Learning and/to Search:

Boosting A* via the learnt cues

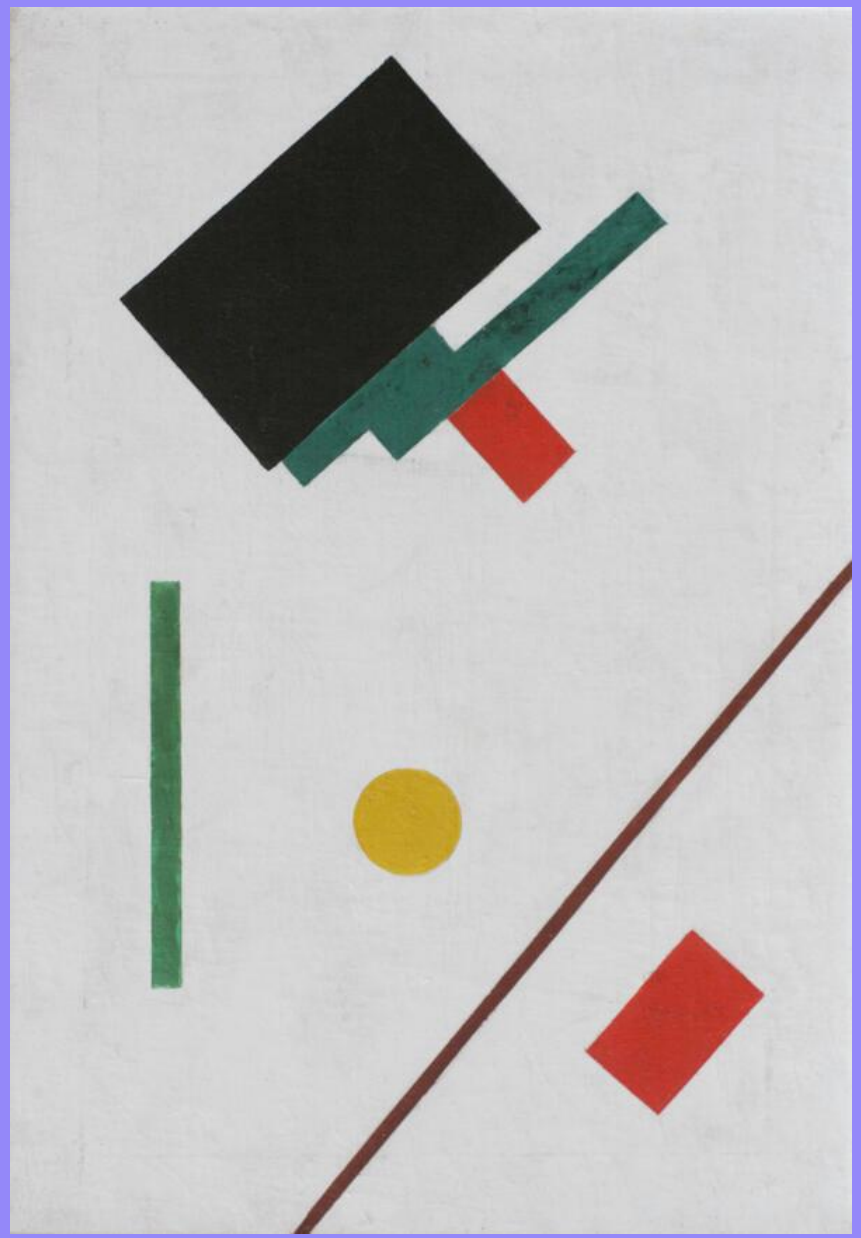
Konstantin Yakovlev

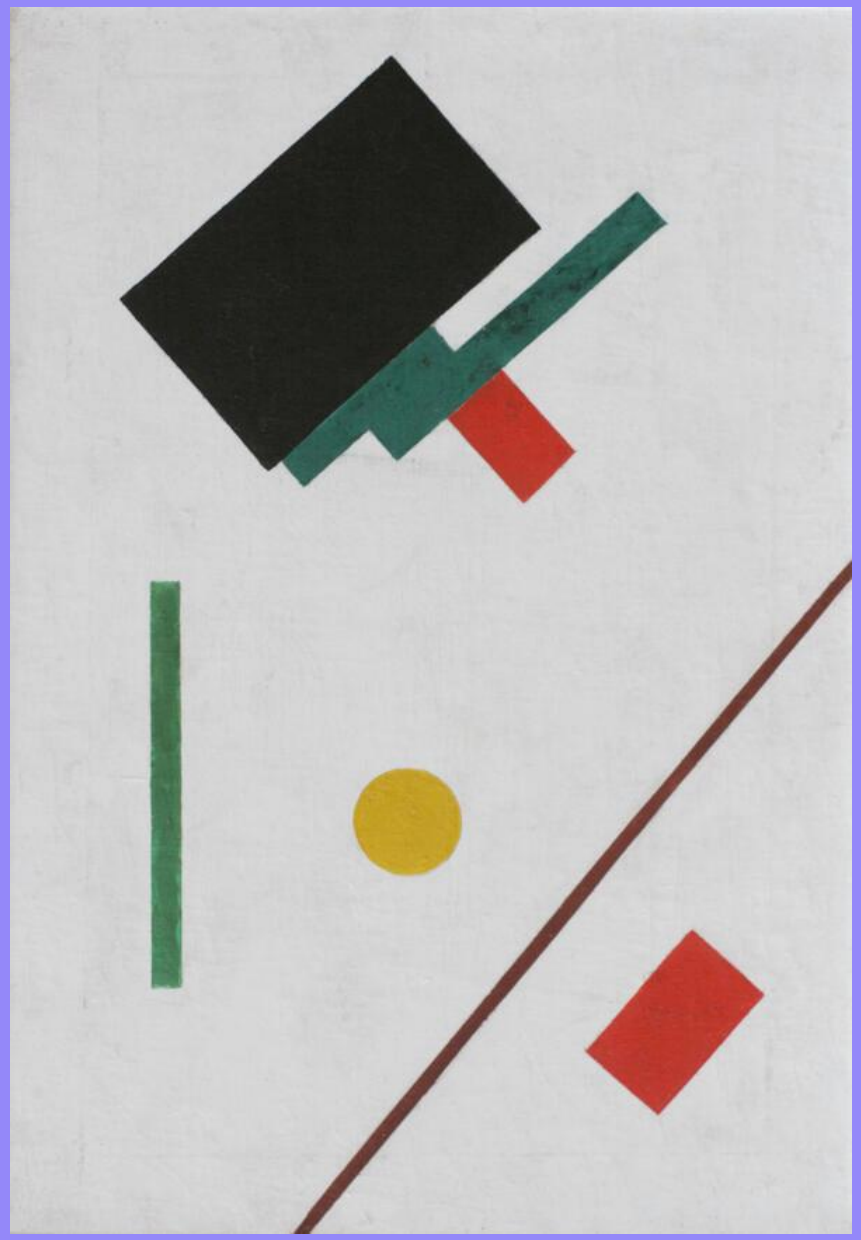
AIRI, RAS, HSE, MIPT

