POLITECNICO DI TORINO
Repository ISTITUZIONALE

Neural Weighted A*: Learning Graph Costs and Heuristics with Differentiable Anytime A*

Original
Neural Weighted A*: Learning Graph Costs and Heuristics with Differentiable Anytime A* / Archetti, A., Cannici, M., Matteucci, M.. - ELETTRONICO. - 13163:(2022), pp. 596-610. (LOD 2021, 7th International Online & Onsite Conference on Machine Learning, Optimization, and Data Science Grasmere, Lake District, England, UK October 4-8, 2021) [10.1007/978-3-030-95467-3_43].

Availability:
This version is available at: 11583/2971580 since: 2022-09-21T14:31:00Z

Publisher:
Springer

Published
DOI:10.1007/978-3-030-95467-3_43

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright
Springer postprint/Author's Accepted Manuscript

This version of the article has been accepted for publication, after peer review (when applicable) and is subject to Springer Nature's AM terms of use, but is not the Version of Record and does not reflect post-acceptance improvements, or any corrections. The Version of Record is available online at: http://dx.doi.org/10.1007/978-3-030-95467-3_43

(Article begins on next page)

27 November 2026

arXiv:2105.01480v2 [cs.LG] 2 Aug 2021

Neural Weighted A*: Learning Graph Costs and Heuristics with Differentiable Anytime A*

Alberto Archetti[0000−0003−3826−4645], Marco Cannici[0000−0002−9217−3552], and Matteo Matteucci[0000−0002−8306−6739]

Politecnico di Milano, Milano 20133, Italy
alberto1.archetti@mail.polimi.it
{marco.cannici,matteo.matteucci}@polimi.it

Abstract. Recently, the trend of incorporating differentiable algorithms into deep learning architectures arose in machine learning research, as the fusion of neural layers and algorithmic layers has been beneficial for handling combinatorial data, such as shortest paths on graphs. Recent works related to data-driven planning aim at learning either cost functions or heuristic functions, but not both. We propose Neural Weighted A*, a differentiable anytime planner able to produce improved representations of planar maps as graph costs and heuristics. Training occurs end-to-end on raw images with direct supervision on planning examples, thanks to a differentiable A* solver integrated into the architecture. More importantly, the user can trade off planning accuracy for efficiency at run-time, using a single, real-valued parameter. The solution suboptimality is constrained within a linear bound equal to the optimal path cost multiplied by the tradeoff parameter. We experimentally show the validity of our claims by testing Neural Weighted A* against several baselines, introducing a novel, tile-based navigation dataset. We outperform similar architectures in planning accuracy and efficiency.

Keywords: Weighted A*· Differentiable algorithms · Data-based planning.

1 Introduction

A* [13] is the most famous heuristic-based planning algorithm, and it constitutes one of the essentials for the computer scientist's toolbox. It is widely used in robotic motion [27] and navigation systems [19], but its range extends to all the fields that benefit from shortest path search on graphs [24]. Differently from other shortest path algorithms, such as Dijkstra [10] or Greedy Best First [25], A* is known to be optimally efficient [25]. This means that, besides returning the optimal solution, there is no other algorithm that can be more efficient, in general, provided the same admissible heuristic. Even though optimality seems a desirable property for A*, it is often more of a burden than a virtue in practical applications. This is because, in the worst case, A* takes exponential time to converge to the optimal solution, and this is not affordable in large search spaces.

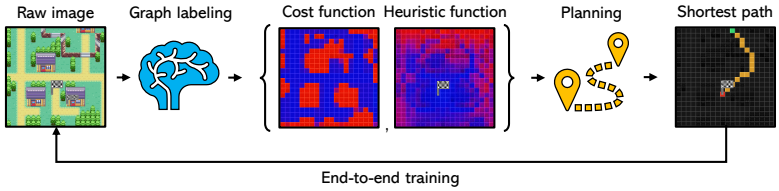

Fig. 1. Can we learn to navigate a terrain effectively by just looking at its map? Neural Weighted A* accurately predicts from the raw image of the navigation area the costs of traversing local regions and a global heuristic for reaching the destination.

Another compelling issue of A* is that hand-crafting non-trivial heuristics is costly and reliant on domain knowledge. Despite the prolific research in deep-learning-based graph labeling [33], neural networks often struggle with data exhibiting combinatorial complexity, such as shortest paths [23]. For this reason, many researchers started including differentiable algorithmic layers directly into deep learning pipelines. These layers implement algorithms with combinatorial operations in the forward pass, while providing a smooth, approximated derivative in the backward pass. This approach helps the neural components to converge faster with fewer data samples, promoting the birth of hybrid architectures, trainable end-to-end, that extend the reach of deep learning to complex combinatorial problems. Many backpropagation-ready algorithmic layers have been developed, such as [2,1,6,23,30,32]. Among these, some [6,23,32] propose differentiable shortest path solvers able to learn graph costs from planning examples on raw image inputs. However, none of the previous works tackles heuristic design, which is the essential aspect that makes A* scale to complex scenarios.

With Neural Weighted A* (NWA*), we propose the first deep-learning-based differentiable planner able to predict graph costs and heuristic functions from unlabeled images of navigation areas (Fig. 1). Training occurs end-to-end on shortest-path examples, exploiting a fusion of differentiable planners from [23,32]. Also, NWA* is the first architecture that enables the user to trade off planning accuracy for convergence speed with a single, real-valued parameter, even at runtime. Balancing search accuracy and efficiency from unlabeled images is crucial in many navigation problems. Among the most notable examples, we find hierarchical planning for robotic navigation, where accuracy and efficiency assume a different priority depending on the spacial granularity at which planning is executed, and real-world pedestrian modeling, where graph labeling from raw images is not feasible by hand [32]. As a final remark, since our method arises from the Weighted A* algorithm [12], the solution cost never exceeds the optimal one by a factor proportional to the tradeoff parameter.

