Path Planning Using Neural A* Search

Feb 25th, 2026

Oliver Berton

Paper Meta

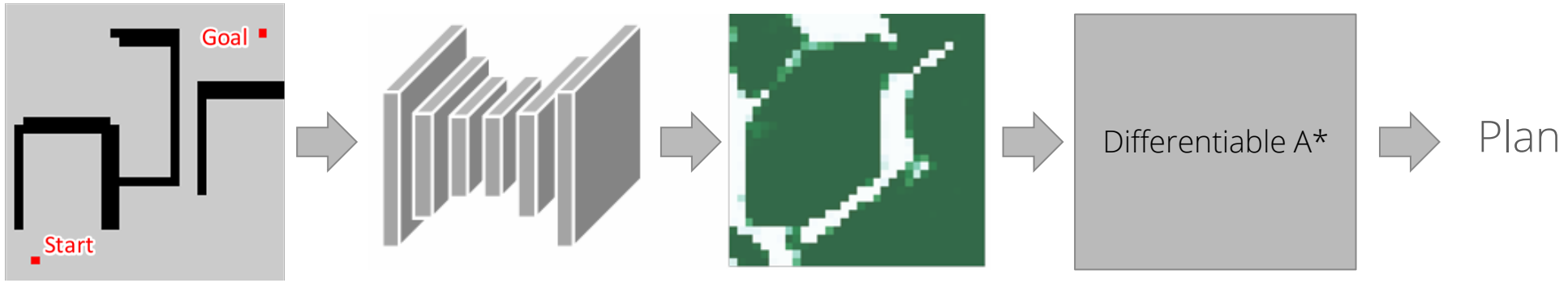
- **Authors:** Ryo Yonetani, Tatsunori Taniai, Mohammadamin Barekatain, Mai Nishimura, Asako Kanezaki
- **ICML 2021**
- **Paper Link:** Path Planning using Neural A* Search

Motivation

- **Advantages of data-driven path planning**
  - Find near-optimal plans more efficiently than classical planners on point-to-point shortest path problems
  - Enable path planning on raw image inputs without the need for pixel-wise labeling
- **Advantages of classical search-based planning (e.g. A*)**
  - Optimality and completeness guarantees
- **Neural A*: Combine advantages of data-driven and search-based planning**

Neural A* Overview

- Combine data-driven path planning (i.e. learning) with search-based planning
- Reformulate A* search to be differentiable and couple it with a convolutional encoder to form an end-to-end trainable neural network planner
- Train this neural network using expert provided ground truth plans



Related Work

Area	Prior Work	Pros	Cons
Sampling-Based Planning	Use data-driven methods to learn most efficient/effective way to sample configuration space	• Can work in high-dimensional state spaces • Asymptotically complete	• No optimality guarantee
Reactive Planners (map current state directly to an action)	Learn policy for moving agents via supervised learning. Use inverse-RL to recover cost map	• Extremely Fast • Useful in dynamic environments	• No completeness or optimality guarantees • Short-sighted, poor performance on large maps

Related Work

Area	Prior Work	Pros	Cons
Search based planning with classical heuristics	Improve heuristic function, or tune search algorithm for increased performance	• Optimality and completeness guarantees	• Can be slow, especially in high-dimensional C-spaces
Search based planning extended with data-driven methods	Learn heuristic or cost functions from demonstrations Learn to plan directly on raw image inputs Apply black-block optimization to solver	• Efficient near-optimal search	• Fine grained annotation required • Treating search as black box makes training difficult

Related Work - SAIL/SAIL-SL

- **Core Concept:** Transforms heuristic design into an Imitation Learning problem, training a model to mimic an Oracle solution (e.g., Dijkstra with full map knowledge).
- **Feature Representation:** Maps nodes to hand-engineered feature vectors encoding Information Gain (occupancy density) and Goal Proximity (geometric distance).
- **Learned Heuristic**: A Neural Network learns the mapping from these features to the predicted Cost-to-Go for neighboring cells.