01 Figure

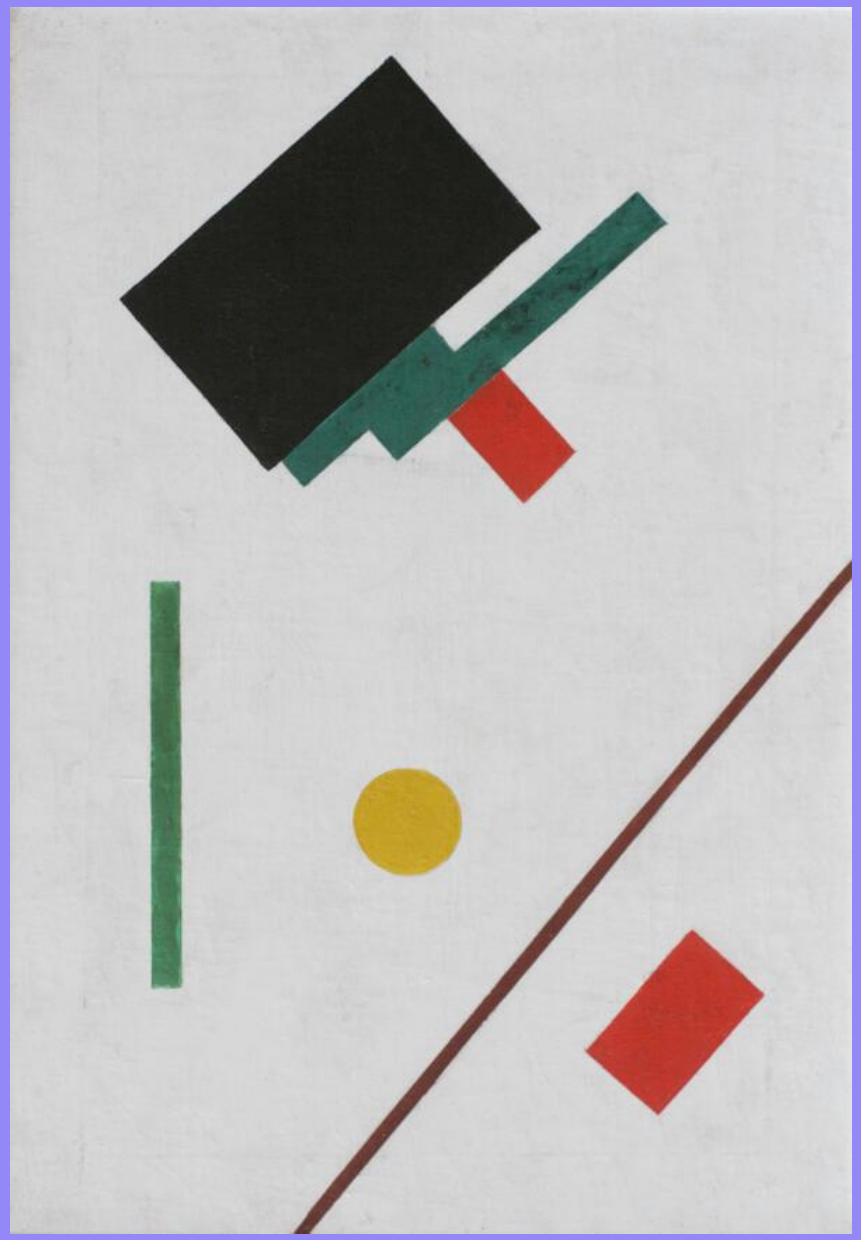
02 Main Stuff

03 Figure





Heuristic Search and ML Divided By The Thin Line And Anticipating The Warm Meeting



