for computational efficiency when solving planning problems via heuristic search; *ii*) utilizing machine learning for solving problems possessing complex combinatorial structure (including, but not limited to graph-based path planning); and *iii*) data-driven learnable approaches to navigation with visual input(s). Next, we overview and discuss these strands of research in more detail.

Trading off optimality for computational efficiency in heuristic search A classical technique for such a trade-off, widely used in practice, is running A* with the heuristic function multiplied by a constant $w \geq 1$ – the so-called weighted A* (WA*) [19]. It guarantees finding solutions with a cost that is at most w times the optimal solution cost. Thus, the solution is *bounded sub-optimal*. When time permits, a series of searches can be performed, each one with the decreased value of w – anytime search [24,25]. Another well-known technique for bounded sub-optimal search is Focal Search [18], whose anytime versions are also known [26]. Focal Search leverages additional heuristic function that complements the conventional search heuristic and is tailored to identifying the search nodes that allow rapid progress towards the goal. More involved algorithms of the same kind include EES [27] and DPS [28], to name a few. Recent results in bounded sub-optimal search are reported in [29].

Other variants to speed up the heuristic search include simultaneous usage of different heuristic functions [30], performing randomized heuristic search [31], etc.

The main difference between the mentioned approaches and our work is that the former assume the heuristic function(s) to be given as the input, while in this paper we infer the heuristics from the instance of the (pathfinding) problem.

[1] https://coast.noaa.gov/dataviewer/.

Machine learning for enhancing combinatorial search Utilizing machine learning for solving problems with complex combinatorial structure (including graph-based path planning) has been getting increased attention recently.

[32] considers a problem of solving combinatorial puzzles from raw visual input. To this end, the authors suggest a method that learns to represent the environment as a latent graph and then invokes a uninformed search algorithm with duplicate states detection. In [33] 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 is evaluated on several problems including pathfinding on the grids, represented as images, where the transition costs are not known apriori. In [34] a learning-based approach for solving certain NP-hard problems is presented that exploits a graph convolutional network to estimate the likelihood of whether a certain vertex of the graph is a part of the optimal solution. In [35], a framework is 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 which node to explore next. Such an approach can be viewed as solving a sequential decision-making problem. Similar approaches are introduced in [36–38]. A special focus on the properties of the learned heuristics, i.e. admissibility, is placed in [39]. Additionally, this work introduces a version of A* search [11] that leverages parallel execution on graphical processing units (GPUs), which are widespread in machine learning computations. Analogous batch-handling techniques for heuristic search are explored in [40].

The papers that are especially relevant to this one are [41,16,17] as they all suggest specific machine learning techniques tailored to grid-based pathfinding. In the former, a generative adversarial (neural) network is proposed to generate the solutions of the pathfinding instances. In [16], a convolutional neural network is used to predict the values of the cost-to-go heuristic. In [17] 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 that 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 one to train the pipeline end-to-end. Neural A* is empirically shown to consistently outperform a range of competitors for grid-based pathfinding. In this work, we use planners from [17] (Neural A*) and [16] as baselines to compare to.

Learnable methods for navigation with visual input A large body of works exists that is dedicated to solving various navigation tasks from image inputs via the learnable approaches (mainly with reinforcement learning) [42–46]. Most of these works study the setup where a visual input comes from a camera mounted on an agent (robot). The navigation tasks with this type of input may vary: exploration [47–49], navigation to a specific object [50–52], or navigation to a goal represented by an image [53]. Recently, with the advances in large language models (LLMs), a number of works have emerged that investigate how these pre-trained LLMs can be utilized for autonomous task and motion planning [54–56]. Differently from the mentioned papers, in this work, we investigate a case where a visual input to a planner represents not a first-person view but rather a bird's-eye (top-down) view of the environment. Indeed, we are not the first to investigate such a setting. For example, [36] suggests value iteration networks that are utilized to learn a navigation policy from Mars terrain images (however, the image size is relatively small, i.e. 16×16). In the previously mentioned work of [33] that focuses on integrating deep neural networks and off-the-shelf combinatorial solvers in an end-to-end trainable pipeline, one of the evaluated setups was path planning on top-down image representations of the agent's environment. The learnable path planning method introduced in [17], i.e. Neural A*, is also capable of pathfinding on image representation of the map. Our study compares our method to Neural A* in a similar setup and shows that the former outperforms the latter.

3. Pathfinding on binary grids

In many practical applications, e.g. in robotics [5–7] or video-games [2–4], the environment in which an agent operates is represented by the so-called occupancy grid [5]. Generally, this is a tessellation of the workspace into the square cells each of which is characterized by a vector of features. In the most basic case, the only feature that is utilized is the traversability of the terrain corresponding to a grid cell. If the terrain is traversable (does not contain any impassable obstacle) the corresponding cell is marked free, and if is blocked, then otherwise. Thus, the grid is binary and the cost of moving between the free cells is equal (or proportional) to the Euclidean distance between them. In this section, we will consider the problem of finding a path on such grids via the combination of the conventional search algorithms and learnable instance-dependent heuristic functions.

3.1. Problem statement

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 $goal$: $\pi = (c_0 = start, c_1, c_2, \dots, c_n = goal)$. A path is valid *iff* all the cells forming this path is 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*. All other paths but the shortest ones are considered *sub-optimal solutions*. Among those, the following

may be of special interest. Assume that a positive real number, $w > 1$, is fixed and provided as an input to a pathfinding algorithm. If the latter guarantees that it will return a path, π^w, whose cost exceeds that of the shortest path by no more than a factor of w, then such path is deemed to be a *bounded sub-optimal solution*: $cost(\pi^w) \leq w \cdot cost(\pi^*)$.

In this work, we are specifically interested in obtaining *i*) valid paths; *ii*) sub-optimal paths (with special interest in bounded sub-optimal ones). The problem of obtaining optimal solutions is beyond our scope.

3.2. Method

3.2.1. Background

*A** One of the most widely used search algorithms capable of solving a large variety of problems including grid-based pathfinding is indeed A* [11]. It is a heuristic search algorithm with provable theoretical guarantees on completeness and optimality.

In brief, 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 referred to as 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 a 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 function h that guides the search focusing the latter towards the goal.

Heuristics The heuristic (function) is called perfect, denoted as h^*, if for every node, its value equals the true cost-to-go: $h^*(n) = cost(\pi^*(n, goal))$. A* search guided by the perfect heuristic expands only nodes that belong to one of the optimal paths. Indeed, the perfect heuristic is instance-dependent. This means that its value differs from one grid map to the other. In practice, the perfect heuristic is unavailable.

The heuristic is considered *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. the Chebyshev distance, the Euclidean distance, or the Octile distance. They can all be efficiently computed in the closed-loop form for any grid cell and their values do not vary from one grid map to the other. Without the loss of generality, in this work, we assume that the Octile distance is used as the heuristic function:

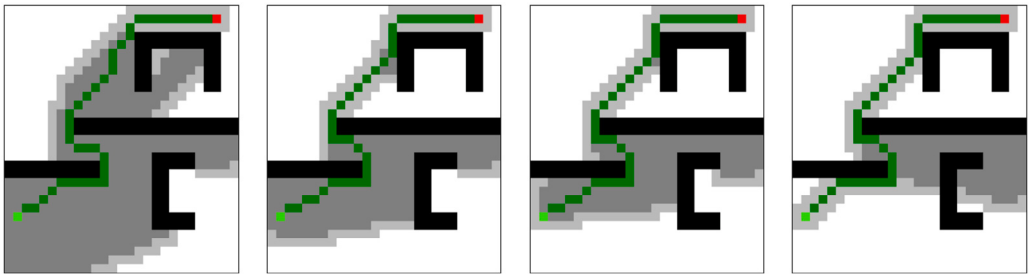
$$h_{octile} = \sqrt{2} \cdot min(\Delta_x, \Delta_y) + 1 \cdot |\Delta_x - \Delta_y| \tag{1}$$

Here $\sqrt{2}$ and 1 are transition costs, associated with the diagonal and cardinal moves (can be arbitrary numbers in general case), and Δ_x, Δ_y are the absolute values of the differences in X-, Y-coordinates of the cells.

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 one can prune any generated successor if it has already been expanded. Still, the number of expansions can be significantly large as depicted in Fig. 2 (on the left). The reason is that the Octile distance, being an instance-independent heuristic, is unaware of the blocked cells and drives the search towards the obstacles via the low f-values of the nodes residing in their vicinity.

*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 sub-optimal solutions w.r.t. w. The effect of weighting the heuristic is illustrated in Fig. 2. As can be noted, the search gets more focused on the goal (gets more greedy) and, as a result, the number of the explored nodes decreases.

Focal search Focal Search (FS) [18] is another technique tailored to lower the number of search iterations while providing the bound on the optimality 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 (the given sub-optimality bound). Technically, $FOCAL = \{n | n \in OPEN, f(n) \leq w \cdot f_{min}\}$. $FOCAL$ is ordered in accordance with the secondary heuristic, h_{FOCAL}, which does not need 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 is 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 sub-optimal solutions. Indeed, the number of search iterations and, thus, the computational efficiency of FS is strongly dependent on h_{FOCAL}.



(a) A* with $w = 1.0$ (b) WA* with $w = 1.5$ (c) WA* with $w = 2.0$ (d) WA* with $w = 10.0$

Fig. 2. Planning a path by A* with the Octile distance as a heuristic function with optional weighting. a) Regular A*. b), c), d) Weighted A*. Grey areas correspond to the explored nodes (dark grey cells – expanded nodes, light grey – generated but not expanded ones).

Algorithm 1: A generic search algorithm. A* executes **black** parts only. WA* is shown in **black** and **red**, Focal Search in **black** and **brown** and our variant of WA* in **black** and **blue**.

Input: Grid Gr, $start$ node, $goal$ node, heuristic function h, sub-optimality factor w, h_{FOCAL} – a secondary heuristic for Focal Search, cf – correction factor (individual weight) for our variant of WA*

Output: path π

$g(start) \leftarrow 0$; $\forall n \neq start\ g(n) \leftarrow \infty$
$OPEN \leftarrow \{start\}$; $CLOSED \leftarrow \emptyset$
while $OPEN \neq \emptyset$ **do**
  $n \leftarrow GetBestNode(OPEN, FOCAL, h_{FOCAL})$
  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**
      $g(n') \leftarrow g(n) + cost(n, n')$
      $f(n') \leftarrow g(n') + w \cdot h(n')/cf(n')$
      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*

Pseudocode Algorithm 1 shows the pseudocode of a generic heuristic search algorithm. Different colors correspond to different variants of the algorithm as explained in the caption.

3.2.2. Search with learned heuristic functions

The general high-level idea is to substitute the instance-independent heuristic function, i.e. the Octile distance, with the one that is instance-dependent, i.e. takes the obstacles into account, and is able to guide the search to effectively circumnavigate these obstacles and reach the goal earlier, thus decreasing the search effort.

Recall that we are interested in two variants of the pathfinding problem. The first one 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 that of the optimal path by more than a factor of w, BSP-PROBLEM.

The solvers that we suggest for VP-PROBLEM and BSP-PROBLEM share their structure. Each of these 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. To solve a BSP-PROBLEM, Focal Search (FS) is used. To solve a VP-PROBLEM, two variants of the search method can be suggested. First, WA* can be used. Second, we can set a sub-optimality bound in FS to infinity and get an (unbounded) sub-optimal solution. In this case, FS becomes similar to the Greedy Best-First Search as it basically selects a node on each iteration based solely on the value of the focal heuristic, h_{FOCAL} (which is predicted by the neural network in our case).

The neural network used in combination with WA* and FS (GBFS) has the same architecture; however, in each case, the output heuristic function is different. Next, we describe these heuristic functions in more detail.

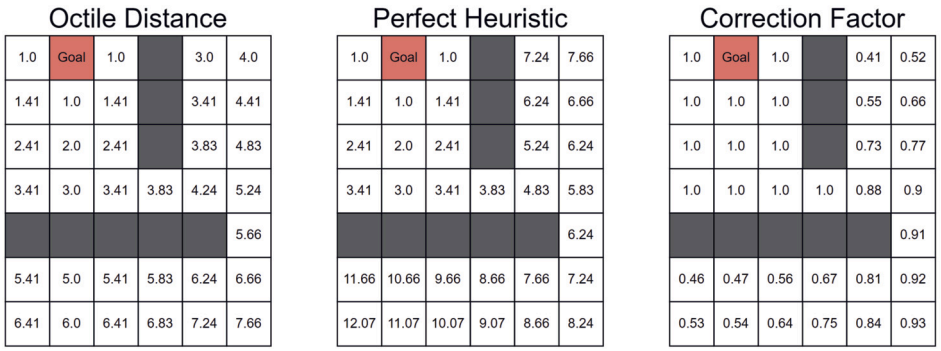


Fig. 3. An example of cf heuristic. Each traversable cell contains the value of the Octile distance, the perfect heuristic and the corresponding correction factor. Only goal location is marked in this example as the start location has no influence for these heuristics.