Pros:

- Generalization: features are relative, so can work on unseen environments
- Efficient search compared to traditional heuristics
- Completeness guarantee

Cons

- Dependency on Oracle
- Requires feature engineering, if hand-engineered features do not capture complexity of the environment, heuristic will fail
- No admissibility/consistency guarantees

Related Work - Blackbox Differentiation

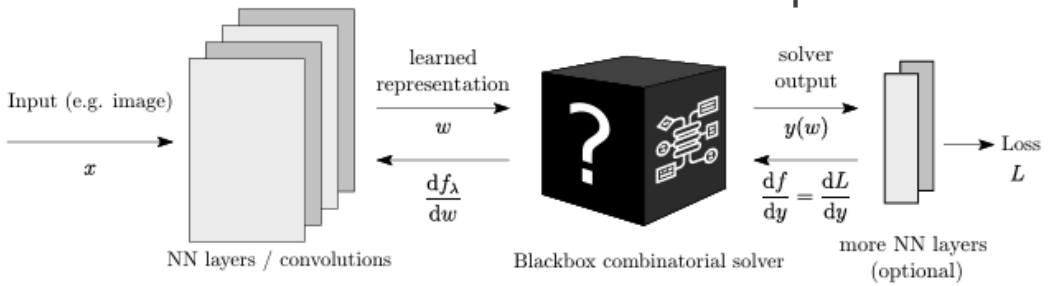
- **Core Concept:** Embed blackbox combinatorial solver into neural network
- **Differentiation:** To find surrogate gradient of solver, perturb input to solver by λ and get new output. Use numerical differentiation to calculate derivative
- **Planning (BB-A*):** They used the architecture with A* as the combinatorial solver to compute minimum cost path between two points. Input to A* is a learned cost map, and the output was the plan, with the distance between the ground truth and actual path used as the loss for training

Pros:

- Simple to implement
- Can work on arbitrary inputs (ex. rgb image inputs)
- Completeness guarantee

Cons:

- Black-box nature of solver makes training difficult (can be sparse signals)
- Computational complexity
- No path solution optimality guarantees



Methodology

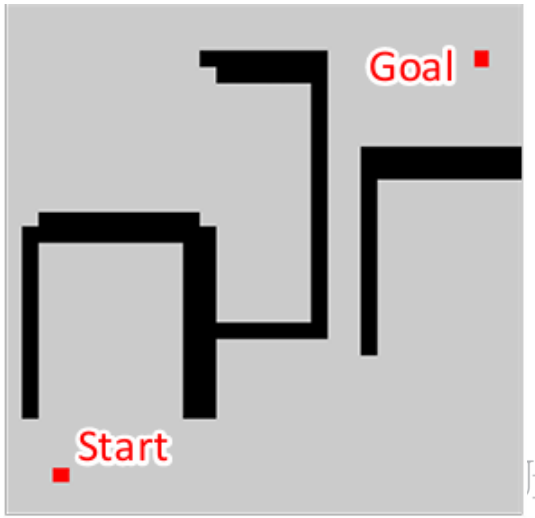
Assumptions

- Static environment during search
- Environment discretized into 2D grid map
- Unit Node Costs
  - **Assume cost of each non-obstacle cell in input map is 1**
  - Simplifies task of encoder, as NN does not need to learn complex cost landscape

Problem Definition

Two Scenarios of Path Planning with Neural A*

1) **Point-to-point shortest path Search**
   - Input is binary map, where 0 represents obstacles and 1 free space
   - Ground truth given by Dijkstra
2) **Path planning on raw image inputs**
   - Input is normalized rgb image
   - Ground truth given by human annotators



Problem Definition

Graph: $G = (V, E)$

Input: $X \in \{0,1\}^v \; or \; X \in [0,1]^{3*v}$

Problem Instance: $Q^{(i)} = (X^{(i)}, v_s^{(i)}, v_g^{(i)})$

Guidance Map: $\mathbf{\Phi^{(i)}} \in \mathbb{R}^{\boldsymbol{v}}$,learned cost of each node

Ground Truth Path: $\overline{\boldsymbol{P}}^{(\boldsymbol{i})} \in \{\mathbf{0}, \mathbf{1}\}^{\boldsymbol{v}}$ where elements are 1 along path