Suprematist Composition

1915

Kazemir Malevich

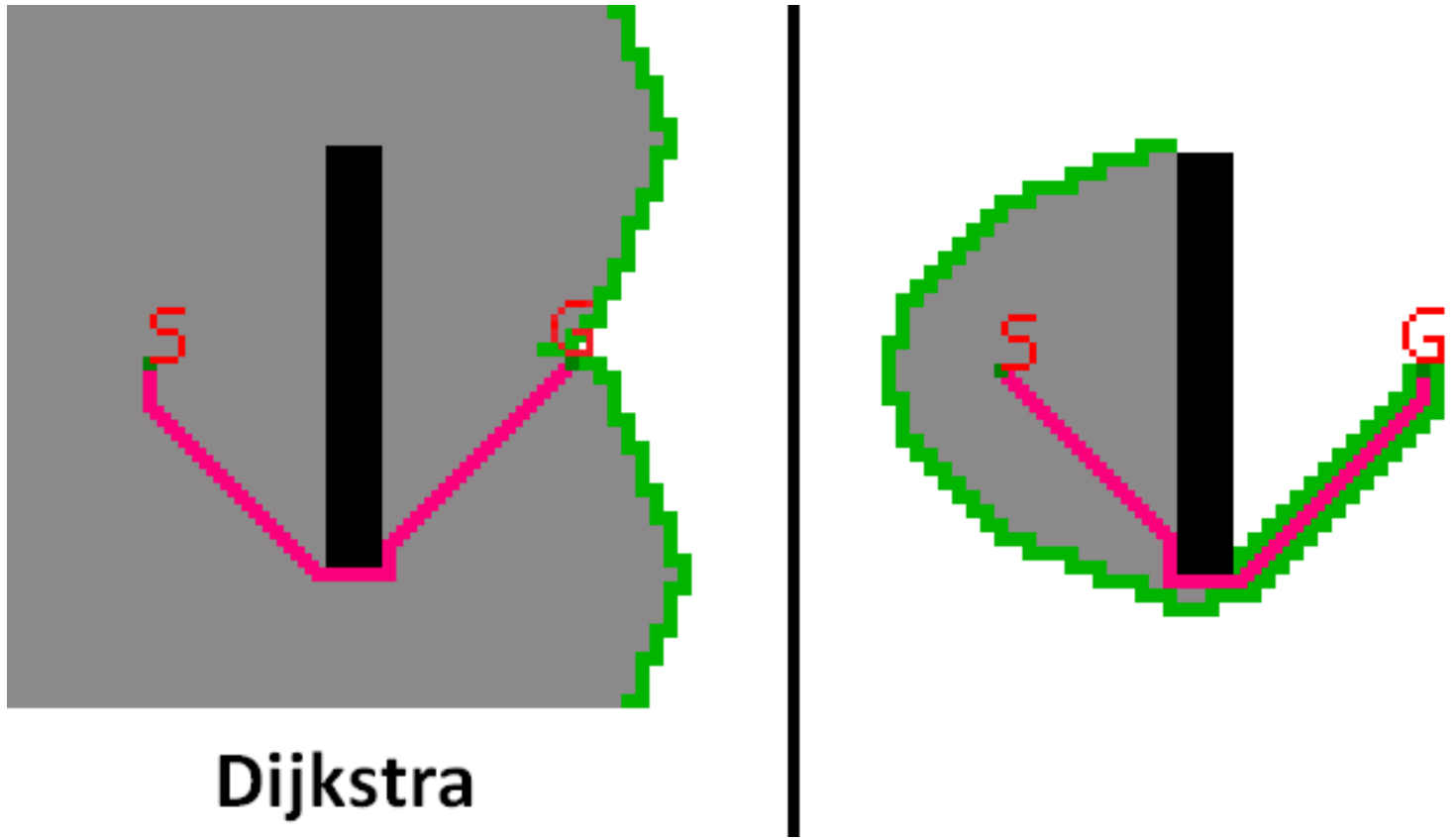
https://ar.culture.ru/ru/subject/malevich-k-s-suprematicheskaya-kompoziciya