3.2.3. *Types of the heuristic functions being 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 heuristic, i.e. the Octile distance in the considered case, to the value of the perfect heuristic: $cf(n) = h(n)/h^*(n)$. An example is given in Fig. 3. We suggest plugging the predicted cf-values into the WA* algorithm as shown in Algorithm 1 (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 opposed to a single constant weight. 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 [16]. 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 (and VP-PROBLEM with GBFS, as well). Intuitively, we want from h_{FOCAL} to distinguish the nodes that are likely to yield rapid progress towards 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 $[0, 1]$. We call a set of pp-values for all grid cells PPM (path probability map); an example is shown in Fig. 4.

Learning to accurately 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 unrealistic and, thus, we use the predicted pp-values as h_{FOCAL} values in the FS and GBFS.

3.2.4. *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 [33,17], one might consider another option of learning, i.e. including the search algorithm in the learning pipeline and backpropagating the search error through it (end-to-end learning). 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 in the considered setup (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 preliminary experiments have shown that supervised learning outperforms end-to-end learning.

To create ground-truth cf-values, we utilize an 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 PPMs is more involved. Recall that in a PPM, we are willing to have values of 1 for the cells lying on the shortest path while all other cells should have smaller values. Meanwhile, numerous shortest paths on 8-connected grids might exist, which differ only in the order of the cardinal/diagonal moves. Indeed, one can find all shortest paths and mark the cells on all of them as 1. However, empirically, we find that focusing on a specific shortest path is beneficial – see Appendix for details.

To find such a path we utilize Theta* [57], an any-angle search algorithm that can be thought of as A* with online path smoothing. Theta* paths are formed of the waypoints (cells) located at the corners of the obstacles and cells that lie along the straight-line segments connecting the waypoints. In the resultant PPM, we assign the values of 1 for such cells. For all other cells, we compute the value that tells how close the cost of the path through the cell n is to the cost of the Theta* path:

$$pp(n) = \frac{cost(\pi_{\texttt{Theta*}}(s, g))}{cost(\pi_{\texttt{Theta*}}(s, n)) + cost(\pi_{\texttt{Theta*}}(n, g))}, \tag{2}$$

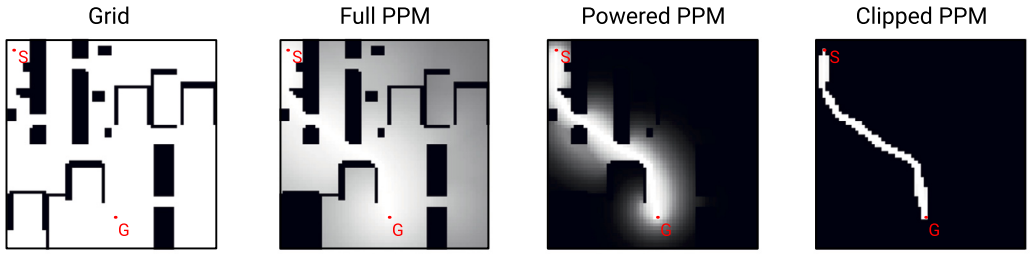


Fig. 4. Path probability maps. From left to right: 1) a problem instance for which PPM is computed; 2) full PPM, the brighter the color is, the closer the *pp*-value to 1; 3) the same PPM but with all *pp*-values raised to the power of 10; 4) the powered PPM after clipping with a threshold of 0.95, i.e., all *pp*-values that are below this threshold are zeroed. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

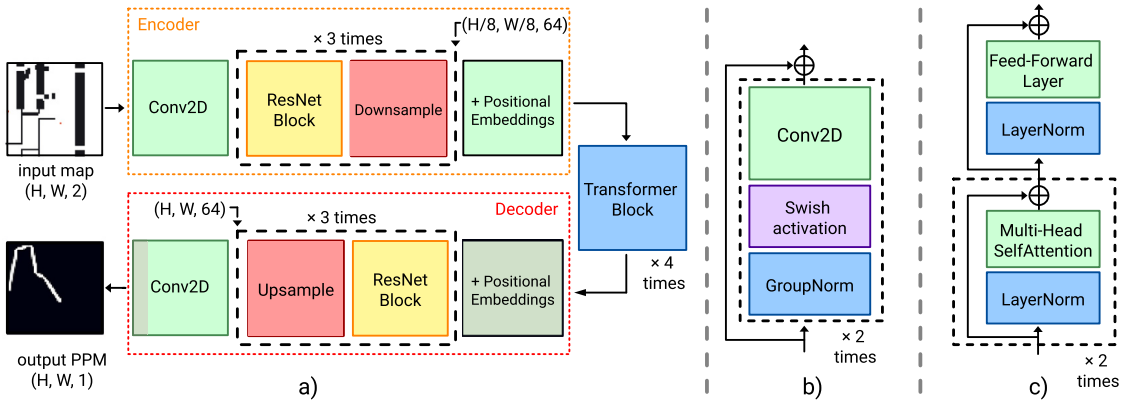


Fig. 5. Overview of the neural network architecture. a) Design of the whole model. CNN encoder is used to produce local features which are further fed into the transformer blocks to catch the long-range dependencies between the features. The resulting representation is passed through the CNN decoder to produce output values. b) Architecture of the ResNet block. c) Architecture of the Transformer block.

where $\pi_{\texttt{Theta*}}(a, b)$ is the Theta* path from a to b.

Technically, to compute PPM, we run two Theta* searches. The one that starts at the *start* cell and is not focused on a certain goal but rather stops when all cells are explored (and the costs of Theta* paths to them are known) and the one that does the same but in the opposite direction (i.e. starts at the *goal* cell). Upon completion of those two searches for any grid cell n, we know both $cost(\pi_{\texttt{Theta*}}(s, n))$ and $cost(\pi_{\texttt{Theta*}}(n, g))$ and can now compute $pp(n)$ for all cells that do not belong to Theta* path itself as prescribed above.

Moreover, during preliminary experiments, we have found out that several other techniques that make the ground-truth PPM more focused, i.e. containing less non-zero values and grouping them around a single path, are beneficial. The first technique is powering the *pp*-values. Example, is shown in Fig. 4, the third pane from the left. Here all *pp*-values are raised to the power of 10. As a result the cells with *pp*-values equal to 1 remain the same while the others decay to zero much intensively. The second technique is clipping the PPM, which is setting the *pp*-value of an element to 0 if it does not exceed a certain threshold – see Fig. 4, the rightmost pane (here the *pp*-values that are lower than 0.95 (after powering) are zeroed).

The empirical results showing that the application of the mentioned techniques is, indeed, beneficial are presented in Appendix. We hypothesize that the reason why it is the case may be that "sharpening and filtering" the PPMs drives the neural network not to waste its capacity on predicting the lower *pp*-values that are, actually, not needed to find a path, but rather forces to predict the *pp*-values only for the regions on the map that are the most needed to reconstruct a path. In the main experiments (reported in the Section 4.4) we used both techniques. First, we raised the *pp*-values to the 10th power and then zeroed all the values that do not exceed the 0.95 threshold (these values were chosen empirically – see Appendix for details).

3.2.5. *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. 5): a convolutional encoder, a spatial transformer and a convolutional decoder. The convolutional encoder utilizes the well-known ResNet blocks [58] and aims to extract the local features of the pathfinding instance such as corners of the obstacles, narrow passages etc. The transformer leverages the mechanism of self-attention [20] 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 the start and the goal. Transformers were originally suggested for text sequences that lack 2D

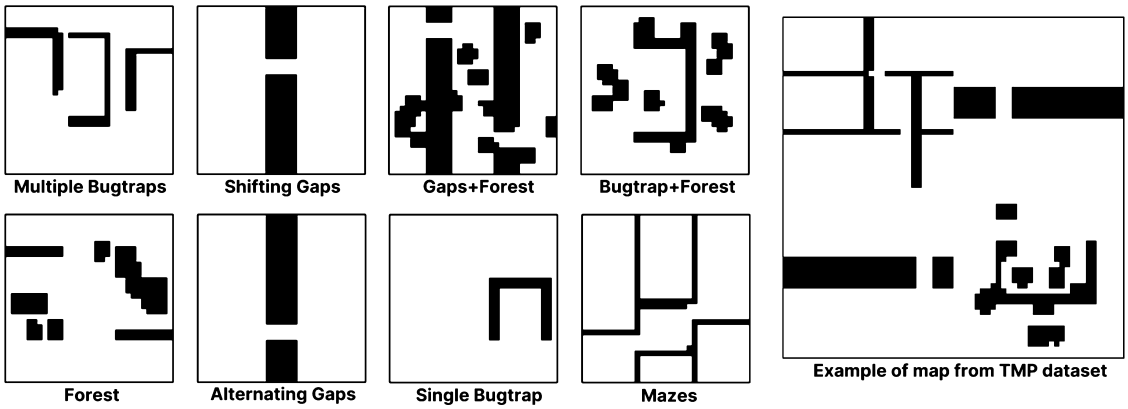


Fig. 6. *Left*) Examples of all types of maps (of size 32×32) presented in the MP dataset. *Right*) An example of the map in the TMP dataset. It is composed of four (different) MP maps; as a result, the size is 64×64.

structure. However, in the considered case, this structure is important. To this end, we utilize the positional embedding technique from Visual Transformers [59,60]. This technique rearranges 2D feature maps into vectors (before the transformer block) and vice versa (afterwards), while preserving the spatial structure. Finally, the transformed feature maps are processed by the convolutional decoder, which provides the final output.

3.3. Empirical evaluation

3.3.1. Dataset

We have adopted the TMP (Tiled Motion Planning) dataset that was used in [17] for empirical evaluation. This dataset is a modification of the MP dataset used in [37]. The latter consists of 32×32 maps with various challenging topologies, such as bugtraps, gaps etc. Each map in the TMP dataset is composed of four MP maps, see Fig. 6. In total, $4,000$ maps of size 64×64 are present in TMP. We further increase 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. The dataset is available online at [61] (*flat* folder). Examples are shown in Fig. 7. For each map, we generate 10 problem instances. The goal is chosen randomly; the start is chosen randomly out of the $1/3$ of the reachable nodes that have the highest cost of the path from the goal. Overall, we have generated $640,000$ instances. They are divided at the ratio 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 [16], 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.

3.3.2. Planners

We denote the planners proposed in this work[2] as WA*+CF (Weighted A* with the correction factor), FS+PPM (Focal Search with Path Probability Map) and GBFS+PPM (Greedy Best-First Search that greedily selects nodes by their pp-values preferring the ones with the smaller f-values to break ties).

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* [17], the state-of-the-art planner that was shown to notably outperform a range of competitors including the approaches presented in [37,33] The second is the planner from [16], which predicts the perfect cost-to-go heuristic and use it in A*. We denote it as A*+HL.

We use the official code of Neural A* and modify it to handle non-uniform move costs (originally, the costs of both diagonal and cardinal moves have been set to 1 in Neural A*). Moreover, we have 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 have used our neural network for predicting cost-to-go heuristic in A*+HL. For bounded sub-optimal planners, i.e. WA* and FS+PPM different suboptimality factors might be used. In Appendix we report the results across a variety of them (ranging from 1.01 to 10). In the main body of the text we report the results for $w = 2$, as this value provides the most balanced trade-off between path length and computation time for WA*.

3.3.3. 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 the Adam optimizer [62] for 35 epochs with a batch size of 512 and OneCycleLR learning rate scheduler [63] at

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

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

	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**

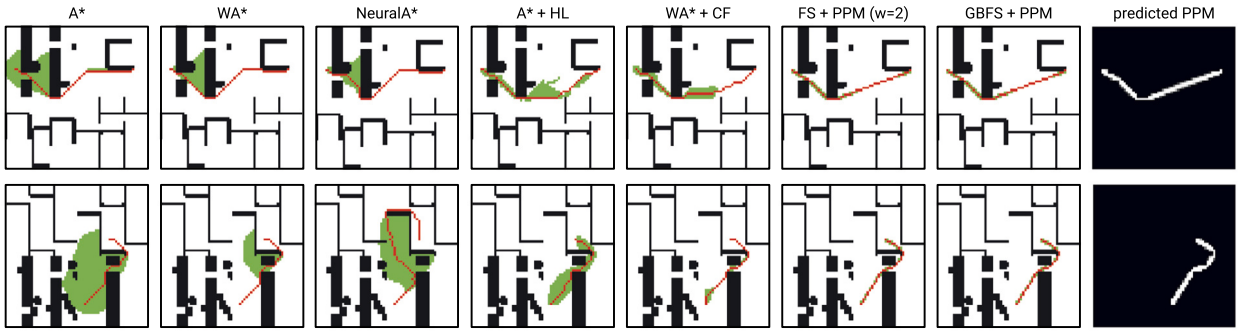


Fig. 7. Several examples of the pathfinding results. The expanded nodes are shown in green, and the path in red. The last column shows the predicted PPMs.

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 [16]. It has taken us 3.5 hours to train each model on NVIDIA A100 80 GB GPU.