Inner Product: $\langle A, B \rangle$ is the inner product of A and B

Problem Definition

Graph: $G = (V, E)$

Input: $X \in \{0,1\}^v \; or \; X \in [0,1]^{3*v}$

Problem Instance: $Q^{(i)} = (X^{(i)}, v_s^{(i)}, v_g^{(i)})$

Learned cost map that "guides" A* search to efficient solution

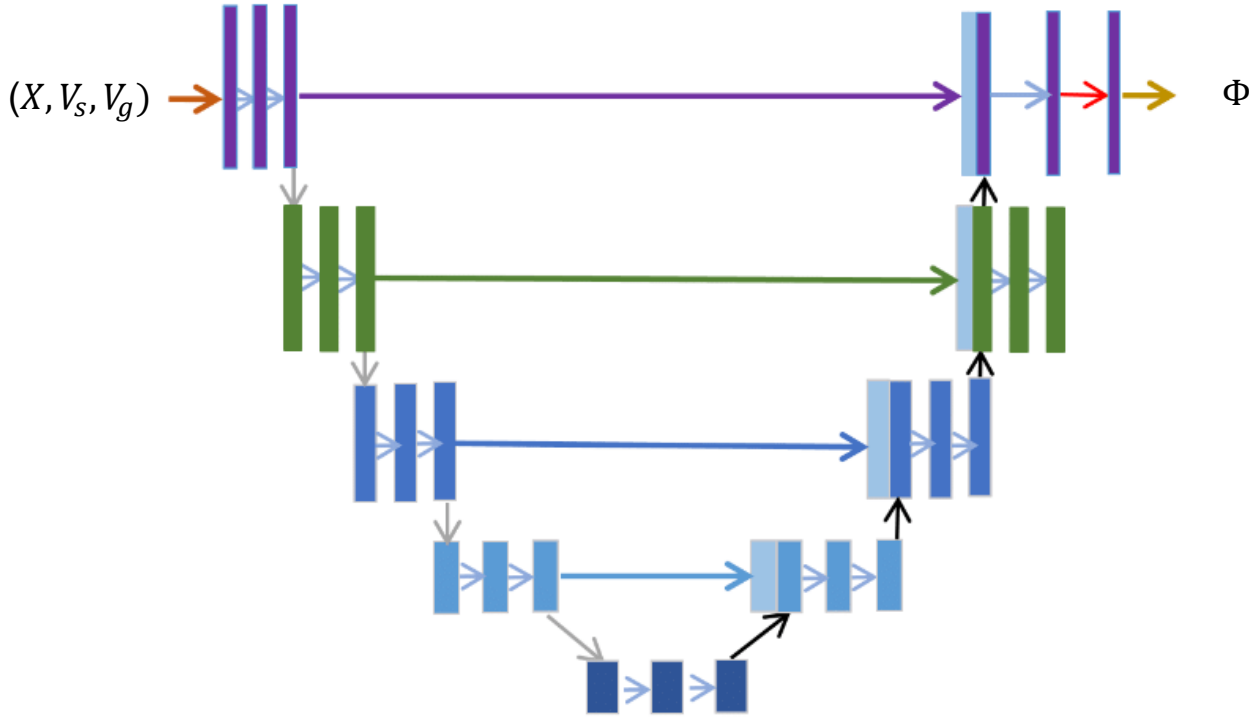
Guidance Map: $\mathbf{\Phi^{(i)}} \in \mathbb{R}^{v}$,learned cost of each node

Ground Truth Path: $\overline{\boldsymbol{P}}^{(i)} \in \{\mathbf{0},\mathbf{1}\}^{v}$ where elements are 1 along path

Inner Product: $\langle A, B \rangle$ is the inner product of A and B

Guidance Map

Maps Q to a learned guidance map Φ



Vanilla A* Search

O: Open Set, C: Closed Set, $N(v)$: set of neighbor nodes of v

Algorithm 1 A* Search

Input: Graph $\mathcal{G}$, movement cost c, start v_s, and goal
Output: Shortest path P

1: Initialize $\mathcal{O} \leftarrow v_s, \mathcal{C} \leftarrow \emptyset$, $\texttt{Parent}(v_s) \leftarrow \emptyset$.
2: **while** $v_g \notin \mathcal{C}$ **do**
3: Select $v^* \in \mathcal{O}$ based on Eq. (1).
4: Update $O \leftarrow O \setminus v^*, \mathcal{C} \leftarrow \mathcal{C} \cup v^*$.
5: Extract $\mathcal{V}_{\text{nbr}} \subset \mathcal{V}$ based on Eq. (2).
6: **for each** $v' \in \mathcal{V}_{\text{nbr}}$ **do**
7: Update $\mathcal{O} \leftarrow \mathcal{O} \cup v'$, $\texttt{Parent}(v') \leftarrow v^*$.
8: **end for**
9: **end while**
10: $P \leftarrow \texttt{Backtrack}(\texttt{Parent}, v_g)$.

$$v^* = \arg\min_{v \in \mathcal{O}} \left(g(v) + h(v)\right), \tag{1}$$

$$\mathcal{V}_{\text{nbr}} = \{v' \mid v' \in \mathcal{N}(v^*) \wedge v' \notin \mathcal{O} \wedge v' \notin \mathcal{C}\}. \tag{2}$$