Heuristic Search – What Is This

Heuristic Search, What Is This?

Version 1

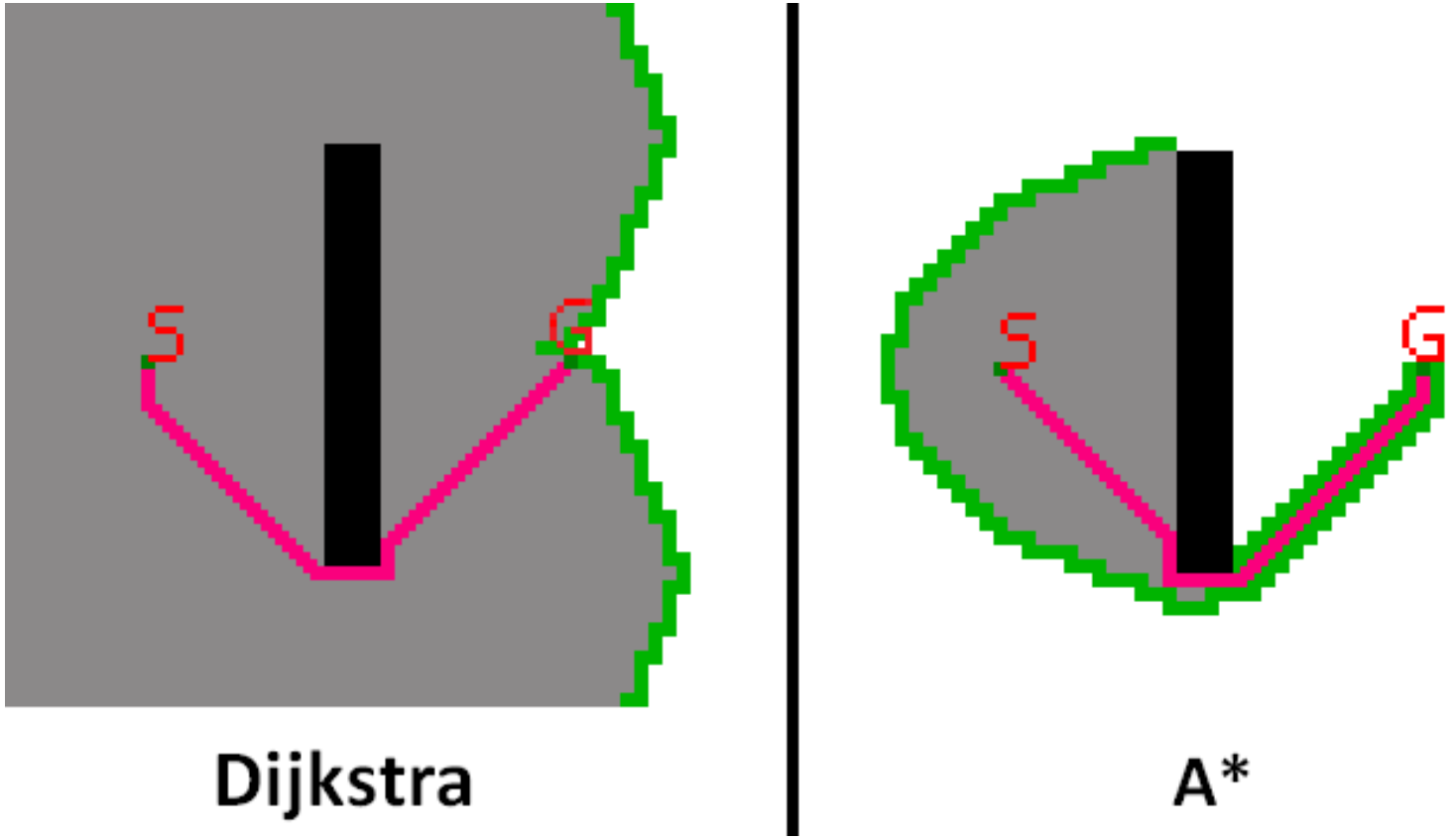
Improved Dijkstra



Heuristic Search, What Is This?