We extensively test Neural Weighted A* against the baselines from [23,32], and conduct experiments on two tile-based datasets. The first is adapted from [23], while the second dataset is novel, and its goal is to provide a scenario more complex than the first. Both datasets are publicly available (Sec. 5). In summary, with Neural Weighted A*, we make the following contributions:

- We develop the first deep-learning system able to generate both cost functions and heuristic functions in a principled way from raw map images.
- We propose the first method to smoothly trade off planning accuracy and efficiency at runtime, compliant with the Weighted A* bound on solution suboptimality.
- We augment an existing dataset and propose a new one for planning benchmarks on planar navigation problems.

2 Related Work

Connections between deep learning and differentiable algorithms arise from different domains. The first examples lie within the 3D rendering literature [16], composing neural-network-based encoders with differentiable renderer-like decoders to learn the constructing parameters of the input scene. Differentiable decoders spread to physics simulations [4,26], logical reasoning [30,31], and control [1,11,15]. Combinatorial optimization is also a topic of interest, from differentiable problem-specific solvers [5,18,30] to general-purpose ones [2,6,23]. Indeed, many combinatorial algorithms and their differentiable implementation have already been studied, such as Traveling Salesman [5,9], (Conditional) Markov Random Fields [7,17,20,34], and Shortest Path [32]. Each of these works shows how structured differentiable components enable deep learning architectures to learn combinatorial patterns easily from data.

A handful of works started experimenting with convexity, one of the most important properties of combinatorial optimization. The first is the neural layer by Amos et al. [3], which is constrained to learn convex functions only. Following this work, Pitis et al. [22] exploit the convex neural layer to design a trainable graph-embedding metric that respects triangle inequality. This is beneficial for learning graph costs that encode mathematically sound distances, even though the method is limited to train on a single graph.

Among search-based planning research, some studies focus on a data-driven approach where planning cues are inferred from raw image inputs [6,23,32]. In [23], Vlastelica et al. develop a technique to differentiate solvers for integer linear optimization problems, treating them as black-boxes. As the shortest path belongs to this set of problems [28], the authors are able to map images of navigation areas to extremely accurate graph costs, such that the paths evaluated on the cost predictions closely resembles the ground-truth ones. At its core, the technique from [23] consists of a smooth interpolation of the piecewise constant function defined by the black-box solver. This technique is well suited for learning accurate costs, but, due to its black-box nature, cannot address heuristic design, the aspect that makes the search efficient, and which we study in this work.

On the other side of the spectrum, Yonetani et al. [32], reformulate the canonical A* algorithm as a set of differentiable tensor operations. Their goal is to develop a deep-learning-based architecture able to learn improved cost functions such that the planning search avoids non-convenient regions to traverse. In order to train the architecture, the authors define a loss function that minimizes

the difference between the nodes expanded by A* and the ground truth paths. Therefore, the neural network is forced to learn shortcuts and bypasses that severely accelerate the search, but may result in inaccurate path predictions.

Lastly, Berthet et al. [6] propose a general-purpose differentiable combinatorial optimizer based on stochastic perturbations with a strong theoretical insight. Despite this work being more recent and general than [23], it was outperformed by [23] in our experimental settings. Therefore, we choose to focus on [23] for the rest of the paper. In fact, our work builds on [23,32] to develop the first learning architecture that is not forced to choose to plan either accurately or efficiently, but is able to smoothly tradeoff between these opposing aspects of planning.

3 Preliminaries

Let $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ be a graph where $\mathcal{N}$ is a finite set of nodes and $\mathcal{E}$ is a finite set of edges connecting the nodes. Let s and t be two distinct nodes from $\mathcal{N}$, called source and target. We define a path y on $\mathcal{G}$ connecting s to t as a sequence of adjacent nodes $(n_0, n_1, \ldots, n_k)$ such that $n_0 = s$, $n_k = t$, and each node is traversed at most once.

In this work, we always refer to the 8-GridWorld setting [6,8,23,32] but the techniques we describe can be easily applied to general graph settings, nevertheless. In 8-GridWorld, nodes are disposed in a grid-like pattern, and edges connect only the nodes belonging to neighboring cells, including the diagonal ones. Each node is paired with a non-negative, real-valued cost belonging to a cost function $\bar{W} \in \mathbb{R}_+^{\mathcal{N}}$. Paths are represented in binary form as $Y \in \{0,1\}^{\mathcal{N}}$ with ones corresponding to the traversed nodes. The total cost of a path Y, denoted as $\langle \bar{W}, Y \rangle$, is the sum of its nodes' costs. Given a graph $\mathcal{G}$ with costs $\bar{W}$, a source node s, and a target node t, the shortest path problem consists of finding the path $\bar{Y}$ having the minimal total cost among all the paths connecting s and t.

3.1 A* and Weighted A*

We focus on A* [13], a heuristic-based shortest path algorithm for graphs. A* searches for a minimum-cost path from s to t iteratively expanding nodes according to the priority measure

$$F(n) = G(n) + H(n). \tag{1}$$

$G(n)$ is the exact cumulated cost from s to n, and $H(n)$ is a heuristic function estimating the cost between n and t. A* is known to be optimally efficient when $H(n)$ is admissible [25], i.e., it never overestimates the optimal cost between n and t. An example of admissible heuristic on 8-GridWorld is

$$H_C(n) = w_{\min} \cdot D_C(n, t) \tag{2}$$

where $w_{\min} = \min_{n \in \mathcal{N}} \bar{W}(n)$ and $D_C(n,t)$ is the Chebyshev distance between n and t in the grid, i.e., $D_C(n,t) = \max\{|n_x - t_x|, |n_y - t_y|\}$.