Making A* Differentiable

1) Convert sets and variables into matrices

$$O, C, V_{nbr} \rightarrow O, C, V_{nbr} \in \{0,1\}^v$$

$v_s, v_g, v^* \rightarrow V_s, V_g, V^* \in \{0,1\}^v$ where v^* is the current node

$$g(v), h(v), \phi(v) \rightarrow G, H, \Phi \in \mathbb{R}^v$$

Making A* Differentiable

1) Convert sets and variables into matrices

Binary matrices with the same map size as X that indicate nodes contained in the set

$$O, C, V_{nbr} \rightarrow \boxed{O, C, V_{nbr} \in \{0,1\}^v}$$

$v_s, v_g, v^* \rightarrow V_s, V_g, V^* \in \{0,1\}^v$ where v^* is the current node

$$g(v), h(v), \phi(v) \rightarrow G, H, \Phi \in \mathbb{R}^v$$

Making A* Differentiable

1) Convert sets and variables into matrices

$$O, C, V_{nbr} \rightarrow O, C, V_{nbr} \in \{0,1\}^v$$

$$v_s, v_g, v^* \rightarrow V_s, V_g, V^* \in \{0,1\}^v$$ where v^* is the current node

$$g(v), h(v), \phi(v) \rightarrow G, H, \Phi \in \mathbb{R}^v$$

One-hot matrices of size X

Making A* Differentiable

1) Convert sets and variables into matrices

$$O, C, V_{nbr} \rightarrow O, C, V_{nbr} \in \{0,1\}^v$$

$$v_s, v_g, v^* \rightarrow V_s, V_g, V^* \in \{0,1\}^v \text{ where } v^* \text{ is the current node}$$

$$g(v), h(v), \phi(v) \rightarrow G, H, \Phi \in \mathbb{R}^v$$

Matrices with the same map size as X that store g values, h values, and guidance costs

Making A* Differentiable

2) Node Selection

$$v^* = \arg\min_{v \in \mathcal{O}} \left(g(v) + h(v) \right), \longrightarrow V^* = \mathcal{I}_{\max} \left(\frac{\exp(-(G+H)/\tau) \odot O}{\langle \exp(-(G+H)/\tau), O \rangle} \right)$$

$\boldsymbol{\tau}$**:** Temperature Parameter

$\boldsymbol{I_{max}(A)}$**:** function that gives the argmax index of A as a binary one-hot matrix during a forward pass, while acting as the identity function for back-propagation. This bypasses the non-differentiability of an argmax

Making A* Differentiable

2) Node Selection

$$v^* = \arg\min_{v \in \mathcal{O}} \left(g(v) + h(v)\right),$$

$$V^* = \mathcal{I}_{\max}\left(\frac{\exp(-(G+H)/\tau) \odot O}{\langle \exp(-(G+H)/\tau), O \rangle}\right)$$

Softmax over elements in open list

τ: Temperature Parameter

$\boldsymbol{I_{max}(A)}$: function that gives the argmax index of A as a binary one-hot matrix during a forward pass, while acting as the identity function for back-propagation. This bypasses the non-differentiability of an argmax

Neural A* Algorithm

3) Node Expansions (i.e. replacing $\mathcal{V}_{\text{nbr}} = \{v' \mid v' \in \mathcal{N}(v^*) \wedge v' \notin \mathcal{O} \wedge v' \notin \mathcal{C}\}$ **)**

$$K = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

If X is binary map:

$$V_{\text{nbr}} = (V^* * K) \odot X \odot (\mathbb{1} - O) \odot (\mathbb{1} - C)$$

$$\bar{V}_{\text{nbr}} = (V^* * K) \odot X \odot O \odot (\mathbb{1} - C)$$

If X is an rgb image:

$$V_{\text{nbr}} = (V^* * K) \odot (\mathbb{1} - O) \odot (\mathbb{1} - C)$$

$$\bar{V}_{\text{nbr}} = (V^* * K) \odot O \odot (\mathbb{1} - C)$$

Neural A* Algorithm

3) Node Expansions (i.e. replacing $\mathcal{V}_{\text{nbr}} = \{v' \mid v' \in \mathcal{N}(v^*) \wedge v' \notin \mathcal{O} \wedge v' \notin \mathcal{C}\}$)

$$K = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Exclude values in closed set