Version 1

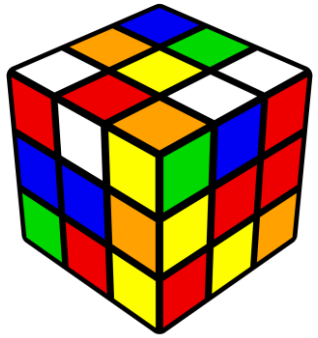
Improved Dijkstra

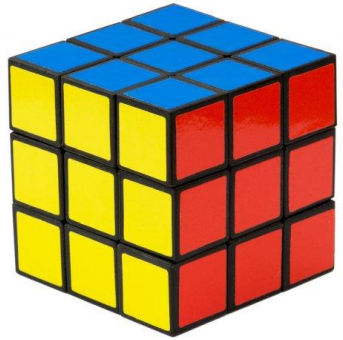


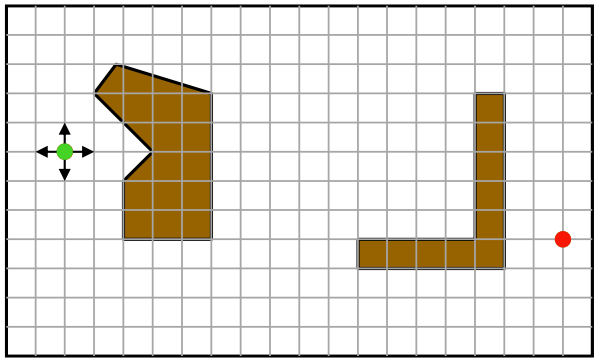
Heuristic Search, What Is This?

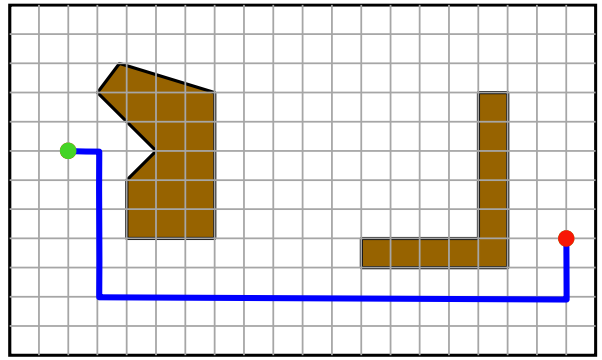
Version 2

A universal way to solve the well-posed tasks possessing combinatorial structure





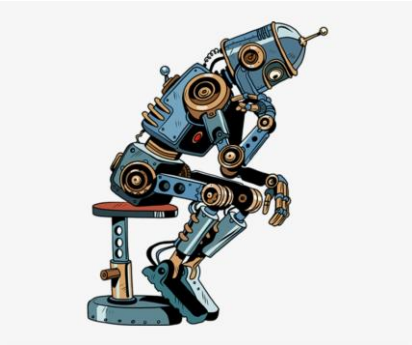




Heuristic Search, What Is This?