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

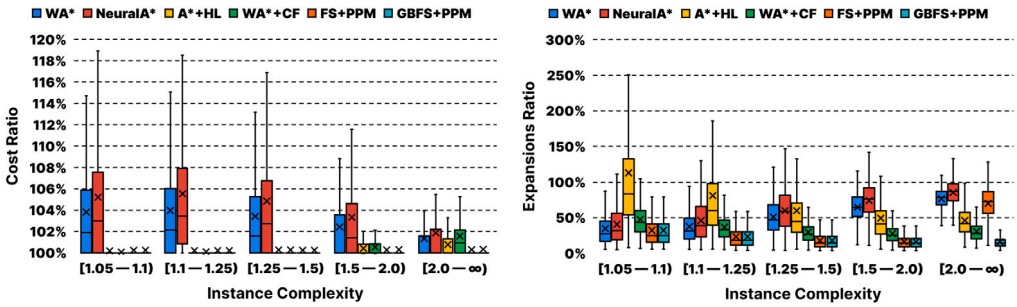
3.3.4. Results

We are primarily interested in the following performance measures: 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. 7 highlights 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 unseen instances solving them near-optimally while reducing the search effort. In terms of Cost Ratio, the best results have been demonstrated by FS+PPM, while the other our planner, WA*+CF, turns out to have outperformed the competitors in terms of the number of instances solved optimally. The number of reduced expansions varied significantly for all algorithms (see the third column after the ± 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.

A more detailed overview of the results is presented in Fig. 8. Here the box-n-whisker plots for the instances grouped together based on their hardness are presented. As one can note, the cost ratio of WA* and NeuralA* decreases when the instances get harder. However it is not the case for the other planners. For very hard instances (with hardness exceeding 2), WA*, NeuralA*, A*+HL, WA*+CF demonstrate similar results. FS+PPM and GBFS+PPM are indeed the ultimate winners in terms of cost ratio. More importantly, their performance does not seem to be tied to the hardness of the pathfinding instances they are facing.

As for the expansions ratio, the following trends can be observed. The performance of WA* and NeuralA* degrades as the hardness grows. Contrary, the performance of the other approaches does not change or even improves with growing hardness. The notable exception is FS+PPM which performance is very good when the hardness is lower than 2 and is inferior when the hardness exceeds this mark. This can be explained by the nature of the Focal Search, which is provided with the sub-optimality threshold of 2 in our experiments. FS hits the sub-optimality bound on the hard instances and is forced to expand redundant nodes with the lower f-values to rise the value of f_{min} in $OPEN$ (which is needed to continue to progress towards the goa)l. Indeed, if the sub-optimality factor was set to a higher value, the drop of the performance would not be thus pronounced, which is confirmed by the results of GBFS+PPM.



(a) Cost ratio. (b) Expansions ratio.

Fig. 8. Cost and expansions ratios w.r.t. the hardness of the test instances.

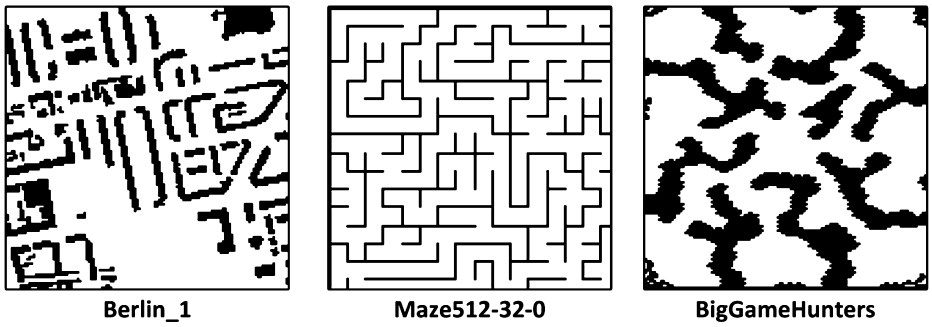


Fig. 9. Maps comprising the out-of-distribution dataset.

Table 2
Experimental results on out-of-distribution dataset. Values before ± indicate the average, while values after ± denote the standard deviation.

	Optimal Found Ratio (%) ↑	Cost Ratio (%) ↓	Expansions Ratio (%) ↓
A*	100	100	100
WA*	8.13	104.31 ± 4.76	57.52 ± 30.72
Neural A*	3.24	107.10 ± 6.77	63.08 ± 34.63
A*+HL	**29.02**	**101.90 ± 2.72**	148.94 ± 136.95
WA*+CF	10.61	106.10 ± 5.59	63.64 ± 36.31
FS+PPM	18.66	105.62 ± 5.61	55.06 ± 39.57
GBFS+PPM	18.59	106.12 ± 6.54	**54.33 ± 47.24**

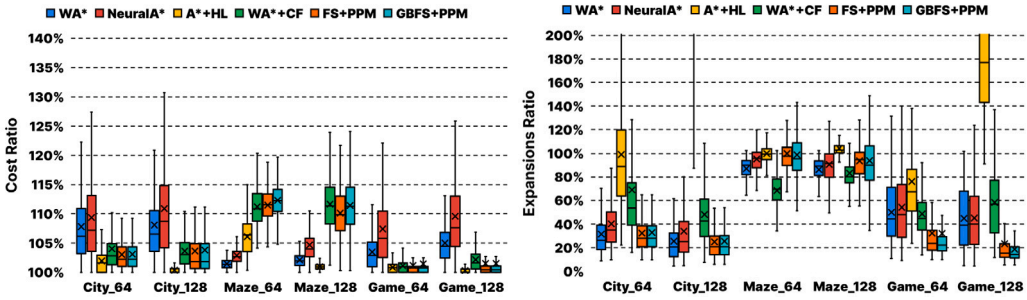
3.3.5. Runtime breakdown

We have 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, directly comparing their runtimes would be incorrect. 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 is as follows. The prediction step for the batch size of 64 and the native torch float32 type required 9.5 ms on Tesla A100 GPU (and 40 ms on GTX 1660S). The average CPU time required for further solving this batch of 64 tasks: A* – 155 ms, WA* – 77 ms, WA*+CF – 60 ms, A*+HL – 96 ms, FS+PPM – 37 ms and GBFS+PPM – 31 ms.

3.3.6. Evaluation on out-of-the-distribution dataset

Besides the main dataset, we have also created an out-of-distribution dataset that consists of three different maps taken from the MovingAI benchmark [64]: `Berlin_1` (City), `maze512-32-0` (Maze) and `BigGameHunters` (Game), see Fig. 9. Each of these maps is scaled to two different sizes: 64×64 and 128×128. There are randomly generated $1,000$ instances per each map and size. As before, the instances with hardness less than 1.05 have been excluded from the experiments.

We have used this dataset to evaluate how the learnable planners suggested in the paper (and their competitors) perform when solving instances that are substantially different in topology and size from the ones used for learning. None of the maps from this dataset has been used for training, i.e. these maps were presented to the planners only at the test phase.



(a) Cost ratio. (b) Expansions ratio.

Fig. 10. Cost and expansions ratios w.r.t. the hardness of the test instances (out-of-the-distribution dataset).

Table 3
Ablation study results. Values before ± indicate the average, while values after ± denote the standard deviation.

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

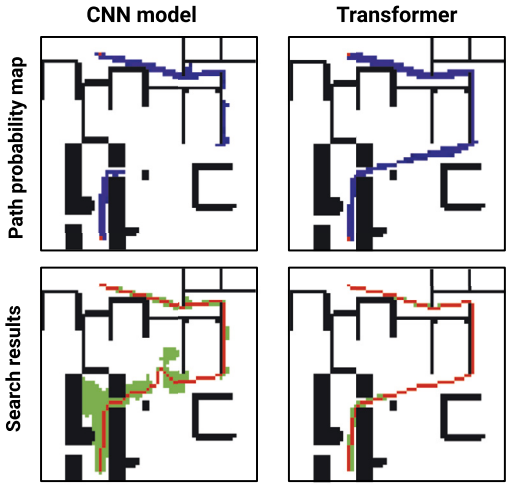


Fig. 11. An example showing the difference between the CNN (only) model and the one with the Transformer block.

Table 2 presents the aggregated results. As expected, the performance of all the learnable planners is worse compared to the main experiments. Still, the best results in terms of expansion ratio are achieved by one of our planners, i.e. GBFS+PPM (the result of FS+PPM is very close).

The detailed box-and-whiskers plots for cost and expansions ratios for this experiment are depicted in Fig. 10. As one can note, the results for the Maze map differ significantly, especially from the expansions ratio perspective. This can be explained by the fact that this type of maps is extremely hard to solve due to the arrangement of the blocked areas that do not form separate obstacles that can be circumnavigated. For the two other type of maps, however, the results are similar to the ones observed on our main dataset, i.e. FS+PPM and GBFS+PPM are able to reduce the number of expansions up to a factor of 4 while producing only a slight overhead in terms of solution costs.

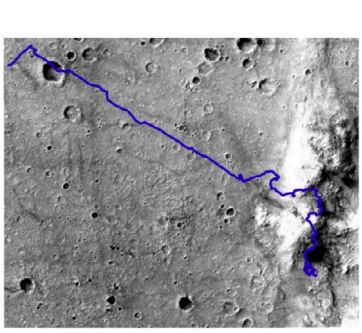
Overall, the observed results support the claim that the suggested approaches have strong generalization capabilities and perform well on the out-of-the-distribution instances (at least until the topology of such instances is too complex, like in the case of mazes).

3.3.7. Ablation study

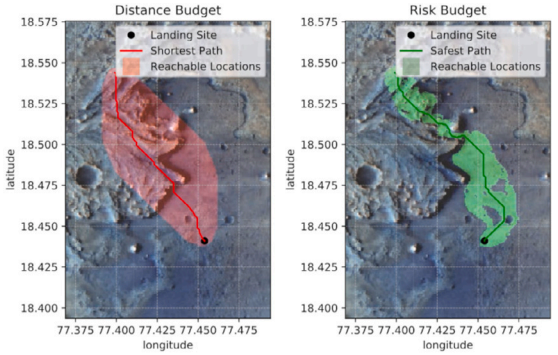
To demonstrate the importance of using the Transformer block in the neural network, we have created a version of the latter that omits this block and is only composed of the convolutional layers (CNN model). We have trained this neural network similarly to the baseline model. To compare them, we select tasks with the hardness exceeding 1.5 as we hypothesize that utilizing the transformer is

Table 4
Results of the evaluation of a greedy algorithm that reconstructs a path by iteratively picking the cell with the maximal *pp*-value in PPM instead of a systematic search.

Success Rate (%)	Cost Ratio (%)	Iterations (Expansions) Ratio (%)
87.80	143.07±72.86	27.59±34.34



(a) Rough Martian terrain and an actual traverse of the exploration rover. Image from [8].



(b) The shortest (left) and the safest (right) paths for Martian rover. Image from [65].

Fig. 12. On path planning for Martian exploratory missions.

especially useful for non-trivial instances. Quantitative results are presented in Table 3, while qualitative results are given in Fig. 11 (for the sake of space, we only demonstrate the results for FS+PPM as the results for WA*+CF are similar).

Clearly, the usage of the Transformer block noticeably increases the performance across all of the considered metrics, as is indicated by Table 3. The last row reports the mean squared error (MSE) between the predictions of the neural network and the ground-truth values.

As Fig. 11 shows, the 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 substantially aids the search algorithm. 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.

3.3.8. Pathfinding with PPMs without systematic search

We have also tried to use the PPMs, provided by the neural network, directly to reconstruct a path without running Focal Search. That is, after feeding the problem instance into the neural network and obtaining the predicted PPM we run the following algorithm. At each iteration we examine the neighbors of the current cell and pick the one with the maximal *pp*-value, add it to the path and transition to this cell. We begin with the start cell and end either when the goal cell is reached or when at some iteration we are not able to pick a cell that has not already been added to the path. The results of such experiment are shown in Table 4.

The first column of the Table 4 shows the percentage of the successfully solved instances. As one can note 12.2% instances remained unsolved. Moreover, as indicated in the second column, the costs of the resultant paths are significantly higher compared to both the optimal ones and the ones obtained by running FS+PPM or GBFS+PPM (recall that the latter planners find the paths with cost ratio of 100.25% on average). The last column shows how many iterations (on average) it took the suggested greedy algorithm to reconstruct paths compared to the number of iterations (expansions) used by A*. Indeed, the former performs notably less iterations compared to the latter and is on par with FS+PPM, GBFS+PPM whose ratios were (recall Table 1) 26% and 23% on average respectively.

Overall, one can summarize that the straightforward approach, that does not embed the predictions of the neural network into an involved search framework, is much less effective. This confirms that both predicting PPMs and utilizing Focal Search that leverages these PPMs is necessary to effectively solve challenging pathfinding instances.