Convolve

If X is binary map:

$$V_{\text{nbr}} = (V^* * K) \odot X \odot (\mathbb{1} - O) \odot (\mathbb{1} - C)$$

$$\bar{V}_{\text{nbr}} = (V^* * K) \odot X \odot O \odot (\mathbb{1} - C)$$

If X is an rgb image:

$$V_{\text{nbr}} = (V^* * K) \odot (\mathbb{1} - O) \odot (\mathbb{1} - C)$$

$$\bar{V}_{\text{nbr}} = (V^* * K) \odot O \odot (\mathbb{1} - C)$$

Neighbors already in open list

Neural A* Algorithm

4) Updating G

$$G' = \langle G, V^* \rangle \cdot \mathbb{1} + \Phi$$

$$\begin{aligned} G \leftarrow\ & G' \odot V_{\mathrm{nbr}} + \min(G', G) \odot \bar{V}_{\mathrm{nbr}} \\ & + G \odot (\mathbb{1} - V_{\mathrm{nbr}} - \bar{V}_{\mathrm{nbr}}), \end{aligned}$$

Neural A* Algorithm

4) Updating G

Compute candidate g based on current node and guidance map

$$G' = \langle G, V^* \rangle \cdot \mathbb{1} + \Phi$$

$$G \leftarrow G' \odot V_{\mathrm{nbr}} + \min(G', G) \odot \bar{V}_{\mathrm{nbr}} + G \odot (\mathbb{1} - V_{\mathrm{nbr}} - \bar{V}_{\mathrm{nbr}}),$$

Neural A* Algorithm

4) Updating G

$$G' = \langle G, V^* \rangle \cdot \mathbb{1} + \Phi$$

If neighbor not in open list, set G to G'

$$G \leftarrow G' \odot V_{\text{nbr}} + \min(G', G) \odot \bar{V}_{\text{nbr}} + G \odot (\mathbb{1} - V_{\text{nbr}} - \bar{V}_{\text{nbr}}),$$

Neural A* Algorithm

4) Updating G

$$G' = \langle G, V^* \rangle \cdot \mathbb{1} + \Phi$$

$$G \leftarrow G' \odot V_{\mathrm{nbr}} + \min(G', G) \odot \bar{V}_{\mathrm{nbr}} + G \odot (\mathbb{1} - V_{\mathrm{nbr}} - \bar{V}_{\mathrm{nbr}}),$$

If neighbor already in open list, use min of previously computed G and G'

Neural A* Algorithm

4) Updating G

$$G' = \langle G, V^* \rangle \cdot \mathbb{1} + \Phi$$

$$G \leftarrow G' \odot V_{\mathrm{nbr}} + \min(G', G) \odot \bar{V}_{\mathrm{nbr}} + G \odot (\mathbb{1} - V_{\mathrm{nbr}} - \bar{V}_{\mathrm{nbr}}),$$

Keep all other g values the same

Neural A* Algorithm

5) Update O and C, enable mini-batches

Binary goal classification flag: $\eta^{(i)} = 1 - \langle V_{\mathrm{g}}^{(i)'}, V^{*(i)} \rangle$

To update O and C: $O^{(i)} \leftarrow O^{(i)} - \eta^{(i)} V^{*(i)}, \;\; C^{(i)} \leftarrow C^{(i)} + \eta^{(i)} V^{*(i)}$

Neural A* Algorithm

5) Update O and C, enable mini-batches

Binary goal classification flag: $\eta^{(i)} = 1 - \langle V_{\mathrm{g}}^{(i)}, V^{*(i)} \rangle$

To update O and C: $O^{(i)} \leftarrow O^{(i)} - \eta^{(i)} V^{*(i)}, \;\; C^{(i)} \leftarrow C^{(i)} + \eta^{(i)} V^{*(i)}$

Keeps O and C constant once a solution is found

Neural A* Algorithm

Now, every operation in A* search is differentiable: can backpropagate through it!

Training Procedure

Forward Pass:

Algorithm 2 Neural A* Search

Input: Problem instances $\{Q^{(i)} = (X^{(i)}, v_{\mathrm{s}}^{(i)}, v_{\mathrm{g}}^{(i)}) \mid i = 1, \ldots, b\}$ in a mini-batch of size b.

Output: Closed-list matrices $\{C^{(i)} \mid i = 1, \ldots, b\}$ and solution paths $\{P^{(i)} \mid i = 1, \ldots, b\}$.