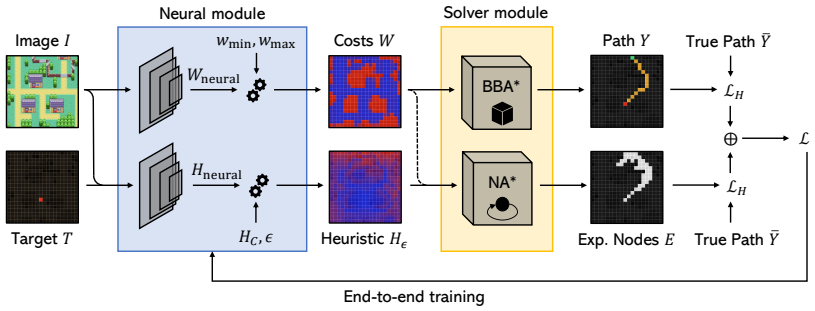

Fig. 2. Schematics of Neural Weighted A*. The neural module (blue) predicts the costs W and the heuristic function H_ϵ. The solver module (yellow) runs two A* solvers, differentiable according to the techniques described in [23,32]. The first solver computes the shortest path Y, while the second computes the nodes expanded by A*, E.

For large graphs, A* may take exponential time to find the optimal solution [12]. Hence, in practical applications, it is preferable to find an approximate solution quickly, sacrificing the optimality constraint. This idea is explored by one of A*'s extensions, called Weighted A* (WA*) [21]. This algorithm is equivalent to a standard A* search, but the heuristic $H(n)$ in Eq. 1 is scaled up by a factor of $1 + \epsilon$, where $\epsilon \geq 0$. Assuming $H(n)$ to be admissible, WA* returns the optimal path for $\epsilon = 0$. Conversely, for $\epsilon > 0$, the heuristic function drives the search, leading to fewer node expansions, but influencing the path trajectory. The cost difference between the WA* solution Y and the optimal path $\bar{Y}$ is linearly bounded [12]:

$$\langle \bar{W}, Y \rangle \leq (1 + \epsilon) \cdot \langle \bar{W}, \bar{Y} \rangle. \tag{3}$$

4 Neural Weighted A*

In the A* algorithm, the nodes are expanded according to the balance between the cumulated costs $G \in \mathbb{R}_+^{\mathcal{N}}$ and the heuristic function H (Eq. 1). If $G \gg H$, A* expands nodes mostly according to G, behaving similarly to Dijkstra's algorithm. If, on the other hand, $H \gg G$, then the A* behavior is closer to a Greedy Best First search. Therefore, tuning the scale of the final cost and heuristic functions is the key to control the tradeoff between planning accuracy and efficiency. We propose to accomplish this goal with a novel, deep-learning-based architecture for graph labeling from planning examples, called Neural Weighted A* (Fig. 2). It is composed of two modules: the *neural module* (Sec. 4.1) and the *solver module* (Sec. 4.2). The neural module generates planning-ready graph costs and a heuristic function from the top view of the navigation area. The solver module executes the planning procedure in the forward pass, while providing a smooth derivative in the backward pass to enable end-to-end training.

In the following, we indicate with the "neural" subscript values directly coming out of neural networks, such as W_{neural} and H_{neural}, while we use the bar superscript, as in $\bar{W}$ and $\bar{Y}$, to indicate ground-truth values.

4.1 The neural module

The neural module is composed of two fully-convolutional neural networks. The first one (upper network in Fig. 2) processes a color image $I \in [0,1]^{\Gamma \times 3}$ of resolution Γ returning a cost prediction $W_{\text{neural}} \in [0,1]^{\mathcal{N}}$. The second neural network (lower network in Fig. 2) takes as input the concatenation of I and the target T, i.e., a matrix with a one corresponding to the target position scaled up to the image resolution Γ, and returns a heuristic prediction $H_{\text{neural}} \in [0,1]^{\mathcal{N}}$. This separation enforces the system to learn costs that are target-agnostic, since T is not included in the input of the first neural network.

In order to control the relative magnitude between the costs and the heuristic function, we uniformly scale the values of W_{neural} in the interval $[w_{\min}, w_{\max}]$ such that $w_{\min} > 0$. We call W the new and final cost function. Then, we compute the final heuristic function as

$$H_\epsilon = (1 + \epsilon \cdot H_{\text{neural}}) \cdot H_C \tag{4}$$

where H_C is the Chebyshev heuristic (Eq. 2), and $\epsilon \geq 0$ is the accuracy-efficiency tradeoff parameter.

For any node $n \in \mathcal{N}$, the purpose of ϵ and $H_{\text{neural}}(n)$ is to modulate the intensity of the final heuristic $H_\epsilon(n)$ between two values, $H_C(n)$ and $(1+\epsilon) \cdot H_C(n)$. When $\epsilon = 0$, $H_\epsilon(n)$ is equal to the admissible Chebyshev heuristic $H_C(n)$. Therefore, the solution optimality is guaranteed. Conversely, when $\epsilon > 0$, $H_\epsilon(n)$ is not admissible, in general, anymore. However, if n is a node likely to be convenient to traverse, it is mapped to a value close to $H_C(n)$, as the neural network learns to predict a value $H_{\text{neural}}(n) \approx 0$. If, on the other hand, n seems very unlikely to be traversed, its heuristic value is scaled up by a factor of $1+\epsilon$, as $H_{\text{neural}} \approx 1$. In this way, by increasing ϵ, we increase the difference in heuristic values between nodes convenient and non-convenient to expand according to the neural prediction, forcing A* to prefer the nodes where $H_{\text{neural}} \approx 0$. Fig. 3 visually illustrates the relationship between H_C, H_{neural}, and H_ϵ.