4. Pathfinding on image representations of digital elevation models

In various practical setups, one may need to find a path on a map encoded not as a binary grid (as before) but rather as an RGB image. For example, in [8,65] such setup is considered for complex planetary exploration missions, more precisely – for exploring the surface of Mars with the semi-automated rovers (wheeled mobile robots). Indeed, Mars terrain is rough and one should be very careful when planning paths over such terrain. A general approach to path planning, according to [8] is that, first, one or more global paths are constructed (automatically or by a specialist) based on the satellite images of the Martian terrain and additional data. Then, the rover uses local sensing and local motion planning to follow the most preferred global path or switch to another one if necessary. Fig. 12a depicts an example of the real traverse of the exploratory rover over the rough Martian terrain.

Indeed planning a path in such scenarios should respect the safety constraints. One of the major aspects to be taken into consideration is avoiding the sharp difference in the elevation along the (global) path. Example (from [65]) is shown in Fig. 12b. Here the shortest path is not the safest one as it, presumably, involves traversing a sloppy area.

If, besides the image representation, the digital elevation model (DEM) of the environment is available, one may, first, design a (transition) cost function that takes the elevation into account and, second, run a search-based path planning algorithm, e.g. A*, on DEM. However, DEMs are often unavailable (due to the high costs associated with their construction) and, thus, planning has to be carried out only relying on the image map of the environment. This is especially the case for the multi-robot navigation setups, involving unmanned ground vehicles and unmanned aerial vehicle, when the latter take images of the outdoor terrain and the former plan their paths based on these images – see [66,10] for example.

Indeed, conventional path planning algorithms are not straightforwardly applicable here as the transition costs associated with moving from one pixel to the other are unknown. This is exactly the setup we wish to investigate and solve with the previously suggested approach. I.e., we wish to *i*) extract the valuable data from the input RGB image (conditioned on a specific start and goal locations) and represent it in a way suitable for heuristic search algorithms; and *ii*) run the latter to construct the sought path.

4.1. Problem statement

Consider a robot navigating an uneven outdoor terrain and a $m \times n \times 4$ tensor, where the first three channels comprise an $m \times n$ RGB image of that terrain (top-down view) and the last channel stores the height data, i.e. the $m \times n$ matrix containing the elevation values. The latter is commonly known and referred to as DEM.

Two adjacent pixels comprise a transition whose cost is defined as follows:

$$cost(p_1, p_2) = dist(p_1, p_2) + \alpha |DEM(p_1) - DEM(p_2)| \tag{3}$$

Here $dist$ is either 1 or $\sqrt{2}$, depending on whether the pixels are cardinally or diagonally adjacent, $DEM(p)$ is the height of the pixel p and α is the user-specified parameter that is used to adjust the importance of elevation change and is also needed for proper scaling when spatial and height resolution are not aligned. Please note that the transition from a pixel to any of its eight neighbors is possible; however, different transitions incur different associated costs. Intuitively, the transitions that result in sharp changes of elevations are penalized more, compared to the transitions when the elevation changes slightly (the penalty can be adjusted with α).

The task is to find a path, i.e. a sequence of transitions, from the dedicated start area (pixel) to the goal one. While we do not aim to find optimal or bounded sub-optimal paths, the lower-cost paths are, indeed, preferable. Moreover, we assume that at the test time we do not have access to DEM, i.e. the path is to be constructed solely from the image input. Still, a representative dataset of the aligned RGB+DEM pairs is available for training/learning beforehand.

4.2. Dataset

To solve the described problem, we need a comprehensive dataset consisting of tens of thousands of RGB images aligned with their DEMs. Currently, there exists a very limited number of datasets of that kind. For example, in [67] a dataset of 248 samples is presented. All these samples represent fragments of the typical agricultural landscapes. Another dataset is the dataset of Martian landscape fragments – HiRISE DTM [68]. It is available online[3] and consists of about 1,000 samples, each of which requires non trivial processing to actually get an aligned RGD-DEM pair.

To this end we have created a novel rich dataset of the properly aligned RGB-DEM samples, that are not synthetically generated but rather represent real landscapes. The principal source used to compile our dataset is `NOAA: Data Access Viewer`,[4] an extensive online toolbox for processing and visualising the geospatial data. We have used a vast region (approximately 128×180 kilometers) of North Carolina characterized by a relatively hilly terrain to obtain the data. The resolution of the collected data is 10 m per pixel, which provides a sufficiently detailed representation of the landscape's physical features. The size of the original RGB image (and DEM) is $12,875 \times 18.000$ pixels. We have used this RGB-DEM pair to create a total of the $18,316$ accurately aligned samples. Our dataset is available online at [61] (*dem* folder). Fig. 13 depicts the source RGB image and the corresponding DEM.

We process the source image as follows. First, we slice it into the tile of square pieces. We use three different tile scales, i.e. 256×256, 512×512, and 1024×1024 pixels; see Fig. 14. Our intent for having different scales is that we want our model to learn the features that are robust to planar scale, but rather capture the way the elevation changes. We believe that scale diversity is needed to encourage the model to identify and prioritize key terrain features regardless of the spatial context they appear in, thereby enhancing its generalization capabilities.

Following the slicing, all pieces are downsampled to a size of either 128×128 or 64×64 pixels using bilinear interpolation (Fig. 14, right). This step is undertaken to ensure compatibility with the neural network's input size constraints and to expedite the training process. Despite the inevitable reduction in image resolution, the downscaled images retain an acceptable level of detail necessary for solving pathfinding tasks. In addition to downsampling, we apply a normalization procedure to each DEM. This normalization step is needed to suit the operational requirements of the neural networks and to ensure that their training is effective. Normalization is

[3] https://www.uahirise.org/dtm/.
[4] https://coast.noaa.gov/dataviewer/.

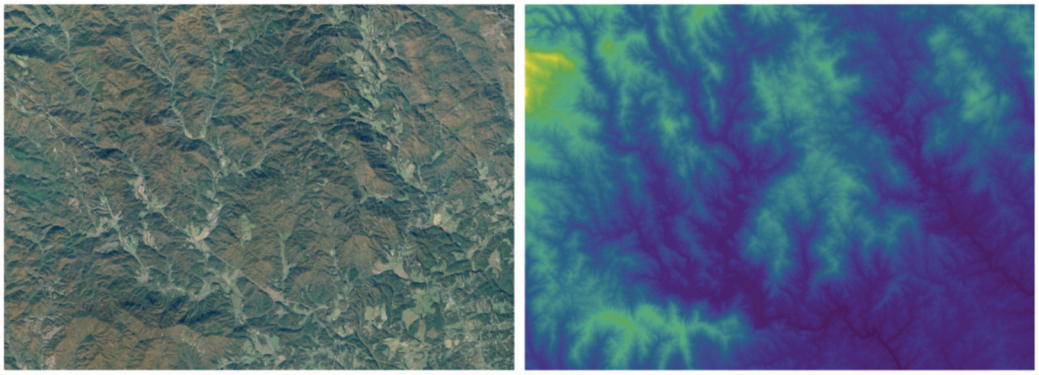


Fig. 13. Visualization of original imagery and a corresponding elevation map. This figure presents the original high-resolution satellite imagery juxtaposed with the corresponding DEM for a selected region in North Carolina. The left panel displays the optical imagery, while the right panel visualizes the respective terrain elevation map, demonstrating the complexity and diversity of the landscape features captured in our dataset.

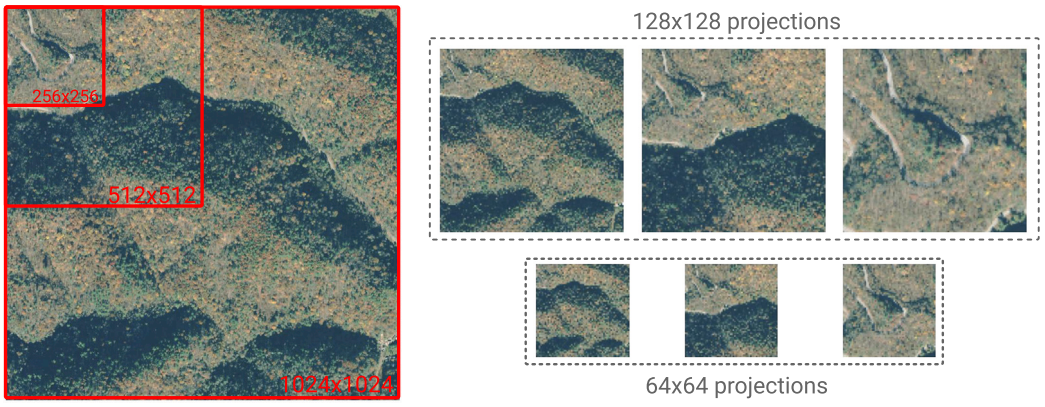


Fig. 14. Left: Examples of the map slices that form the base of our dataset (the size of each slice is either 1024×1024 or 512×512, or 256×256. Right: The corresponding downscaled projections (128×128, 64×64). Despite the reduction in resolution, essential pathfinding details are preserved.

performed by subtracting the minimum value of the DEM from each pixel on the map, effectively translating the range of elevation values to start at zero. Subsequently, we scale the resulting values by dividing each pixel by the maximum value in the DEM.

$$\text{dem}_i = \text{dem}_i - \min(\text{dem}), \quad \text{dem}_i = \frac{\text{dem}_i}{\max(\text{dem})} \tag{4}$$

As a result, we end up with the scale invariant DEMs, i.e. the height of any pixel in any DEM belongs to $[0, 1]$ range.

To further enhance our dataset, we employ data augmentation through the rotation of image slices. This technique is applied with the goal of expanding the quantity and diversity of our training data, thereby reducing overfitting. Image rotation is a particularly useful augmentation method in our context as it helps bolster the model's robustness against varying orientations, a common challenge in real-world image-based tasks.

For each map sample of our dataset, we create ten distinct pathfinding tasks. The generation of these tasks is performed through a uniform random sampling, where the start and goal positions for each instance are randomly assigned. By creating multiple tasks per map, we are able to increase the size of the dataset (without requiring additional geographical data).

The resultant dataset is comprised of $18,316$ maps (pairs of the aligned RGB-images and DEMs) and $183,160$ problem instances. We use 8-1-1 train-val-test split (as before). The dataset is publicly available in our repository. We believe this is the first dataset of such kind that *i*) is based on the real geo-spatial data; *ii*) contains accurately aligned RGB-DEM pairs; and *iii*) is large enough for learning-based methods.

4.3. Method

Our approach for the image-based pathfinding is the same as for the grid-based pathfinding, i.e. it is centered on the utilization of the deep neural networks that take an RGB image as the input and provide an informative output that can be used by a search-based pathfinding algorithm. We explore two options for such output. The primary option is, as before, to reconstruct the PPM, and use if further to guide the search in GBFS. The secondary option is to reconstruct DEM and run A* on it. The problem with this variant is that DEMs are scale-invariant, i.e. the minimum height in each DEM is 0 and the maximum height is 1 while the real absolute values

(measured in meters) are not known. Thus, one needs to properly set the α scaling coefficient in (recall Eq. 3), which is problematic. In other words, one needs to properly guess the absolute difference between minimal and maximal height for an input image. Please note that in the case of reconstructing PPM and invoking GPBS on it, this problem does not arise, which points to this method being preferable. Please also note that in both cases, i.e. PPM+GBFS and DEM+A*, the resultant paths are not guaranteed to be optimal or bounded sub-optimal as the real transition costs are not known (this is why running FS with a fixed suboptimality bound on PPM does not make sense and we opt for GBFS only).

To learn both elevation maps and path probability maps, as before, we rely on supervised learning. In the former case, the ground-truth samples, i.e. DEMs, are directly available in the dataset. In the latter case, the ground-truth PPMs are constructed by us using the DEM data.

To construct ground-truth PPMs, we generally follow the same approach as before. First, we identify a single path on each DEM that we want to focus on, i.e. the one which is not a geometrically shortest one by rather the one with the lowest cost (that takes the elevation change into account). Then we assign the *pp*-values of 1 to the pixels forming this path and, to the other pixels, we assign the values that are proportional to the costs of the least-cost paths from these pixels to the pixels of the designated path. Technically, the implementation is as follows.

Obtaining ground-truth PPMs As before, we run one unfocused search from the *start* and another one from the *goal* go get the costs of the (least-cost) paths to any pixel forming the DEM. 'Unfocused' means that we stop the search not when reaching a particular pixel but rather when all the pixels are explored. We use A* for the search (not Theta*, because the latter is not tailored to pathfinding on an uneven terrain where the elevation changes from one pixel to the other). The forward pass of A* also gives us a path from *start* to *goal*, $\pi(s,g)$, that will be marked as the most desirable in PPM. Next, we run another unfocused A* with its OPEN list initialized with the pixels belonging to $\pi(s,g)$. This gives us, for every pixel, the value that shows how far it is from the desired path, with taking the costs associated with change in elevation into account. Finally, the *pp*-value of every pixel n is computed by:

$$pp(n) = \frac{cost(\pi(s,g))}{cost(\pi(s,n)) + cost(\pi(n,g)) + cost(\pi(n,\pi(s,g)))} \tag{5}$$

Consequently, all the values of PPM are in $(0,1]$ range, 1 are assigned to the pixels forming a single least-cost path and the higher the *pp* value is, the closer (in terms of cost) it lies to this path. Notably, we do not use clipping for PPMs (as experimentally this did not provide any gain). In Appendix we elaborate on this phenomenon in more details.