Version 3

The backbone for automated planning



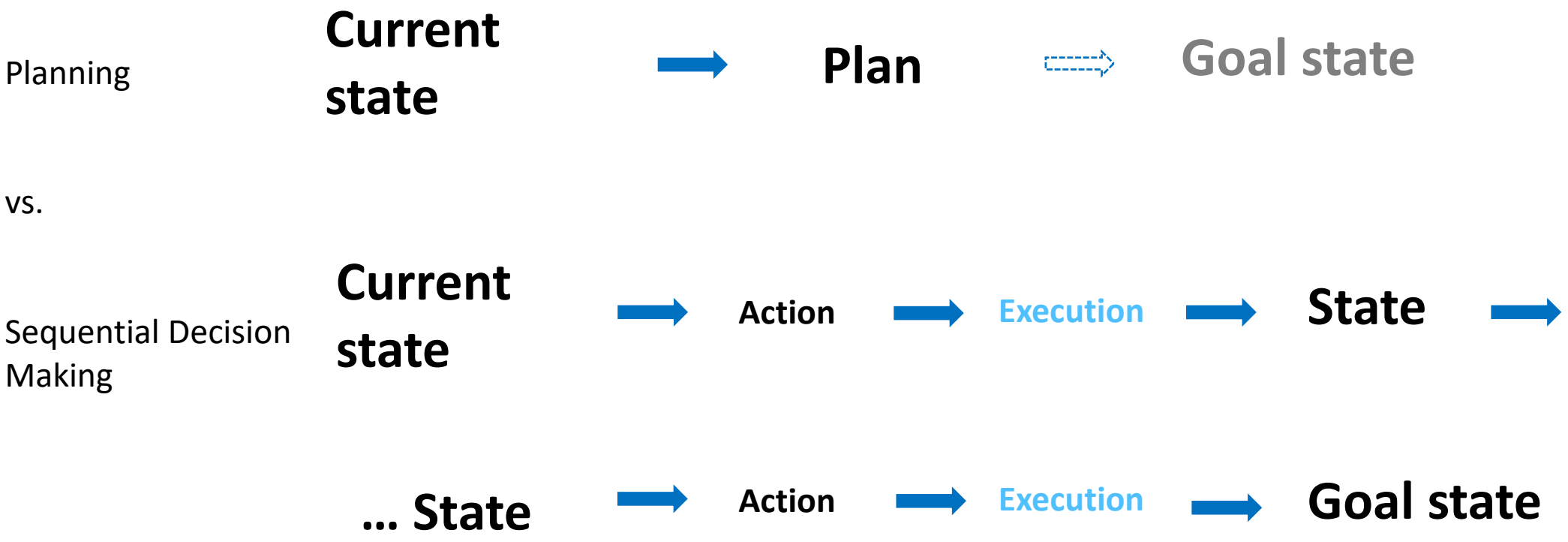
Robot at location (3, 7)

Robot at location (12, 2)

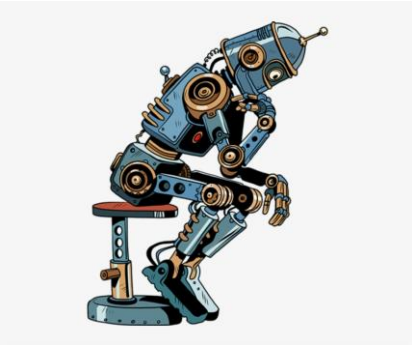
Current state → **<u>Sequence of actions</u> (Plan)** → **Goal state**

AIRI × Университет Сириус

Planning vs. Sequential Decision Making



Planning



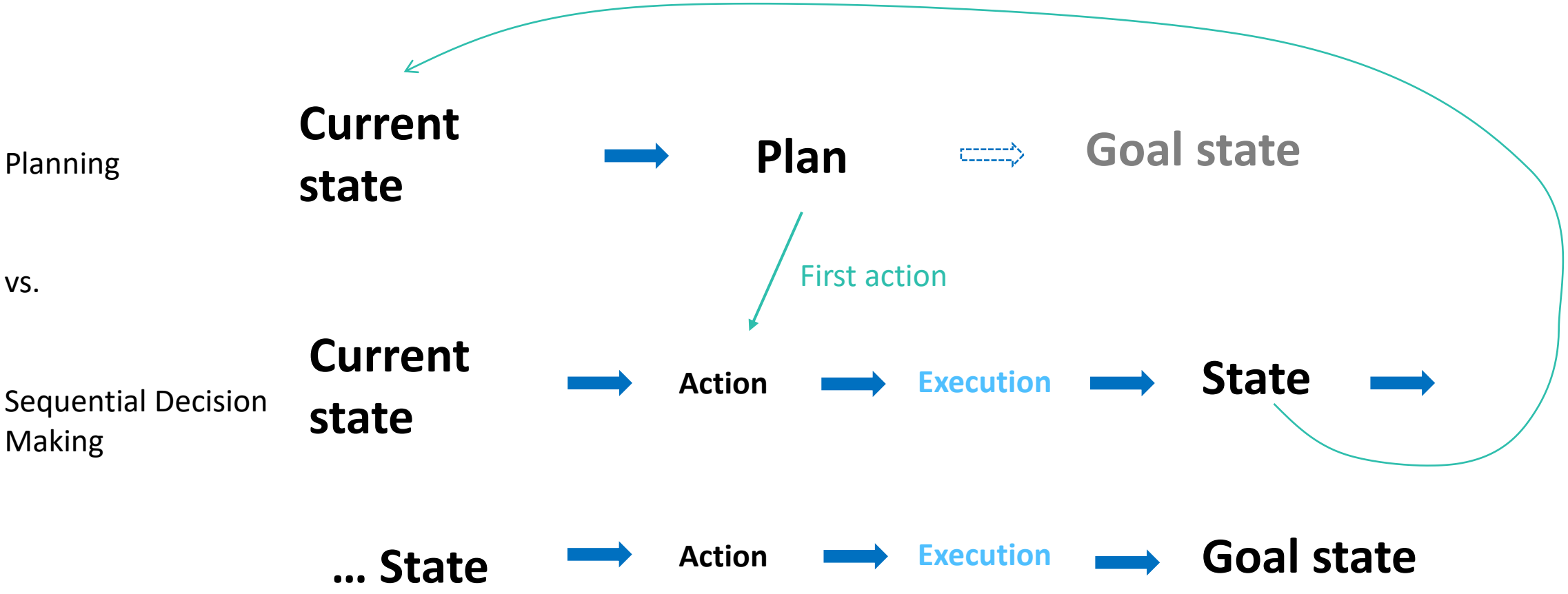
- **Plan**
- **Not actually reaching the goal state**