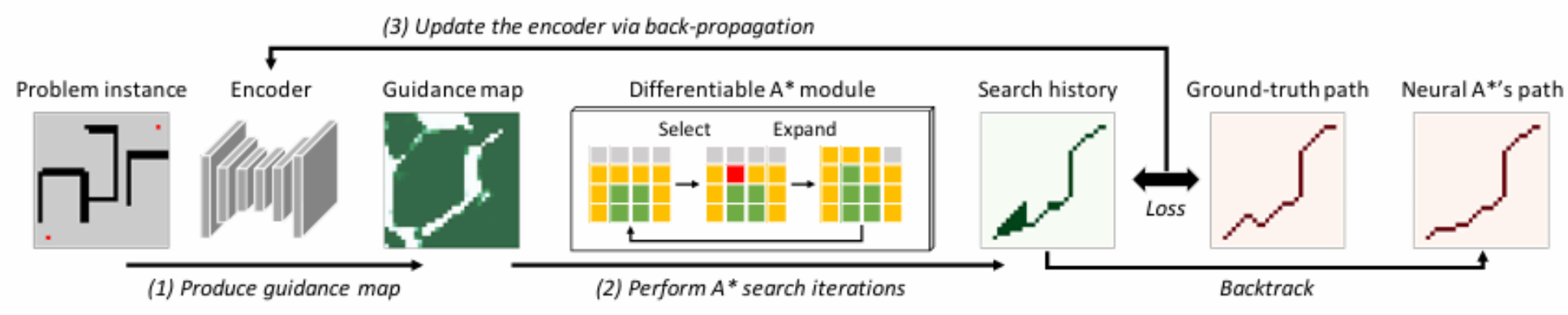
1: **for all** $i = 1, \ldots, b$ **do in parallel**
2: Compute $V_{\mathrm{s}}^{(i)}, V_{\mathrm{g}}^{(i)}$ from $v_{\mathrm{s}}^{(i)}, v_{\mathrm{g}}^{(i)}$.
3: Compute $\Phi^{(i)}$ from $X^{(i)}, V_{\mathrm{s}}^{(i)}, V_{\mathrm{g}}^{(i)}$ by the encoder.
4: Initialize $O^{(i)} \leftarrow V_{\mathrm{s}}^{(i)}$, $C^{(i)} \leftarrow \mathbf{0}$, $G^{(i)} \leftarrow \mathbf{0}$.
5: Initialize $\texttt{Parent}^{(i)}(v_{\mathrm{s}}^{(i)}) \leftarrow \emptyset$.
6: **end for**
7: **repeat**
8: **for all** $i = 1, \ldots, b$ **do in parallel**
9: Select $V^{*(i)}$ based on Eq. (3).
10: Compute $\eta^{(i)} = 1 - \langle V_{\mathrm{g}}^{(i)}, V^{*(i)} \rangle$.
11: Update $O^{(i)}$ and $C^{(i)}$ based on Eq. (8).
12: Compute $V_{\mathrm{nbr}}^{(i)}$ based on Eq. (4).
13: Update $O^{(i)} \leftarrow O^{(i)} + V_{\mathrm{nbr}}^{(i)}$.
14: Update $G^{(i)}$ based on Eq. (5) and Eq. (6).
15: Update $\texttt{Parent}^{(i)}$ based on Algorithm 1-L6,7.
16: **end for**
17: **until** $\eta^{(i)} = 0$ for $i = 1, \ldots, b$
18: **for all** $i = 1, \ldots, b$ **do in parallel**
19: $P^{(i)} \leftarrow \texttt{Backtrack}(\texttt{Parent}^{(i)}, v_{\mathrm{g}}^{(i)})$.
20: **end for**

Training Procedure

Backward Pass:

Loss: $\mathcal{L} = \|C - \bar{P}\|_1 / |\mathcal{V}|$

Loss penalizes both falsely expanded nodes, and not expanding nodes on optimal path



Results

Experimental Setup

Datasets for point-to-point shortest path search

- Motion Planning (MP) Dataset
  - Binary grid-world environments
  - Environments resized to 32x32
- Tiled MP Dataset
  - Extension of MP dataset with more complex and diverse obstacles
  - 64x64 environmental map size
- City/Street Map (CSM) Dataset
  - 30 city maps with explicit obstacle annotations as binary images
  - Crop random 128x128 patch from images and resize to 64x64

Datasets for planning on raw image inputs

- Stanford Drone Dataset (SSD)
  - convert paths pedestrians took on surveillance videos to annotated RGB images

Experimental Setup

Setup:

- Randomly sampled start and goal points for each map
- Obtain ground truth shortest-path for each environment using Dijkstra search or human annotators
- Use Chebyshev distance for heuristic function, i.e $\max\left((x - x_g), (y - y_g)\right)$

Metrics

- **Path Optimality Ratio (Opt):** percentage of generated plans that are identical to the ground truth path
- **Reduction ratio of node explorations (Exp):** reduction in node expansions of planner as compared to vanilla A*
- **Harmonic mean (Hmean):** harmonic mean of Opt and Exp. Quantifies extent of tradeoff between optimality and efficiency
- **Chamfer distance:** average spatial distance between predicted path and ground truth path