4.3.1. Models

To learn PPMs and DEMs, we rely on supervised learning. Primarily, we use the same autoencoder model as before (see Section 3.2.5), but augmented with two additional transformer layers (so the number of parameters raises from 1M to 1.2M). The training setup is identical to the one used before (see Section 3.3.3).

Moreover, we also examine how the more advanced image-to-image translation methods [69] may perform when solving the problem at hand. Specifically, we examine StyleGAN3 [22] and Latent Diffusion [23]. These models are recognized for their effectiveness in handling high-resolution and complex images. Fig. 15 offers a schematic overview of these models compared to the autoencoder baseline.

StyleGAN3 Generative Adversarial Networks (GANs) are a class of machine learning models designed to produce new data that mimic the distribution of a given training dataset. GANs are composed of two primary components: a Generator (G) and a Discriminator (D). The generator's role is to create new data instances, while the discriminator's role is to differentiate between instances from the real dataset and the ones created by the generator. The ultimate goal is to have a generator capable of producing data that the discriminator cannot differentiate from the real dataset.

The training process of a GAN involves an adversarial game, where the generator and the discriminator are trained simultaneously. The discriminator is updated to better distinguish between real (x) and noise (z) generated data, while the generator is updated based on how well the discriminator was able to classify its output as real or fake. This simultaneous training process is often likened to a two-player min-max game, where the discriminator aims to maximize its accuracy (minimize its loss), while the generator aims to maximize the discriminator's error (maximize its gain). Over time, this adversarial process leads the generator to producing increasingly realistic data.

Formally, such a minimax problem can be written as follows:

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{\text{data}}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z)))], \tag{6}$$

where the p_{data} is the distribution of a given training dataset and $z \sim p_z(z)$ are the Gaussian noise samples.

In the context of our work, we use StyleGAN3 [22], a state-of-the-art GAN, to perform a complex image-to-image translation task. Our goal is to generate elevation and path probability maps from satellite imagery. This task can be considered as mapping one data distribution (satellite imagery) to another (elevation and path probability maps). The inherent capability of GANs to learn and mimic complex data distributions makes them suitable for this task. The advanced features of StyleGAN3, such as an alias-free generator and improved conditioning schemes, offer us the potential to generate high-quality output maps that can effectively aid in the pathfinding process.

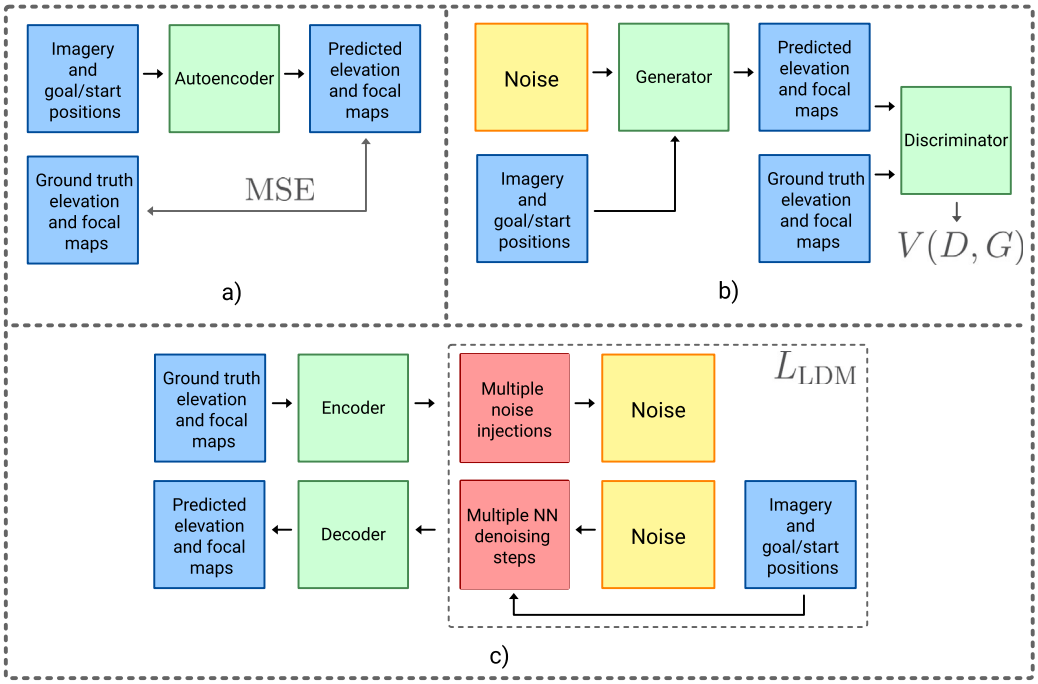


Fig. 15. Comparative overview of the three image translation approaches used in our work. Each method exhibits unique processing structures and mechanisms. (a) The Autoencoder transforms input RGB satellite imagery into predicted elevation maps and PPMs through an encode-decode process. (b) The StyleGAN3, a representative of Generative Adversarial Networks, leverages a noise vector and an adversarial training process to generate the outputs. (c) The Latent Diffusion Model uses a forward and reverse diffusion process for the generation task.

Latent diffusion Latent Diffusion Models (LDMs) belong to a class of probabilistic models that have recently yielded significant progress in high-resolution image synthesis. These models are a variant of Diffusion Models (DMs) designed to learn data distribution $p(x)$ by gradual denoising a normally distributed variable. The training process of diffusion models involves a series of steps, each of which attempts to denoise the data slightly using timestamp-conditioned autoencoder $\epsilon_\theta(x_t, t)$, moving it closer to the true data distribution.

LDMs enhance this process by introducing a latent space z and specific encoder ($z = \mathcal{E}(x)$) and decoder ($x = \mathcal{D}(z)$), which allows for a more efficient representation of the data. The training process of LDMs is divided into two stages. The first one, the universal autoencoding stage, involves training the encoder and decoder models, which are trained only once and can be reused for multiple DM training or to explore possibly completely different tasks. The second stage involves training the actual generative model ϵ_θ, which learns the semantic and conceptual components of the data.

The objective function for LDMs, derived from the Evidence Lower Bound (ELBO), involves a sum over the timesteps of the diffusion process. It aims to minimize the Kullback-Leibler divergence between the posterior distribution and the prior distribution, effectively making the posterior distribution as close as possible to the prior. This is achieved by minimizing the reconstruction error between the original data and the generated data, weighted by the change in the signal-to-noise ratio from one-time step to the next. The simplified version of this objective looks as follows:

$$L_{\text{LDM}} = \mathbb{E}_{\mathcal{E}(x),\epsilon\sim\mathcal{N}(0,I),t}\left[\left\|\epsilon - \epsilon_\theta(z_t, t)\right\|_2^2\right] \tag{7}$$

The primary strength of LDMs lies in their exceptional conditioning capabilities. This ability is substantially boosted by integrating cross-attention layers into the model's architecture. Cross-attention layers are inherently flexible, allowing the model to handle various conditioning inputs. These inputs could range from text to bounding boxes or any input described as a sequence of vectors. This flexibility makes LDMs a potent tool for handling different conditioning inputs effectively. We leverage this flexibility by using satellite imagery as conditioning inputs to predict the elevation and path probability maps. Our work showcases how effectively the LDMs can be adapted for a highly specific application such as image-based pathfinding.

Moreover, the inherent flexibility of the cross-attention mechanism also allowed us to introduce an additional level of conditioning through start and goal coordinate information. We concatenate them with the encoded image features by transforming the (x, y) coordinates into a vector representation using a trainable linear projection. This enhances the conditioning process by providing the model with a more nuanced spatial context. We refer to the model that involves this additional conditioning as LDM-cond.

Table 5
Comparative evaluation results. Values before ± indicate the average, while values after ± denote the standard deviation across the dataset.

	NeuralA*	GBFS+PPM
Cost Ratio (%)	121 ± 20	**106 ± 10**
Expansions Ratio (%)	64 ± 43	**44 ± 32**

Table 6
Comparison of different models that infer PPMs from images. Values before ± indicate the average, while values after ± denote the standard deviation across the dataset.

128×128 resolution	LDM	LDM-cond	StyleGAN3	Autoencoder
Cost Ratio (%)	105 ± 8	**103 ± 5**	105 ± 9	106 ± 9
Expansions Ratio $\times 10^2$ (%)	44.9 ± 78.6	**44.2 ± 67**	54.1 ± 99.4	62.9 ± 92.5
MSE $\times 10^{-3}$	1.66 ± 0.14	**1.61 ± 0.13**	1.72 ± 0.14	1.8 ± 0.17

4.4. Empirical evaluation

Empirical evaluation is carried out on the test part of our dataset. The latter is comprised of $18,316$ pathfinding instances (each instance is a map with a unique start-goal pair). We assume that the planner does not have access to the DEM data and must rely only on the input image for pathfinding. The size of the latter is either 64×64 or 128×128.

We conduct three different experiments. First, we compare our primary pipeline, i.e. GBFS+PPM, against Neural A*. PPM is predicted with the basic autoencoder model in this experiment. Next, we examine how the more involved models improve the prediction of the PPM and pathfinding consequently. Finally, we evaluate A*+DEM pipeline, i.e. we use the predicted DEM and run A* on it (this requires an additional cost-tuning as described in Section 4.3).

*GBFS+PPM vs. Neural A** Neural A* [17] is the state-of-the-art ML-based planner that is capable of planning on images. For a fair comparison, for this experiment, we use 64×64 images as the performance of Neural A* significantly degrades when planning on 128×128 images (which is not the problem for our planner as we will see later).

In each run, we track the cost of the constructed path and the number of expansions. Table 5 shows the aggregated results relative to A* on the ground truth DEM. E.g., the average expansions ratio of 44% for our solver means that on average the number of expansions made by it is 66% lower compared to the number of expansions that a regular A* would have made on the ground truth DEM inputs (which are unavailable to our solver and Neural A*). Generally, the results show that the cost of the GBFS-PPM solution does not considerably exceed that of the ground truth solution (which, we emphasize, cannot in principle be constructed on the image-only inputs), while the number of expansions significantly decreases. Moreover, in both metrics, we outperform Neural A* evidently.

Different models for GBFS+PPM In this experiment, we assess whether we can improve the performance of the suggested approach by leveraging more advanced models that are in charge of predicting PPMs: StyleGan3, Latent Diffusion Model (LDM) and LDM additionally conditioned on the start and goal location (LDM-cond). The primary model is denoted as Autoencoder.

For this experiment, we use 128×128 images as input. As before, we track the costs of the obtained paths as well as the number of expansions made by the planner. We normalize these values by dividing the ones achieved by Focal Search on ground-truth PPMs. Additionally, we analyze the mean squared error (MSE) between the predicted PPMs and the ground truth ones. The results are presented in Table 6.

First, note that all models exhibit similar cost ratios which, on average, do not exceed 100% (cost of the optimal solution) much. This shows that predicting PPM and then running GBFS leads to finding near-optimal paths even for a larger input size (128×128 compared to 64×64 in the previous experiment). The best cost ratio is achieved by LDM-cond. This applies to expansions as well – the LDM-cond on average requires the lowest number of expansions compared to the other models and the standard deviation also shrinks. Generally, one may note that the deviation in expansions is noticeably large. We explain this by the presence of a large variety of tasks in the test dataset which differ much in their complexity. We also hypothesize that if the size of the training dataset was larger, all models would exhibit more stable results. Finally, LDM-cond achieves the lowest MSE, followed closely by LDM, while the MSE of StyleGAN3 and Autoencoder is slightly higher. This confirms that LDM-cond generates more accurate PPMs.

Overall, this comparative analysis provides evidence that utilizing more involved models instead of the autoencoder, indeed, positively affects the performance of planning both in terms of cost and expansions (Fig. 16).