Lastly, we observe that $H_\epsilon(n) \leq (1+\epsilon) \cdot H_C(n)$. Since H_C is an admissible heuristic function for 8-GridWorld, we are guaranteed, by the Weighted A* bounding result (Sec. 3.1, Eq. 3), to never return a path whose cost exceeds the optimal one by a factor of $1+\epsilon$.

4.2 The solver module

The solver module is composed of two differentiable A* solvers. The first, called Black-Box A*, implements the A* algorithm with black-box differentiation as in [23]. It computes the shortest path Y given the costs W, the admissible Chebyshev heuristic H_C (Eq. 2) and the source-target nodes. The second solver

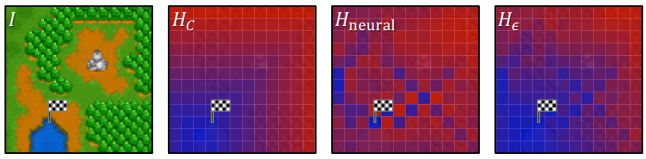

Fig. 3. From left to right, first: image sample from the Warcraft dataset (Sec. 5). The target node is in the bottom left region of the map. Second: Chebyshev heuristic H_C (Eq. 2). Red indicates high values; blue indicates low values. Third: neural prediction H_{neural}. Fourth: final heuristic H_ϵ (Eq. 4) for $\epsilon = 9$.

implements Neural A*, as in [32]. It returns the nodes E expanded during the A* search given W, H_ϵ (Eq. 4), and the source-target nodes. The two solvers provide two separate gradient signals. As Y is computed following the black-box derivative from [23], its value is differentiable only with respect to W. Following the Neural A* approach [32], instead, the matrix of expanded nodes, E, is differentiable only with respect to H_ϵ. Within the solver module, we effectively combine the two differentiation techniques, enabling the neural module to learn both costs and heuristics with the proper gradient signal. To this end, we stop propagating the gradient of H_ϵ towards W in the computational graph while running the Neural A* solver (dashed arrow in Fig. 2). This is because H_ϵ is evaluated considering the target T, and we want to be sure that T has no influence whatsoever on the target-agnostic costs W.

In principle, having two separate solvers for the evaluation of Y and E may lead to inconsistencies, as H_ϵ, for $\epsilon > 0$, affects the trajectory of the shortest path. In such a case, Y may contain nodes not belonging to E. However, this side-effect is unavoidable during training to guarantee the correct gradient information propagation and to ensure that W does not depend on T. These theoretical reasons are confirmed by a much lower performance during the experiments when trying to include H_ϵ as heuristic function in Black-Box A*. At testing time, to guarantee the output consistency, the solver module is substituted by a standard A* algorithm with W, H_ϵ, and the source-target pair as inputs, returning Y and E in a single execution.

4.3 Loss function

The only label required for training Neural Weighted A* is the ground-truth path $\bar{Y} \in \{0,1\}^{\mathcal{N}}$. In the ideal case, both Y and E are equivalent to $\bar{Y}$, meaning that A* expanded only the nodes belonging to the true shortest path. In a more realistic case, Y is close to $\bar{Y}$ following the same overall course but with minor node differences, while E contains $\bar{Y}$ alongside some nodes from the surrounding area. Since all of these tensors contain binary values, we found the Hamming loss $\mathcal{L}_H$, as in [23], to be the most effective to deal with our learning problem. The final loss is

$$\mathcal{L} = \alpha \cdot \mathcal{L}_H(\bar{Y}, Y) + \beta \cdot \mathcal{L}_H(\bar{Y}, E) \tag{5}$$

Table 1. Datasets' summary statistics.

	Warcraft	Pokémon
Maps I (train, validation, test)	$10000, 1000, 1000$	$3000, 500, 500$
Map resolution Γ	96×96	320×320
Tile resolution	8×8	16×16
Grid shape $\mathcal{N}$	12×12	20×20
Cost range	$[0.8, 9.2]$	$[1.0, 25.0]$
Targets per image	2	2
Sources per target	2	2
Total number of samples	48000	16000

where α and β are positive, real-valued parameters that bring the loss components to the same order of magnitude. A possible alternative to the Hamming loss is the L1 loss, as in [32]. However, we did not find any reason to prefer it over the Hamming loss. The behavior of L1 in terms of gradient propagation is similar, but the experimental results were worse.

5 Data Generation

To experimentally test our claims about Neural Weighted A*, we use two tile-based datasets.[1] The first is a modified version of the Warcraft II dataset from [23] (Sec. 5.1). The second is a novel dataset from the FireRed-LeafGreen Pokémon tileset (Sec. 5.2). In the latter, the search space is bigger, and the tileset is richer, making the setting more complex. However, Neural Weighted A* shows similar performance in both scenarios outperforming the baselines of [23,32]. Tab. 1 collects the summary statistics of the two datasets.