Experimental Setup

Baselines:

- SAIL/SAIL-SL
- BB-A*
- Greedy best-first search (BF)
- Weighted A* (WA*)
- Neural BF (greedy best-first search using guidance cost)

Results - Point to Point Shortest Path Planning

MP Dataset

	Opt	Exp	Hmean
BF	65.8 (63.8, 68.0)	44.1 (42.8, 45.5)	44.8 (43.4, 46.3)
WA *	68.4 (66.5, 70.4)	35.8 (34.5, 37.1)	40.4 (39.0, 41.8)
SAIL	34.6 (32.1, 37.0)	**48.6 (47.2, 50.2)**	26.3 (24.6, 28.0)
SAIL-SL	37.2 (34.8, 39.5)	46.3 (44.8, 47.8)	28.3 (26.6, 29.9)
BB-A*	62.7 (60.6, 64.9)	42.0 (40.6, 43.4)	42.1 (40.5, 43.6)
Neural BF	75.5 (73.8, 77.1)	45.9 (44.6, 47.2)	**52.0 (50.7, 53.4)**
Neural A*	**87.7 (86.6, 88.9)**	40.1 (38.9, 41.3)	**52.0 (50.7, 53.3)**

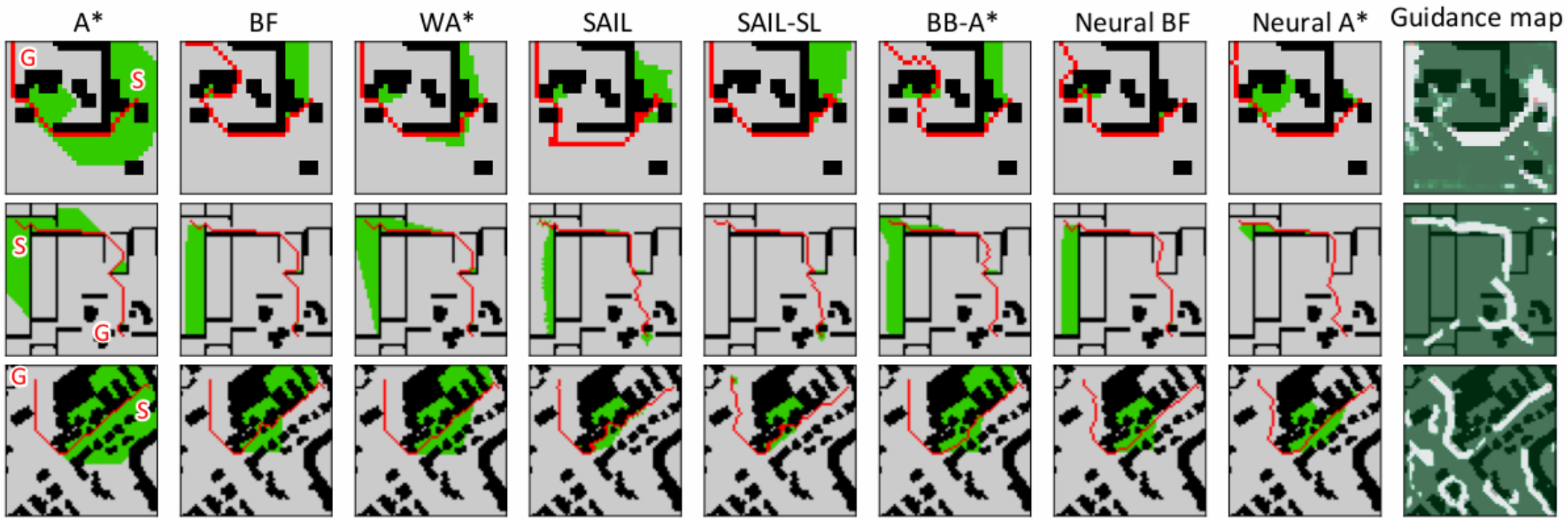
Tiled MP Dataset

	Opt	Exp	Hmean
BF	32.3 (30.0, 34.6)	58.9 (57.1, 60.8)	34.0 (32.1, 36.0)
WA*	35.3 (32.9, 37.7)	52.6 (50.8, 54.5)	34.3 (32.5, 36.1)
SAIL	5.3 (4.3, 6.1)	58.4 (56.6, 60.3)	7.5 (6.3, 8.6)
SAIL-SL	6.6 (5.6, 7.6)	54.6 (52.7, 56.5)	9.1 (7.9, 10.3)
BB-A*	31.2 (28.8, 33.5)	52.0 (50.2, 53.9)	31.1 (29.2, 33.0)
Neural BF	43.7 (41.4, 46.1)	**61.5 (59.7, 63.3)**	44.4 (42.5, 46.2)
Neural A*	**63.0 (60.7, 65.2)**	55.8 (54.1, 57.5)	**54.2 (52.6, 55.8)**