A+DEM* In this experiment, we evaluate how the problem at hand can be solved not by leveraging PPMs but rather by predicting DEMs from images and then running A* on these DEMs. This variant requires that the user specifies the proper scaling factor α for the cost function of A* (recall Eq. 3). This is needed as the output of the neural network that predicts DEM from image is within $[0, 1]$, where 0 corresponds to the pixels with the lowest elevation and 1 to the ones of the highest elevation. Meanwhile for the cost

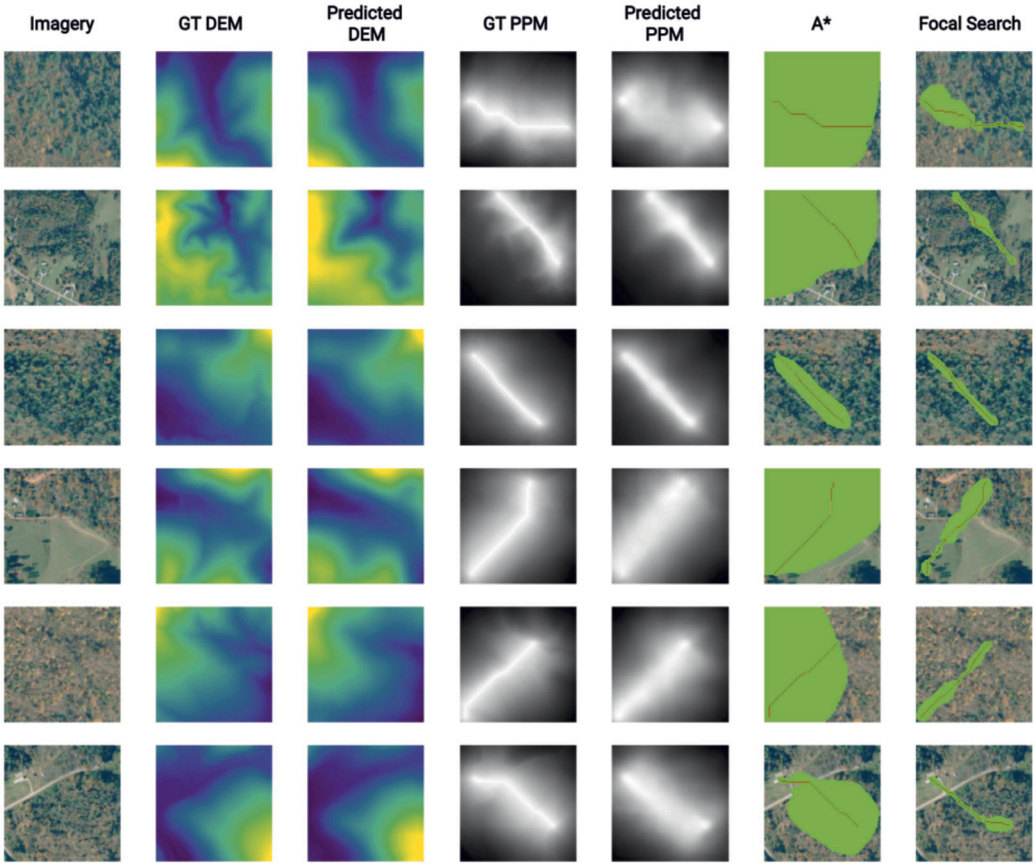
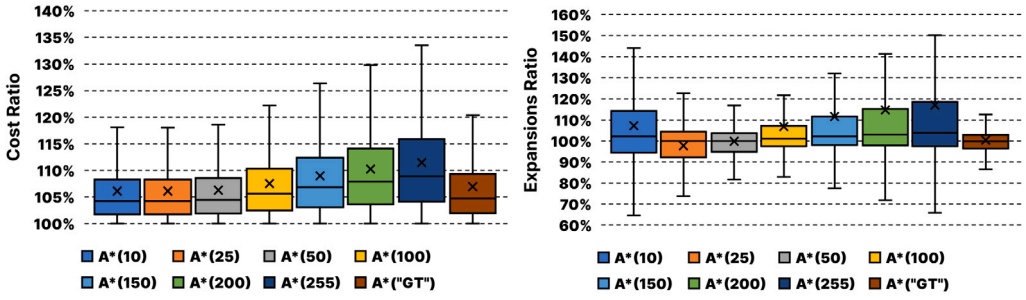


Fig. 16. Two leftmost columns depict the samples (RGB image + DEM) presented in the test part of our dataset. Then the predicted DEM is shown. Grayscale images depict path probability maps (ground-truth ones and the ones inferred by the neural network). Finally, the last two columns depict the search results, i.e. the explored search nodes are shown in green and the resultant path in black. A* stands for running A* on the ground-truth DEM, while the last column depicts the results of the Focal Search on the inferred path probabilty map.



(a) Cost ratio. (b) Expansions ratio.

Fig. 17. Cost and expansions ratios w.r.t. the different scale factor for predicted DEMs.

function, we need this range to be larger in order to be consistent with the first component of the cost function that defines the cost of lateral shift (which is 1 for the cardinal moves and $\sqrt{2}$ for the diagonal moves).

In the experiment we vary the scaling factor manually in a range 10, 25, 50, 100, 150, 200, 255. We also include into the comparison the true scaling factor we have extracted from the ground truth DEM. I.e. for each input image we examine the corresponding DEM (which is not available to the planner) and set the scaling factor to be equal to $max(DEM) - min(DEM)$. We denote this variant as A*(GT).

The results are presented in Fig. 17. Here costs and expansions numbers are normalized by the corresponding values of A* invoked on the ground truth DEMs. Generally, one can note that even with the correct scaling factor A*(GT) is not able to decrease substantially the number of expansions (right plot, right bar). And setting this factor in an unfavorable way, e.g. to 10 or 255, prominently enlarges

the deviation. This is in contrast to planning with GBFS+PPM (recall Table 5). In terms of the path cost, the state of affairs is less dramatic. Generally, A*+DEM is able to find a path of acceptable quality: the maximal cost overhead (w.r.t. the optimal path) is +35% for A*(255). Last but not least, the obtained results provide clear evidence that the performance of A*+DEM is influenced notably by the choice of the scaling factor. This confirms that utilizing PPMs (coupled with GBFS) is more preferable for pathfinding on image inputs as *i*) does not require setting any parameters that influence the performance; and *ii)* it outperforms planning with A* on the reconstructed DEM even when the scaling factor for the latter is known.

5. Discussion and limitations

In this work, we have suggested to utilize supervised machine learning to extract the data that can be successfully used for search-based planning when looking for a path on a binary grid or image. The following three limitations, associated with the suggested approach and techniques, can be distinguished. The first, is the ability to generalize to the problem instances that do not resemble the ones used for training that comes into question. In Section 4.4, we have conducted an empirical evaluation on such scenarios (out-of-the-distribution) for grids-with-costs domain. Indeed, the performance of our method degrades; however, it still *i*) does provide the correct output (due to the utilization of the sound search-based algorithm); and *ii*) outperforms the competitors, including the conventional planning techniques. The second limitation is that our approach requires the labeled ground-truth data in order to conduct (supervised) learning. It is not a problem for the grids-with-costs domain as here one can obtain ground-truth labels directly from the problem instances, as described in the paper. However, for the planning-on-images domain, we have relied on the additional data (i.e. corresponding digital elevation models) to compute the ground-truth outputs. If such additional data is not available (at the training stage) our approach cannot be applied. The third limitation is related to the computing resources needed to accommodate the presented pipeline. On the one hand, the models suggested in the paper are quite lightweight (comparing to modern models used in computer vision and natural language processing), e.g. the autoencoder model contains 1M parameters. As described in Section 4.4, the inference takes approximately 10-40 ms (depending on the hardware used) for a batch size of 64. This meets the real-time requirements of the real-world robotic applications. However, if the robot is equipped with very limited computation resources and is not processing the inputs in batch mode, the inference time is likely to increase and might become a bottleneck. To mitigate this issue neural networks' distillation and compression methods may be needed, However, this has been left out of the scope of this paper.

Notably, all mentioned limitations are generic and typical to any problem-solving method that relies on (supervised) machine learning.

6. 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 considered two setups: the one where the transition costs are known and the one where they are not (planning on images). We have suggested utilizing a deep neural network composed of both convolutional and attention layers to predict a heuristic proxy that we refer to as the path probability map. The latter can be used in combination with Focal Search or Greedy Best First Search resulting in a solver that is capable to generalize and solve challenging pathfinding problems efficiently. Empirically, we have shown that our approach outperforms 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 and planning with kinodynamic constraints (including sample-based planning).

CRediT authorship contribution statement

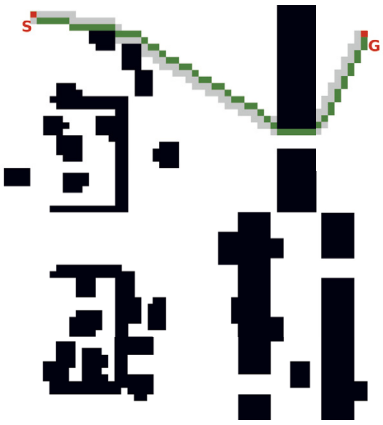
Daniil Kirilenko: Investigation, Software, Validation, Visualization, Writing – original draft. **Anton Andreychuk:** Data curation, Investigation, Methodology, Software, Validation, Visualization, Writing – original draft. **Aleksandr I. Panov:** Investigation, Methodology, Resources, Writing – review & editing. **Konstantin Yakovlev:** Conceptualization, Investigation, Methodology, Project administration, Supervision, Writing – original draft, Writing – review & editing.

Declaration of competing interest

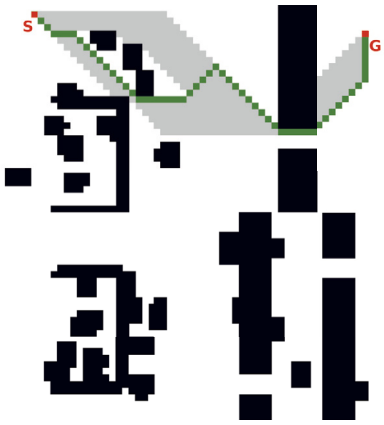
The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgement

This work was partially supported by the Analytical Center for the Government of the Russian Federation in accordance with the subsidy agreement (agreement identifier 000000D730321P5Q0002; grant No. 70-2021-00138).



(a) Path found by FS+PPM on a perfect Theta*-PPM



(b) Path found by FS+PPM on a perfect A*-PPM

Fig. A.18. Difference between the ground-truth PPMs constructed based on A* and Theta* path costs. Only the portion of PPM with *pp*-values of 1 is shown in gray. Green cells denote the path found by FS+PPM.

Table A.7
Empirical results for the experiments involving different types of PPMs. Values before ± indicate the average, while values after ± show the standard deviation.

	Optimal Found Ratio (%) ↑	Cost Ratio (%) ↓	Expansions Ratio (%) ↓
FS+Theta*-PPM	82.97	**100.24 ± 0.74**	26.36 ± 21.08
GBFS+Theta*-PPM	83.02	100.25 ± 0.90	**23.60 ± 18.34**
FS+A*-PPM	61.20	103.6 ± 2.45	53.03 ± 28.73
GBFS+A*-PPM	60.06	103.9 ± 2.32	49.93 ± 27.81

Appendix A. On different ways to create PPMs

A.1. A vs. theta* to create grouth-truth PPMs*

We have considered two approaches to construct ground-truth PPMs for pathfinding on grids (as mentioned in Section 3.2.2 of the main text). The one where we use Theta* to compute the distances and the one where we used A* for that. The principal difference is that in the latter case much more cells are likely to have *pp*-values of 1, while in the former case this number is lower (and these cells naturally form a narrow stripe spanning from start to goal) – see Fig. A.18.

Assume now that PPM is predicted by the neural network absolutely accurately and is fed further to Focal Search. How will the sought path be constructed in case we use A*-PPM? For the sake of simplicity assume that the subotimality factor is large enough for all generated nodes to be part of both OPEN and FOCAL. In this case on any iteration there will be several nodes in FOCAL with *pp*-value equal to 1 to pick from. Thus the node to be picked should be decided by the (FOCAL) tie-breaking rule. The most intuitive tie-breaking rule is to prefer nodes with the lower h-values, i.e. the nodes residing closer to the goal. With this tie-breaker the search will pick one node with *pp*-value of 1 after another and finally reach the goal (without expanding any node with *pp*-value less than 1) in a way depicted on Fig. A.18b. Clearly, for this particular instance an unnecessary zig-zag detour is present and, consecutively, the cost of the path significantly increases. The reason of such zig-zag behavior is that h-based tie-breaker forces the search to transition from a cell belonging to one optimal path to the cell belonging to the other optimal path which is closer to the goal. Meanwhile this exact transition is, in fact, suboptimal (despite both endpoints belong to (different) optimal paths).

To avoid such detours one might consider designing an involved tie-breaking strategy that is based not only on the h- or g-values of the nodes but on which moves have already been applied to reach a specific cell. Instead, as we suggest in this work, one might decrease the number of cells in PPM that have the values of 1, by utilizing Theta* instead of A* while constructing this PPM. When doing so the area occupied by the cells with *pp*-values of 1 shrinks and the negative effect of transitioning between the cells belonging to the different optimal paths diminishes. Thus, invoking Focal Search with the standard tie-breaking technique (of preferring the nodes with the lower h-values) does not lead to a pathological behavior, described above (as depicted on Fig. A.18b).

To assess the described effect at scale we conduct an experiment where we have used two types of the ground-truth PPMs for training the neural network: A*-PPMs and Theta*-PPMs. Then we run our main experiments, i.e. invoke FS+PPM (with $w = 2$) and GBFS+PPM on a large set of unseen pathfinding instances as described in Section 4.4. The results are shown in Table A.7. Clearly, Theta*-PPMs is beneficial as the cost of the resultant paths is better and the number of expansions is notably lower (up to 2x compared to A*-PPMs).