5.1 The Warcraft dataset

The original version of the Warcraft dataset [23] contains paths only from the top-left corner to the bottom-right corner of the image. To make the dataset more challenging, we randomly sampled the source-target pairs from $\mathcal{N}$. For each image-costs pair in the dataset, we chose two target points. The targets lie within a 3-pixel margin from the grid edges. Then, we randomly picked two source points for each target, making four source-target pairs for each map. Each source point is sampled from the quadrant opposite to its target to ensure that each path traverses a moderate portion of the map, as shown in Fig. 4.

5.2 The Pokémon dataset

The Pokémon dataset is a novel, tile-based dataset we present in this paper. It comes with 4000 RGB images of 320×320 pixels generated from Cartographer [29], a random Pokémon map generation tool. Each image is composed

[1] `https://github.com/archettialberto/tilebased_navigation_datasets`

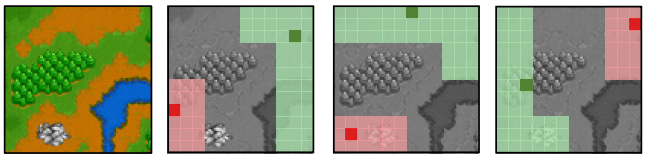

Fig. 4. From left to right, first: image sample from the Warcraft dataset. Second to fourth: examples of valid source-target pairs. Red indicates the target sampling regions, while green indicates the source sampling regions.

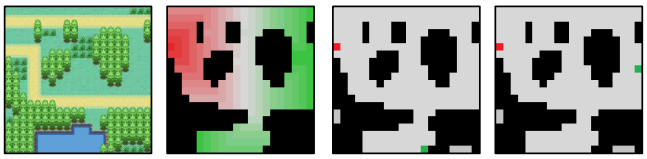

Fig. 5. From left to right, first: image from the Pokémon dataset. Second: wall regions (black), number of steps from the target region (red-to-green gradient), and source-sampling region (green). Third and fourth: examples of valid source-target pairs.

of 400 tiles, each of 16×16 pixels, arranged in a 20×20 pattern. Each tile is linked to a real-valued cost in the interval $[1, 25]$. The training set comprises 3000 image-costs pairs, while the test and validation sets contain 500 pairs each. For each image-costs pair, we sampled two target points, avoiding non-traversable regions in the original Pokémon game, i.e., where $\bar{W}(n) = 25$. We refer to these regions as walls. Then, we sampled two sources for each target, such that the number of steps separating them is at least 12 (Fig. 5).

The Pokémon dataset provides a setting more challenging than Warcraft. First, the number of rows and columns increases from 12 to 20, making the search space nearly four times bigger. Also, the tileset is richer. Warcraft is limited to only five terrain types (grass, earth, forest, water, and stone), and there is a one-to-one correspondence between terrain types and cost values. These aspects make the tile-to-cost patterns very predictable for the neural component of the architectures. Pokémon, on the other hand, has double the number of individual cost values, spread between the tilesets from four different biomes: beach, forest, tundra, and desert. Also, each image sample may contain buildings. The variability in terms of visual features is richer, and similar costs may correspond to tiles exhibiting very different patterns.

6 Experimental Validation

In the following, we describe the experiments to test the validity of our claims. Each time we refer to results obtained with Neural Weighted A*, we note the ϵ value used for the evaluation of H_ϵ.

Table 2. Metrics' definitions.

Metric	Definition
Cost Ratio	$\langle \bar{W}, Y \rangle / \langle \bar{W}, \bar{Y} \rangle$
Generalized Cost Ratio	$\langle \bar{W}, Y^{(\text{rnd_s})} \rangle / \langle \bar{W}, \bar{Y}^{(\text{rnd_s})} \rangle$
Expanded Nodes	$\sum_{n \in \mathcal{N}} E(n)$
Generalized Expanded Nodes	$\sum_{n \in \mathcal{N}} E^{(\text{rnd_s})}(n)$

6.1 Metrics

To measure the path prediction accuracy of the compared architectures, we use the *cost ratio*, as in [23]. In order to account for cost-equivalent paths, we define the cost ratio as the ratio between the predicted path cost and the optimal path cost, according to the ground-truth costs $\bar{W}$. A cost ratio close to 1 indicates that the system produces cost functions correctly generalizing on new maps.

The cost ratio involves paths that start from the sample source and end in the sample target. Our goal, however, is to generate costs and heuristics that are source-agnostic. In principle, the cost function depends only on the image, while the heuristic also considers the target. The source point should not influence any of the two functions. To account for this behavior, we define the *generalized cost ratio* as the cost ratio measured according to $Y^{(\text{rnd_s})}$, i.e., the path prediction from a random source point to the target. This new source is sampled uniformly from the valid sampling regions of the two datasets at each metric evaluation.

To measure the efficiency of the architectures, we simply count how many nodes have been expanded at the end of the A* execution. We refer to this metric as *expanded nodes*. Also, we provide the *generalized expanded nodes* metric to account for randomly sampled sources. Tab. 2 collects the metrics' definitions.

6.2 Experiments

We compare Neural Weighted A* (*NWA**) against the following baselines:

BBA* **(Black-Box A* [23]).** A fully convolutional neural network computes W from I. Then, a Black-Box A* module evaluates the shortest path Y. We follow the implementation of [23], except for the Dijkstra algorithm, substituted by A* with admissible Chebyshev heuristic (Eq. 2).