CSM Dataset

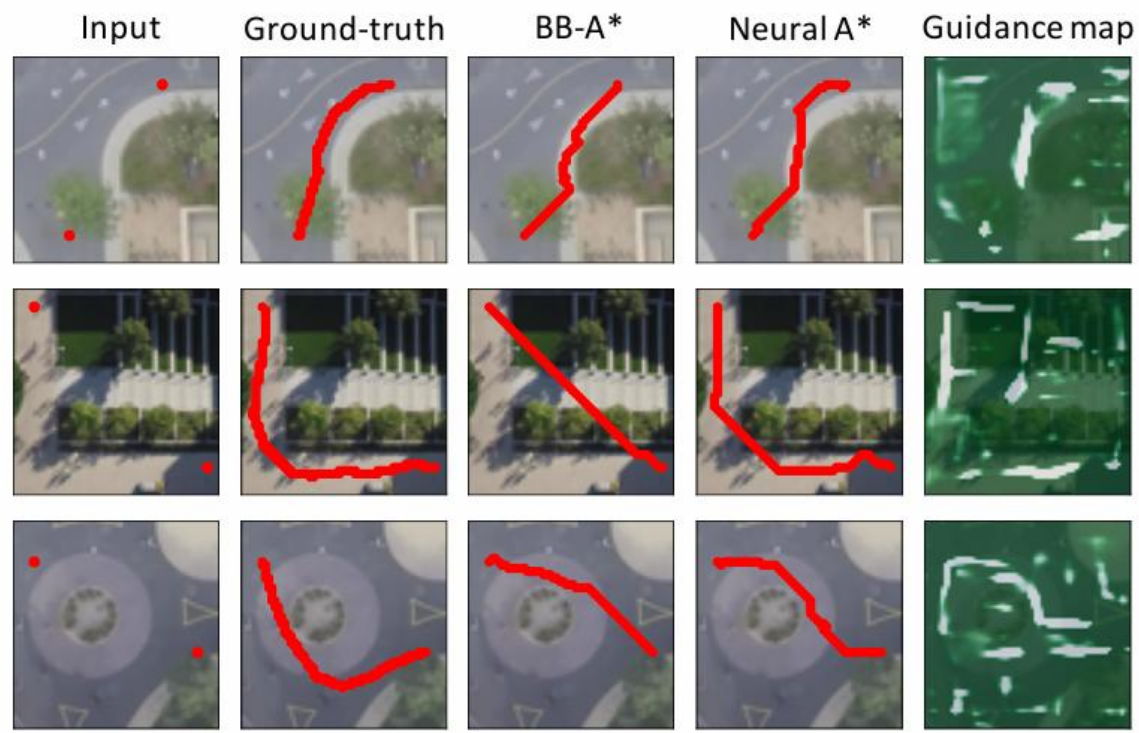
	Opt	Exp	Hmean
BF	54.4 (51.8, 57.0)	39.9 (37.6, 42.2)	35.7 (33.9, 37.6)
WA*	55.7 (53.1, 58.3)	37.1 (34.8, 39.3)	34.4 (32.6, 36.3)
SAIL	20.6 (18.6, 22.6)	41.0 (38.8, 43.3)	18.3 (16.7, 19.9)
SAIL-SL	21.4 (19.4, 23.3)	39.3 (37.1, 41.6)	17.6 (16.1, 19.1)
BB-A*	54.4 (51.8, 57.1)	40.0 (37.7, 42.3)	35.6 (33.8, 37.4)
Neural BF	60.9 (58.5, 63.3)	**42.1 (39.8, 44.3)**	40.6 (38.7, 42.6)
Neural A*	**73.5 (71.5, 75.5)**	37.6 (35.5, 39.7)	**43.6 (41.7, 45.5)**

Results - Point to Point Shortest Path Planning



Results - Raw Image Inputs

	Intra-scenes	Inter-scenes
BB-A*	152.2 (144.9, 159.1)	134.3 (132.1, 136.4)
Neural A*	**16.1 (13.2, 18.8)**	**37.4 (35.8, 39.0)**



Limitations

- Assumption of unit node costs in input
- Does not work in higher dimensional or continuous state spaces
- No optimality guarantee w.r.t costs in input
- Large memory required to store all A* operations

Q&A