Table A.8
Cost Ratios and Expansions Ratios (in % w.r.t. A*) for Greedy Best First Search (GBFS) that utilizes PPMs predicted by different neural networks, i.e. neural networks that have been trained to predict differently post-processed PPMs. Values before ± indicate the average, while values after ± show the standard deviation.

clipping value	Cost Ratio		Expansion Ratio	
	power 1	power 10	power 1	power 10
0	101 ± 1	100.3 ± 0.4	68 ± 35	28 ± 25
0.3	101 ± 1	100.3 ± 0.4	71 ± 40	30 ± 22
0.6	102 ± 1.2	100.4 ± 0.6	75 ± 37	35 ± 21
0.9	101 ± 0.6	101 ± 1.2	79 ± 41	32 ± 22
0.95	107 ± 5	**100.2 ± 0.9**	57 ± 23	**23 ± 18**

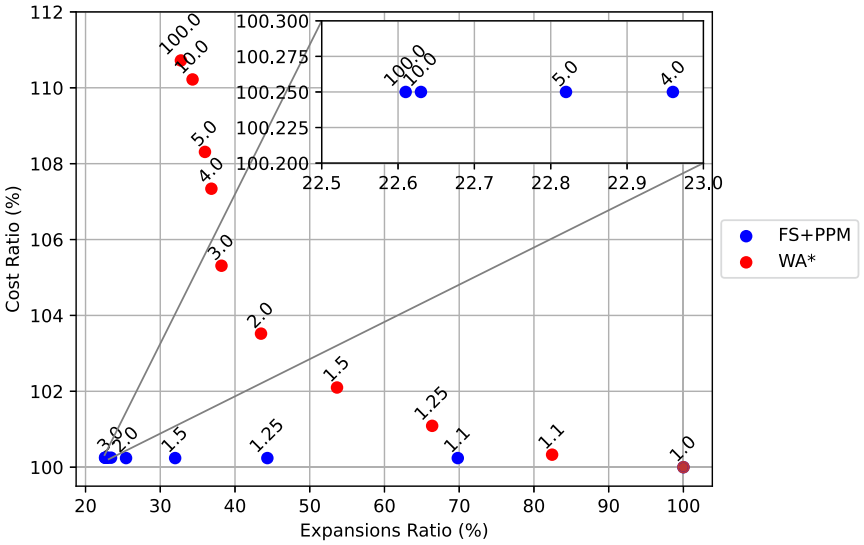


Fig. B.19. Comparison of WA* and FS+PPM with different subotimality factors.

A.2. Additional techniques to further focus theta-PPMs*

As said in the main text, after obtaining ground-truth PPMs based on the Theta* paths we additionally employ the following two techniques to post-process them: powering the *pp*-values and zeroing the ones that do not exceed a specified threshold. Both of these techniques make the resultant PPM more sharp: the *pp*-values of 1 stay unaltered while the others decay to zero much intensively and the ones that are close to zero (measured by comparing to the threshold) are filtered out (zeroed). As said in the main text, we hypothesize that sharpening and filtering the PPMs drives the neural network not to waste its capacity on predicting the lower *pp*-values that are redundant to find a path, but rather forces to predict the *pp*-values only for the regions that are the most needed to find a solution.

To quantitatively measure the effect of applying both of these techniques we have constructed a range of different PPMs for all training instances of our dataset, trained neural network on them, which resulted in having a range of different trained neural networks (which differ in their weights), and then run the experiments on the test part of the dataset utilizing these different neural networks and GBFS+PPM. The results are presented in Table A.8.

Each cell in the table corresponds to either cost ratio or expansions ratio of GBFS+PPM that was obtained utilizing the neural network trained on the PPMs with different clipping thresholds and power values. As one can see, powering the PPM results in a notable performance gain (both in terms of expansions and solution costs). The effect of clipping is less pronounced, however, setting the clipping value to 0.95 seems to be the most beneficial.

Appendix B. On varying the suboptimality factor in grid-based pathfinding

In Section 4.4 we have reported the results of WA* and FS+PPM with the suboptimality factor, w, set to 2. Here, in Fig. B.19 and Table B.9, we report the results for the full spectrum of w, from 1.1 to 10. Additionally, we report the results when $w = 100$ (this corresponds to Greedy Best First Search in our setup) and $w = 1$ (this is A*).

Fig. B.19 shows the scatter plot of the results. Each dot with the color *col* and the coordinates (x, y) should be read as "algorithm *col* uses (on average) x% of expansions (compared to A*) and its resultant cost is (on average) y% of the optimal one". The closer

Table B.9
Cost Ratio and Expansions Ratio for FS+PPM and WA* with different subotimality factors. Values before ± indicate the average, while values after ± show the standard deviation.

Suboptimality factor	Expansions Ratio (%)↓		Cost Ratio(%)↓	
	FS+PPM	WA*	FS+PPM	WA*
1.00	100.00 ± 0.00	100.00 ± 0.00	100.00 ± 0.00	100.00 ± 0.00
1.10	100.24 ± 0.59	100.33 ± 0.76	69.81 ± 29.22	82.43 ± 17.19
1.25	100.24 ± 0.71	101.09 ± 1.80	44.33 ± 30.66	66.38 ± 24.18
1.50	100.24 ± 0.75	102.10 ± 3.03	31.99 ± 25.29	53.65 ± 25.89
2.00	100.24 ± 0.75	103.52 ± 4.85	25.42 ± 19.70	43.48 ± 25.46
3.00	100.25 ± 0.84	105.31 ± 6.85	23.41 ± 17.57	38.19 ± 24.22
4.00	100.25 ± 0.90	107.34 ± 8.55	22.96 ± 16.74	36.84 ± 24.04
5.00	100.25 ± 0.89	108.31 ± 9.36	22.82 ± 16.51	35.98 ± 23.99
10.00	100.25 ± 0.89	110.22 ± 11.37	22.63 ± 16.32	34.32 ± 23.76
100.00	100.25 ± 0.89	110.72 ± 12.13	22.61 ± 16.37	32.76 ± 23.37

Table C.10
Cost Ratios and Expansions Ratios (in % w.r.t. A*) for Greedy Best First Search (GBFS) that utilizes PPMs predicted by different neural networks, i.e. neural networks that have been trained to predict differently post-processed PPMs. Values before ± indicate the average, while values after ± show the standard deviation.

clipping value	Cost Ratio		Expansion Ratio	
	power 1	power 10	power 1	power 10
0	106 ± 10	136 ± 39	44 ± 32	21 ± 24
0.3	109 ± 14	130 ± 44	49 ± 34	16 ± 11
0.6	110 ± 12	135 ± 31	47 ± 31	20 ± 19
0.9	151 ± 49	141 ± 39	18 ± 15	15 ± 14
0.95	146 ± 53	133 ± 42	21 ± 19	25 ± 31

the dot sits to the lower left corner, the better the performance is. Clearly, FS+PPM consistently outperforms WA* across all the suboptimality factors. In the main text we reported the results for $w = 2$ as this value provides the most balanced trade-off between the solution cost and the number of expansions for WA* – observe that the corresponding red dot is the closest to the origin.

Table B.9 presents the same data in tabular form.

Appendix C. On post-processing of the PPMs for image-based pathfinding

As reported earlier, for grid-based pathfinding two techniques to post-process ground-truth PPMs (that are further used to train the neural network) are, evidently, beneficial: powering and clipping. For image-based pathfinding we have also tried to employ these techniques, however the effect is questionable – see Table C.10.

Looking at the expansions ratios from *power* 1 PPMs with different clipping rations one might propose that 0.95 clipping is beneficial. However, observe how the cost increases (both the average and the standard deviation). In fact, for high clipping thresholds the resultant paths degenerate into the straight lines (which is the cause of the cost increase). In other words, the neural network trained on clipped PPMs looses its ability to capture the nuances of the image to construct meaningful detours when needed. The technique of powering the *pp*-values is also not beneficial due to the notable increase in solutions costs. Overall, to obtain meaningful solutions, i.e. paths that actually detour the costly areas (i.e. the ones that are characterized by greater slopes), neither powering nor clipping is valuable. Thus we do not employ these techniques for image-based pathfinding.

Providing a sound explanation, even informal, why the techniques of powering and clipping are not beneficial for the considered image-based pathfinding setup while being effective for grid-based pathfinding is challenging due to the inherent differences in these tasks. Specifically, in grid-based pathfinding transition costs for any pair of orthogonally/diagonally adjacent cells are the same thus the *pp*-values do not need to embed the difference in these costs, while in image-based path-finding transition costs are different due to the difference in elevation. This, possibly, explains why powering the PPM is not beneficial in image-based pathfinding – it indirectly distorts the transition costs between the cells and complicates the problem of implicit reconstruction of these costs in the form of PPM. Next, in grid-based pathfinding, if force the neural network to learn *pp*-values for all the cells, the former would have to predict the degree of proximity to the optimal path for every cell, which could distract it from the more important task of establishing the shaping the (optimal) path itself. Thus, clipping low *pp*-values enhances the quality of the predictions by ensuring the network prioritizes the most important cells. In image-based pathfinding, all cells are potentially traversable and low *pp*-values represent areas to be avoided due to higher traversal difficulty. Clipping these values would remove valuable information, making it harder for the neural network to learn to predict the degree of traversability of a cell, thus affecting the quality of a path. In other words it may be the case that in grid-based pathfinding the most important task is to understand where the path is located while in image-based pathfinding one needs to make a judjement regarding each individual cell – high likely it should be avoided. Despite these arguments being made, we emphasize that they are hypothetical and not rigorous. Meanwhile, the results of the empirical evaluation provide

clear evidence that both powering *pp*-values and clipping them negatively impact the performance of the suggested planners in the considered image-based pathfinding setup.

Data availability

We included the link to the repository that contains our code and data into the paper.