NA* **(Neural A* [32]).** A fully convolutional neural network computes W from I, using s and t as additional input channels. Then, a Neural A* module [32] evaluates the expanded nodes E. The non-admissible heuristic

$$H_{\text{NA}^*}(n) = D_C(n,t) + 0.001 \cdot D_E(n,t) \tag{6}$$

is used to speed up the search, as in [32]. This heuristic is the weighted sum of the Chebyshev distance D_C and the Euclidean distance D_E between n and t in the grid. The non-admissibility arises from the fact that the scaling term

$w_{\min}$ is missing, differently from Eq. 2. Despite reducing the expanded nodes, this heuristic adds a strong bias towards paths that move straight to the target, greatly penalizing the cost ratio.

***ADM_NA** (Admissible Neural A*).** We propose this architecture as a clone of the original *NA** architecture [32], but we substitute the non-admissible heuristic $H_{\text{NA}*}$ (Eq. 6) with the admissible Chebyshev heuristic H_C (Eq. 2). Our goal is to minimize the influence of the fixed, non-admissible heuristic $H_{\text{NA}*}$ [32] on the expanded nodes E. In this way, the numerical results reflect more accurately the neural predictions, ensuring a fair comparison with *NWA**.

***NS_NA** (No-Source Neural A*).** This architecture is equal to *ADM_NA**, except for the source channel, not included in the neural network input. Differently from *NA** and *ADM_NA**, by hiding the information related to the source node location, we expect *NS_NA** to exhibit no sensible performance downgrade between the cost ratio values and the generalized cost ratio values. The same holds for the expanded nodes and the generalized expanded nodes.

6.3 Implementation details

Each architecture uses convolutional layer blocks from ResNet18 [14], as in [23], to transform tile-based images into cost or heuristic functions, encoded as single-channel tensors. We substitute the first convolution to adapt to the number of input channels, varying between 3 (*BBA**), 4 (*NS_NA**, *NWA**), and 5 (*NA**, *ADM_NA**). In *BBA**, we perform average pooling to reduce the output channels to 1. Then, to ensure that the weights are non-negative, we add a ReLU for Warcraft and a sigmoid for Pokémon. All the baselines involving Neural A* (*NA**, *ADM_NA**, and *NS_NA**), instead, include a 1×1 convolution followed by a sigmoid. Finally, in *NWA**, the channel-reduction strategy depends on the neural network. The cost-predicting ResNet18 is followed by an average operation and a normalization between $w_{\min} = 1$ and $w_{\max} = 10$. The heuristic-predicting ResNet18 is followed by a 1×1 convolution and a normalization in the range $[0, 1]$. Each architecture trains with the Adam optimizer. The learning rate is equal to 0.001. The batch size is 64 for Warcraft and 16 for Pokémon to account for GPU memory usage. The λ parameter of the black-box solvers of *BBA** and *NWA** is 20. The τ parameter of the Neural A* solvers of *NA**, *ADM_NA**, *NS_NA**, and *NWA** is set to the square root of the grid width, so 3.46 for Warcraft and 4.47 for Pokémon. In Eq. 5, we impose $\alpha = 1$ and $\beta = 0.1$ to bring the loss components to the same order of magnitude. We found that the training procedure is not affected by small deviations from the parameters described in this section. Finally, we detected sensible improvements in the efficiency when training *NWA** with random ϵ, as it expanded noticeably fewer nodes than training with fixed ϵ. Since the other metrics do not exhibit sensible differences, we always refer to the test results of *NWA** obtained after training with ϵ sampled from $[0, 9]$ at each H_ϵ evaluation. To ensure the full reproducibility of our experiments, we share the source code.[2]

[2] `https://github.com/archettialberto/neural_weighted_a_star`

Table 3. Quantitative results on the Warcraft and Pokémon datasets.

		Warcraft dataset				Pokémon dataset			
Experiment	ϵ	CR	Gen. CR	EN	Gen. EN	CR	Gen. CR	EN	Gen. EN
*BBA**	-	**1.0**	**1.0**	69.8	69.94	1.57	1.65	79.78	79.29
*NA**	-	1.29	1.41	**9.81**	**9.82**	2.15	2.57	**15.02**	**14.64**
*ADM_NA**	-	1.04	1.17	13.12	22.48	**1.11**	1.24	26.57	37.03
*NS_NA**	-	1.12	1.11	21.19	21.19	1.16	**1.17**	39.76	39.44
*NWA**	0.0	**1.0**	**1.0**	68.42	69.04	1.06	1.05	124.06	121.2
*NWA**	1.0	1.01	1.01	49.54	50.57	**1.03**	**1.03**	80.21	78.43
*NWA**	4.0	1.03	1.03	26.61	27.15	1.08	1.06	56.4	56.21
*NWA**	9.0	1.1	1.09	14.2	14.47	1.22	1.14	31.54	31.04
*NWA**	11.0	1.13	1.11	12.74	13.0	1.22	1.24	23.37	23.06
*NWA**	14.0	1.15	1.13	**11.94**	**12.12**	1.22	1.35	**20.09**	**19.84**

6.4 Results

We collect the quantitative results of our experiments in Tab. 3. The table is split into four sections, two for each dataset. Each section contains either baseline experiments or *NWA**-related experiments measured on the same *NWA** architecture fixing different ϵ values at testing time. We comment on the experiments' results by answering the following three questions:

Does NWA* learn to predict cost functions correctly? By observing Table 3, *NWA** reaches a nearly perfect cost ratio on both datasets for a considerable range of ϵ values. This was expected for $\epsilon \approx 0$, but the (generalized) cost ratio remains very low for ϵ up to 4, which is an excellent result considering the corresponding expanded nodes speedup. In Warcraft, *NWA** behaves as *BBA** cost-ratio-wise, while, in Pokémon, it outperforms all the baselines. Since, by setting ϵ to low values, the path predictions do not take into account H_ϵ, the positive cost-ratio-related performance implies that *NWA** learned to predict cost functions that make A* return paths close to the ground-truth.