Robot at location (3, 7)

Robot at location (12, 2)

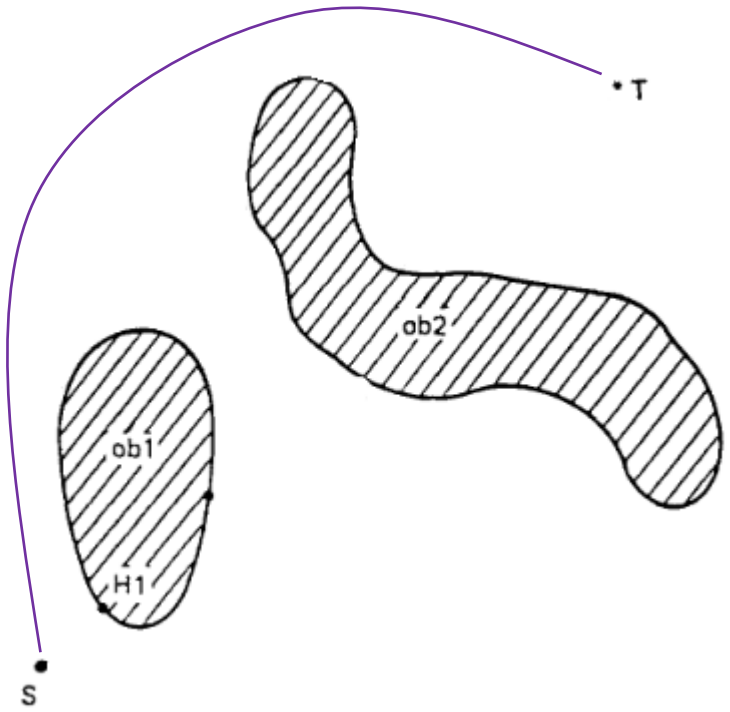
AIRI × Университет Сириус

Planning vs. Sequential Decision Making



Planning vs. Sequential (Reactive) Decision Making

Planning



BUG algorithm (Lumelsky & Stepanov, 1987)