References

[1] N. Rivera, C. Hernández, N. Hormazábal, J.A. Baier, The 2ˆ k neighborhoods for grid path planning, J. Artif. Intell. Res. 67 (2020) 81–113.
[2] J.P. Bailey, A. Nash, C.A. Tovey, S. Koenig, Path-length analysis for grid-based path planning, Artif. Intell. 301 (2021) 103560.
[3] W. Lee, R. Lawrence, Fast grid-based path finding for video games, in: Proceedings of the 26th Canadian Conference on Artificial Intelligence (Canadian AI 2013), 2013, pp. 100–111.
[4] R. Lawrence, V. Bulitko, Database-driven real-time heuristic search in video-game pathfinding, IEEE Trans. Comput. Intell. AI Games 5 (2012) 227–241.
[5] A. Elfes, Using occupancy grids for mobile robot perception and navigation, Computer 22 (1989) 46–57.
[6] E.G. Tsardoulias, A. Iliakopoulou, A. Kargakos, L. Petrou, A review of global path planning methods for occupancy grid maps regardless of obstacle density, J. Intell. Robot. Syst. 84 (2016) 829–858.
[7] P. Sodhi, B.-J. Ho, M. Kaess, Online and consistent occupancy grid mapping for planning in unknown environments, in: Proceedings of the 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2019), IEEE, 2019, pp. 7879–7886.
[8] G. Hedrick, N. Ohi, Y. Gu, Terrain-aware path planning and map update for Mars sample return mission, IEEE Robot. Autom. Lett. 5 (2020) 5181–5188.
[9] T. Guan, Z. He, D. Manocha, L. Zhang, Ttm: terrain traversability mapping for autonomous excavator navigation in unstructured environments, arXiv preprint, arXiv:2109.06250, 2021.
[10] J. Li, G. Deng, C. Luo, Q. Lin, Q. Yan, Z. Ming, A hybrid path planning method in unmanned air/ground vehicle (UAV/UGV) cooperative systems, IEEE Trans. Veh. Technol. 65 (2016) 9585–9596.
[11] P.E. Hart, N.J. Nilsson, B. Raphael, A formal basis for the heuristic determination of minimum cost paths, IEEE Trans. Syst. Sci. Cybern. 4 (1968) 100–107.
[12] D. Speck, A. Biedenkapp, F. Hutter, R. Mattmüller, M. Lindauer, Learning heuristic selection with dynamic algorithm configuration, in: Proceedings of the 31st International Conference on Automated Planning and Scheduling (ICAPS 2021), 2021, pp. 597–605.
[13] M. Janner, Y. Du, J. Tenenbaum, S. Levine, Planning with diffusion for flexible behavior synthesis, in: Proceedings of the 39th International Conference on Machine Learning (ICML 2022), 2022, pp. 9902–9915.
[14] P. Ramachandran, B. Zoph, Q.V. Le, Searching for activation functions, arXiv preprint, arXiv:1710.05941, 2017.
[15] M. Tan, Q. Le, Efficientnet: rethinking model scaling for convolutional neural networks, in: Proceedings of the 36th International Conference on Machine Learning (ICML 2019), 2019, pp. 6105–6114.
[16] T. Takahashi, H. Sun, D. Tian, Y. Wang, 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), 2019, pp. 764–772.
[17] R. Yonetani, T. Taniai, M. Barekatain, M. Nishimura, A. Kanezaki, Path planning using neural A* search, in: Proceedings of the 38th International Conference on Machine Learning (ICML 2021), 2021, pp. 12029–12039.
[18] J. Pearl, J.H. Kim, Studies in semi-admissible heuristics, IEEE Trans. Pattern Anal. Mach. Intell. (1982) 392–399.
[19] I. Pohl, Heuristic search viewed as path finding in a graph, Artif. Intell. 1 (1970) 193–204.
[20] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N. Gomez, Ł. Kaiser, I. Polosukhin, Attention is all you need, in: Proceedings of the 31st Conference on Neural Information Processing Systems (NeurIPS 2017), 2017.
[21] D. Kirilenko, A. Andreychuk, A. Panov, K. Yakovlev, Transpath: learning heuristics for grid-based pathfinding via transformers, in: Proceedings of the 37th AAAI Conference on Artificial Intelligence (AAAI 2023), 2023, pp. 12436–12443.
[22] T. Karras, M. Aittala, S. Laine, E. Härkönen, J. Hellsten, J. Lehtinen, T. Aila, Alias-free generative adversarial networks, Adv. Neural Inf. Process. Syst. 34 (2021) 852–863.
[23] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, B. Ommer, High-resolution image synthesis with latent diffusion models, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2022, pp. 10684–10695.
[24] M. Likhachev, G.J. Gordon, S. Thrun, ARA*: Anytime A* with provable bounds on sub-optimality, in: S. Thrun, L.K. Saul, B. Schölkopf (Eds.), Advances in Neural Information Processing Systems, vol. 16, NIPS 2003, MIT Press, 2003, pp. 767–774, http://papers.nips.cc/paper/2382-ara-anytime-a-with-provable-bounds-on-sub-optimality.pdf, 2003.
[25] E.A. Hansen, R. Zhou, Anytime heuristic search, J. Artif. Intell. Res. 28 (2007) 267–297.
[26] L. Cohen, M. Greco, H. Ma, C. Hernández, A. Felner, T.S. Kumar, S. Koenig, Anytime focal search with applications, in: Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI 2018), 2018, pp. 1434–1441.
[27] J.T. Thayer, W. Ruml, Bounded suboptimal search: a direct approach using inadmissible estimates, in: Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI) 2011, 2011, pp. 674–679.
[28] D. Gilon, A. Felner, R. Stern, Dynamic potential search – a new bounded suboptimal search, in: Proceedings of the 9th Annual Symposium on Combinatorial Search (SOCS 2016), 2016, pp. 36–44.
[29] M. Fickert, T. Gu, W. Ruml, New results in bounded-suboptimal search, in: Proceedings of the 36th AAAI Conference on Artificial Intelligence (AAAI 2022), 2022, pp. 10166–10173.
[30] S. Aine, S. Swaminathan, V. Narayanan, V. Hwang, M. Likhachev, Multi-heuristic a, Int. J. Robot. Res. 35 (2016) 224–243.
[31] M. Likhachev, A. Stentz, R* search, in: Proceedings of the 23rd AAAI Conference on Artificial Intelligence (AAAI 2008), AAAI Press, 2008, pp. 344–350, http://www.aaai.org/Library/AAAI/2008/aaai08-054.php.
[32] M. Bagatella, M. Olšák, M. Rolínek, G. Martius, Planning from pixels in environments with combinatorially hard search spaces, Adv. Neural Inf. Process. Syst. 34, NeurIPS 2021 (2021) 24707–24718.
[33] M.V. Pogančić, A. Paulus, V. Musil, G. Martius, M. Rolinek, Differentiation of blackbox combinatorial solvers, in: Proceedings of the 8th International Conference on Learning Representations (ICLR 2020), 2020.
[34] Z. Li, Q. Chen, V. Koltun, Combinatorial optimization with graph convolutional networks and guided tree search, in: Proceedings of the 32nd Conference on Neural Information Processing System, NeurIPS 2018, 2018.
[35] M. Pándy, W. Qiu, G. Corso, P. Veličković, Z. Ying, J. Leskovec, P. Lio, Learning graph search heuristics, in: Proceedings of the 1st Learning on Graphs Conference (LoG 2022), 2022, pp. 10:1–10:13.
[36] A. Tamar, Y. Wu, G. Thomas, S. Levine, P. Abbeel, Value iteration networks, in: Proceedings of the 30th International Conference on Neural Information Processing Systems (NeurIPS 2016), 2016, pp. 2154–2162.
[37] M. Bhardwaj, S. Choudhury, S. Scherer, Learning heuristic search via imitation, in: Proceedings of the 1st Conference on Robot Learning (CoRL 2017), 2017, pp. 271–280.

[38] A.I. Panov, K.S. Yakovlev, R. Suvorov, Grid path planning with deep reinforcement learning: preliminary results, Proc. Comput. Sci. 123 (2018) 347–353.
[39] T. Li, R. Chen, B. Mavrin, N.R. Sturtevant, D. Nadav, A. Felner, Optimal search with neural networks: challenges and approaches, in: Proceedings of the 15th International Symposium on Combinatorial Search (SoCS 2015), 2022, pp. 109–117.
[40] M. Greco, J. Toro, C. Hernández-Ulloa, J.A. Baier, K-focal search for slow learned heuristics, in: Proceedings of the 15th International Symposium on Combinatorial Search (SoCS 2015), 2022, pp. 279–281.
[41] N. Soboleva, K. Yakovlev, Gan path finder: preliminary results, in: Proceedings of the 42nd German Conference on AI (KI 2019), 2019, pp. 316–324.
[42] C. Xia, A. El Kamel, Neural inverse reinforcement learning in autonomous navigation, Robot. Auton. Syst. 84 (2016) 1–14, https://doi.org/10.1016/j.robot.2016.06.003, publisher: Elsevier B.V.
[43] D.S. Chaplot, E. Parisotto, R. Salakhutdinov, Active neural localization, in: International Conference on Learning Representations, 2018, pp. 1–15, http://arxiv.org/abs/1801.08214, arXiv:1801.08214.
[44] H.T.L. Chiang, A. Faust, M. Fiser, A. Francis, Learning navigation behaviors end-to-end with AutoRL, IEEE Robot. Autom. Lett. 4 (2019) 2007–2014, https://doi.org/10.1109/LRA.2019.2899918, arXiv:1809.10124.
[45] A. Francis, A. Faust, H.T.L. Chiang, J. Hsu, J.C. Kew, M. Fiser, T.W.E. Lee, Long-range indoor navigation with PRM-RL, IEEE Trans. Robot. 36 (2020) 1115–1134, https://doi.org/10.1109/TRO.2020.2975428, arXiv:1902.09458.
[46] B. Liu, X. Xiao, P. Stone, A lifelong learning approach to mobile robot navigation, IEEE Robot. Autom. Lett. 6 (2021) 1090–1096, https://doi.org/10.1109/LRA.2021.3056373, arXiv:2007.14486.
[47] D. Shah, A. Bhorkar, H. Leen, I. Kostrikov, N. Rhinehart, S. Levine, Offline reinforcement learning for visual navigation, in: 6th Conference on Robot Learning, 2022, http://arxiv.org/abs/2212.08244, arXiv:2212.08244.
[48] D. Shah, B. Eysenbach, N. Rhinehart, S. Levine, Rapid exploration for open-world navigation with latent goal models, in: Proceedings of the 5th Conference on Robot Learning, vol. 164, 2021, pp. 674–684, http://arxiv.org/abs/2104.05859, arXiv:2104.05859.
[49] A. Staroverov, D.A. Yudin, I. Belkin, V. Adeshkin, Y.K. Solomentsev, A.I. Panov, Real-time object navigation with deep neural networks and hierarchical reinforcement learning, IEEE Access 8 (2020) 195608–195621, https://doi.org/10.1109/ACCESS.2020.3034524, https://ieeexplore.ieee.org/document/9241850/.
[50] D.S. Chaplot, D. Gandhi, A. Gupta, R. Salakhutdinov, Object goal navigation using goal-oriented semantic exploration, in: Advances in Neural Information Processing Systems, vol. 33, 2020, pp. 1–11, http://arxiv.org/abs/2007.00643, arXiv:2007.00643.
[51] A. Staroverov, A. Panov, Hierarchical landmark policy optimization for visual indoor navigation, IEEE Access 10 (2022) 70447–70455, https://doi.org/10.1109/ACCESS.2022.3182803, https://ieeexplore.ieee.org/document/9795006/.
[52] M. Deitke, E. VanderBilt, A. Herrasti, L. Weihs, J. Salvador, K. Ehsani, W. Han, E. Kolve, A. Farhadi, A. Kembhavi, R. Mottaghi, ProcTHOR: Large-Scale Embodied AI Using Procedural Generation, Advances in Neural Information Processing Systems, vol. 35, 2022, http://arxiv.org/abs/2206.06994, arXiv:2206.06994.
[53] L. Mezghani, S. Sukhbaatar, T. Lavril, O. Maksymets, D. Batra, P. Bojanowski, K. Alahari, Memory-augmented reinforcement learning for image-goal navigation, in: IEEE/RSJ International Conference on Intelligent Robots and Systems, 2022, http://arxiv.org/abs/2101.05181, arXiv:2101.05181.
[54] M. Shridhar, L. Manuelli, D. Fox, Perceiver-actor: a multi-task transformer for robotic manipulation, in: Proceedings of the 6th Conference on Robot Learning, vol. 205, 2023, pp. 785–799, http://arxiv.org/abs/2209.05451, arXiv:2209.05451.
[55] B.Y. Lin, C. Huang, Q. Liu, W. Gu, S. Sommerer, X. Ren, On grounded planning for embodied tasks with language models, in: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 37, 2023, pp. 13192–13200, https://arxiv.org/abs/2209.00465?utm_source=researcher_app&utm_medium=referral&utm_campaign=RESR_MRKT_Researcher_inbound, arXiv:2209.00465.
[56] D. Shah, B. Osinski, B. Ichter, S. Levine, LM-nav: robotic navigation with large pre-trained models of language, vision, and action, in: Proceedings of the 6th Conference on Robot Learning, vol. 205, 2023, pp. 492–504, http://arxiv.org/abs/2207.04429, arXiv:2207.04429.
[57] A. Nash, K. Daniel, S. Koenig, A. Felner, Theta*: any-angle path planning on grids, in: Proceedings of the 22nd AAAI Conference on Artificial Intelligence (AAAI 2007), 2007, pp. 1177–1183.
[58] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: Proceedings of the 29th IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2016), 2016, pp. 770–778.
[59] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, N. Houlsby, An image is worth 16x16 words: transformers for image recognition at scale, in: Proceedings of the 9th International Conference on Learning Representations (ICLR 2021), 2021, https://openreview.net/forum?id=YicbFdNTTy.
[60] K. Han, Y. Wang, H. Chen, X. Chen, J. Guo, Z. Liu, Y. Tang, A. Xiao, C. Xu, Y. Xu, Z. Yang, Y. Zhang, D. Tao, A survey on vision transformer, IEEE Trans. Pattern Anal. Mach. Intell. 45 (1) (2023) 87–110, https://doi.org/10.1109/TPAMI.2022.3152247.
[61] A. Panov, K. Yakovlev, Transpath dataset: generative models for grid-based and image-based pathfinding, Mendeley data, https://doi.org/10.17632/bc3zmk2thy.1, 2023.
[62] D.P. Kingma, J. Ba, Adam: a method for stochastic optimization, arXiv preprint, arXiv:1412.6980, 2014.
[63] L.N. Smith, N. Topin, Super-convergence: very fast training of residual networks using large learning rates, arXiv preprint, arXiv:1708.07120, 2018, http://arxiv.org/abs/1708.07120.
[64] N.R. Sturtevant, Benchmarks for grid-based pathfinding, IEEE Trans. Comput. Intell. AI Games 4 (2012) 144–148.
[65] A. Candela, D. Wettergreen, An approach to science and risk-aware planetary rover exploration, IEEE Robot. Autom. Lett. 7 (2022) 9691–9698.
[66] V.-D. Hoang, D.C. Hernández, J. Hariyono, K.-H. Jo, Global path planning for unmanned ground vehicle based on road map images, in: Proceedings of the 7th International Conference on Human System Interactions (HSI 2014), IEEE, 2014, pp. 82–87.
[67] S. Vélez, R. Vacas, H. Martín, D. Ruano-Rosa, S. Álvarez, High-resolution UAV RGB imagery dataset for precision agriculture and 3d photogrammetric reconstruction captured over a pistachio orchard (pistacia vera l.) in Spain, Data 7 (2022) 157.
[68] S.S. Sutton, M. Chojnacki, A.S. McEwen, R.L. Kirk, C.M. Dundas, E.I. Schaefer, S.J. Conway, S. Diniega, G. Portyankina, M.E. Landis, et al., Revealing active Mars with hirise digital terrain models, Remote Sens. 14 (2022) 2403.
[69] Y. Pang, J. Lin, T. Qin, Z. Chen, Image-to-image translation: methods and applications, IEEE Trans. Multimed. 24 (2022) 3859–3881, https://doi.org/10.1109/TMM.2021.3109419.