Does NWA* learn to predict heuristic functions correctly? By setting $\epsilon \gg 0$, H_ϵ drives the search. A* converges faster, but it may return suboptimal paths. Therefore, we expect a small penalty on cost ratios, but a noticeable decrease in the node expansions. Again, the empirical results confirm this trend on both datasets. For $\epsilon = 14$, we outperform all the baselines in terms of generalized node expansions. The only exception is *NA**, which expands fewer nodes, but exhibits an extremely higher cost ratio. *NWA** trades off few node expansions to be much more reliable than *NA** in terms of path predictions.

Can NWA* trade off planning accuracy for efficiency? By setting ϵ close to 0, *NWA** behaves accurately (low cost ratio, high expanded nodes). By increasing ϵ, *NWA** becomes more efficient (higher cost ratio, lower expanded nodes). To visually illustrate the extent of the tradeoff capabilities of *NWA**, we

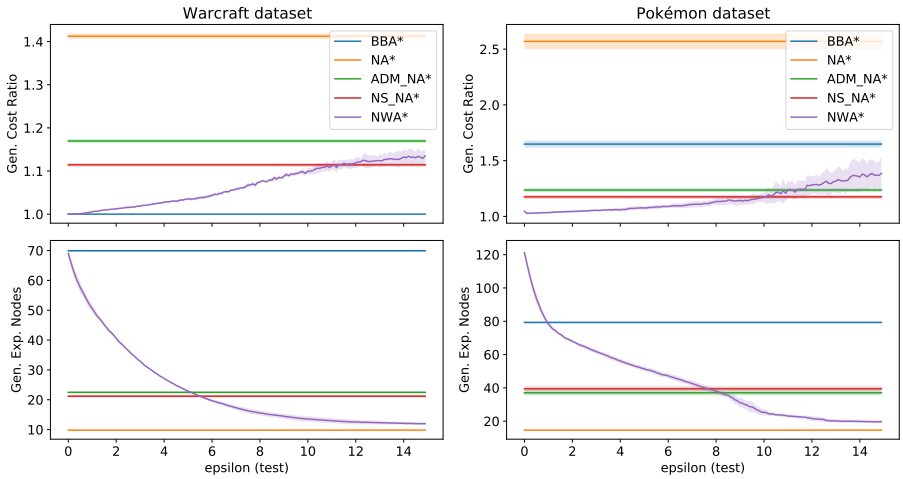

Fig. 6. Comparison between generalized cost ratio and generalized expanded nodes across the experiments for several ϵ values (mean $\pm$ std over five restarts). For low ϵ, *NWA** is the most accurate model, while for higher ϵ, it is the most efficient. The only exception is *NA**, which is barely faster but much less reliable in terms of cost ratio.

plot in Fig. 6 the generalized metrics (y-axis) for all the experiments with respect to several ϵ values (x-axis). Since the baseline architectures do not depend on ϵ, their behavior is plotted as a horizontal line for comparison. *NWA**, on the other hand, smoothly interpolates between the accuracy of *BBA** and the efficiency of *NA**-related architectures, offering to the user the possibility of finely tuning ϵ to the desired planning behavior, from the most accurate to the most efficient.

7 Conclusions

With Neural Weighted A*, we propose a differentiable, anytime shortest path solver able to learn graph costs and heuristics for planning on raw image inputs. The system trains with direct supervision on planning examples, making data labeling cheap. Unlike any similar data-driven planner, we can choose to return the optimal solution or to trade off accuracy for convergence speed by tuning a single, real-valued parameter, even at runtime. We guarantee the solution sub-optimality to be constrained within a linear bound proportional to the tradeoff parameter. We experimentally test the validity of our claims on two tile-based datasets. By inspecting the numerical results, we see that Neural Weighted A* consistently outperforms the accuracy and the efficiency of the previous works, obtaining, in a single architecture, the best of the two worlds.

References

1. Amos, B., Jimenez, I., Sacks, J., Boots, B., Kolter, J.Z.: Differentiable mpc for end-to-end planning and control. In: Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., Garnett, R. (eds.) Advances in Neural Information Processing Systems. vol. 31. Curran Associates, Inc. (2018)
2. Amos, B., Kolter, J.Z.: Optnet: Differentiable optimization as a layer in neural networks. In: International Conference on Machine Learning. pp. 136–145. PMLR (2017)
3. Amos, B., Xu, L., Kolter, J.Z.: Input convex neural networks. In: International Conference on Machine Learning. pp. 146–155. PMLR (2017)
4. de Avila Belbute-Peres, F., Smith, K., Allen, K., Tenenbaum, J., Kolter, J.Z.: End-to-end differentiable physics for learning and control. In: Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., Garnett, R. (eds.) Advances in Neural Information Processing Systems. vol. 31. Curran Associates, Inc. (2018)
5. Bello, I., Pham, H., Le, Q.V., Norouzi, M., Bengio, S.: Neural combinatorial optimization with reinforcement learning. International Conference on Learning Representations, Workshop Track (2016)
6. Berthet, Q., Blondel, M., Teboul, O., Cuturi, M., Vert, J.P., Bach, F.: Learning with differentiable pertubed optimizers. In: Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M.F., Lin, H. (eds.) Advances in Neural Information Processing Systems. vol. 33, pp. 9508–9519. Curran Associates, Inc. (2020)
7. Chen, L.C., Schwing, A., Yuille, A., Urtasun, R.: Learning deep structured models. In: International Conference on Machine Learning. pp. 1785–1794. PMLR (2015)
8. Choudhury, S., Bhardwaj, M., Arora, S., Kapoor, A., Ranade, G., Scherer, S., Dey, D.: Data-driven planning via imitation learning. The International Journal of Robotics Research **37**, 1632–1672 (2018)
9. Deudon, M., Cournut, P., Lacoste, A., Adulyasak, Y., Rousseau, L.M.: Learning heuristics for the tsp by policy gradient. In: International conference on the integration of constraint programming, artificial intelligence, and operations research. pp. 170–181. Springer (2018)
10. Dijkstra, E.W., et al.: A note on two problems in connexion with graphs. Numerische mathematik **1**(1), 269–271 (1959)
11. East, S., Gallieri, M., Masci, J., Koutnik, J., Cannon, M.: Infinite-horizon differentiable model predictive control. In: International Conference on Learning Representations (2020)
12. Hansen, E.A., Zhou, R.: Anytime heuristic search. Journal of Artificial Intelligence Research **28**, 267–297 (2007)
13. Hart, P.E., Nilsson, N.J., Raphael, B.: A formal basis for the heuristic determination of minimum cost paths. IEEE transactions on Systems Science and Cybernetics **4**(2), 100–107 (1968)
14. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 770–778 (2016)
15. Karkus, P., Ma, X., Hsu, D., Kaelbling, L.P., Lee, W.S., Lozano-Perez, T.: Differentiable algorithm networks for composable robot learning. In: Robotics: Science and Systems (RSS) (2019)
16. Kato, H., Beker, D., Morariu, M., Ando, T., Matsuoka, T., Kehl, W., Gaidon, A.: Differentiable rendering: A survey. ArXiv **abs/2006.12057** (2020)

17. Liu, Z., Li, X., Luo, P., Loy, C.C., Tang, X.: Semantic image segmentation via deep parsing network. In: Proceedings of the IEEE international conference on computer vision. pp. 1377–1385 (2015)
18. Nazari, M., Oroojlooy, A., Snyder, L., Takac, M.: Reinforcement learning for solving the vehicle routing problem. In: Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., Garnett, R. (eds.) Advances in Neural Information Processing Systems. vol. 31. Curran Associates, Inc. (2018)
19. Paden, B., Cáp, M., Yong, S.Z., Yershov, D.S., Frazzoli, E.: A survey of motion planning and control techniques for self-driving urban vehicles. IEEE Transactions on Intelligent Vehicles **1**, 33–55 (2016)
20. Paschalidou, D., Ulusoy, O., Schmitt, C., Van Gool, L., Geiger, A.: Raynet: Learning volumetric 3d reconstruction with ray potentials. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 3897–3906 (2018)
21. Pearl, J., Kim, J.H.: Studies in semi-admissible heuristics. IEEE transactions on pattern analysis and machine intelligence pp. 392–399 (1982)
22. Pitis, S., Chan, H., Jamali, K., Ba, J.: An inductive bias for distances: Neural nets that respect the triangle inequality. In: International Conference on Learning Representations (2020)
23. Pogančić, M.V., Paulus, A., Musil, V., Martius, G., Rolinek, M.: Differentiation of blackbox combinatorial solvers. In: International Conference on Learning Representations (2019)
24. Rios, L.H.O., Chaimowicz, L.: A survey and classification of A* based best-first heuristic search algorithms. In: da Rocha Costa, A.C., Vicari, R.M., Tonidandel, F. (eds.) Advances in Artificial Intelligence – SBIA 2010. pp. 253–262. Springer Berlin Heidelberg, Berlin, Heidelberg (2010)
25. Russell, S.J., Norvig, P.: Artificial intelligence - a modern approach: the intelligent agent book. Prentice Hall series in artificial intelligence, Prentice Hall (1995)
26. Seo, S., Liu, Y.: Differentiable physics-informed graph networks. ArXiv **abs/1902.02950** (2019)
27. Smith, C., Karayiannidis, Y., Nalpantidis, L., Gratal, X., Qi, P., Dimarogonas, D.V., Kragic, D.: Dual arm manipulation—a survey. Robotics and Autonomous systems **60**(10), 1340–1353 (2012)
28. Taccari, L.: Integer programming formulations for the elementary shortest path problem. European Journal of Operational Research **252**(1), 122–130 (2016)
29. Unbayleefable: Cartographer (2021), `https://www.pokecommunity.com/showthread.php?t=429142`
30. Wang, P.W., Donti, P., Wilder, B., Kolter, Z.: Satnet: Bridging deep learning and logical reasoning using a differentiable satisfiability solver. In: International Conference on Machine Learning. pp. 6545–6554. PMLR (2019)
31. Yang, F., Yang, Z., Cohen, W.W.: Differentiable learning of logical rules for knowledge base reasoning. In: Guyon, I., Luxburg, U.V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., Garnett, R. (eds.) Advances in Neural Information Processing Systems. vol. 30. Curran Associates, Inc. (2017)
32. Yonetani, R., Taniai, T., Barekatain, M., Nishimura, M., Kanezaki, A.: Path planning using Neural A* search. ArXiv **abs/2009.07476** (2020)
33. Zhang, Z., Cui, P., Zhu, W.: Deep learning on graphs: A survey. ArXiv **abs/1812.04202** (2018)
34. Zheng, S., Jayasumana, S., Romera-Paredes, B., Vineet, V., Su, Z., Du, D., Huang, C., Torr, P.H.: Conditional random fields as recurrent neural networks. In: Proceedings of the IEEE international conference on computer vision. pp. 1529–1537 (2015)