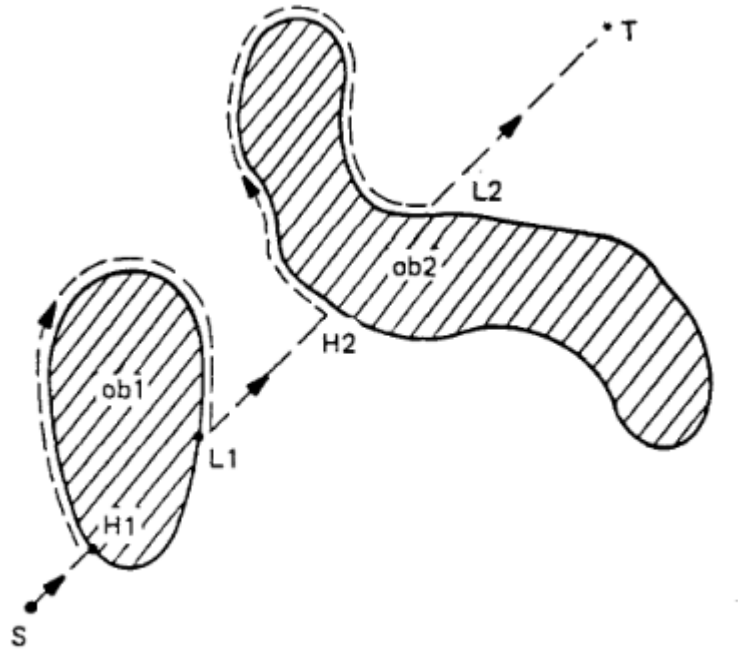
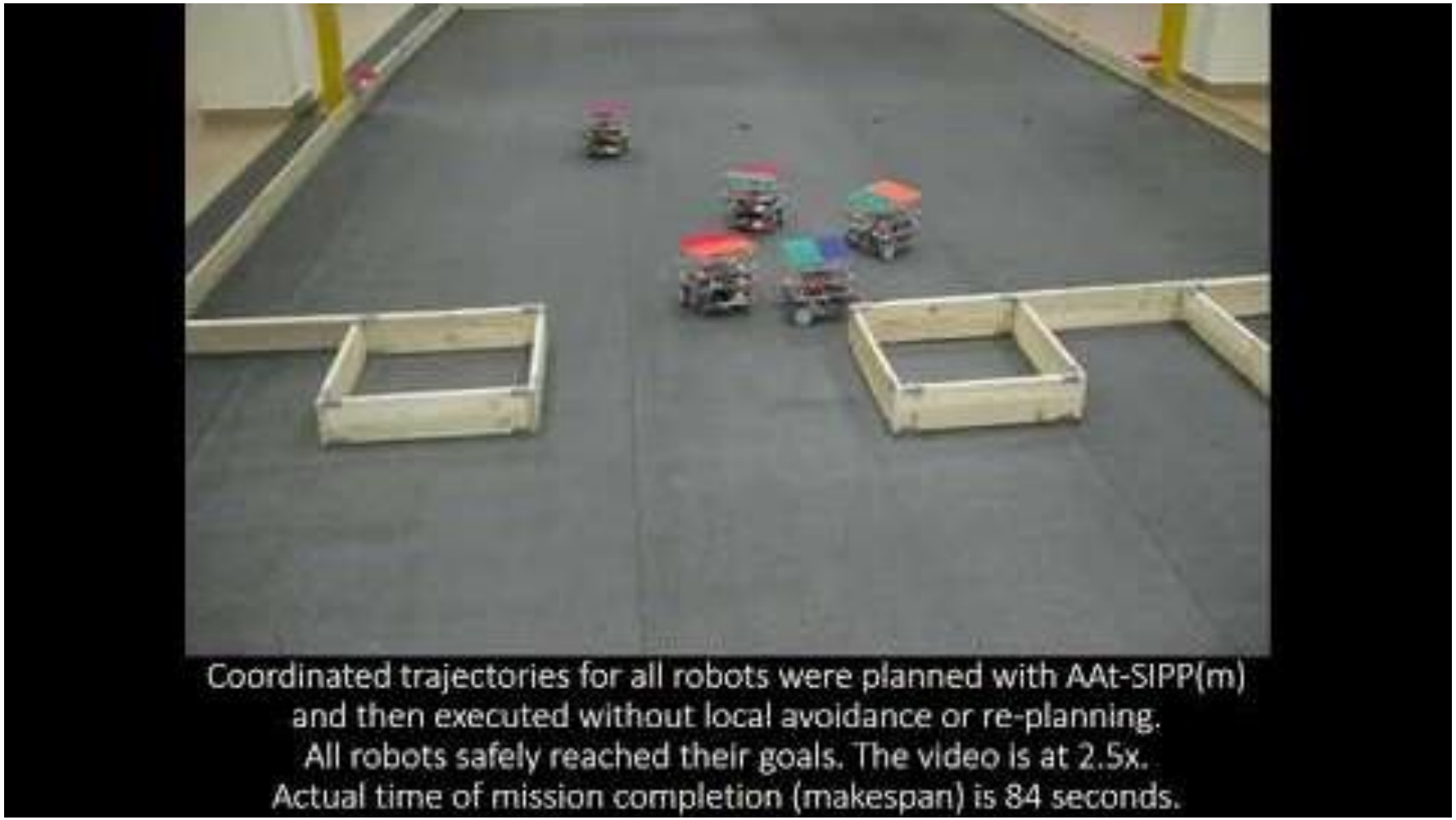


Fig. 4. Automaton's path (dotted line) under Algorithm Bug2.

Lumelsky, V. J., & Stepanov, A. A. (1987). Path-planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape. *Algorithmica*, *2*(1-4), 403-430.

Planning vs. Sequential (Reactive) Decision Making



https://www.youtube.com/watch?v=1Jrye5S0ZV8

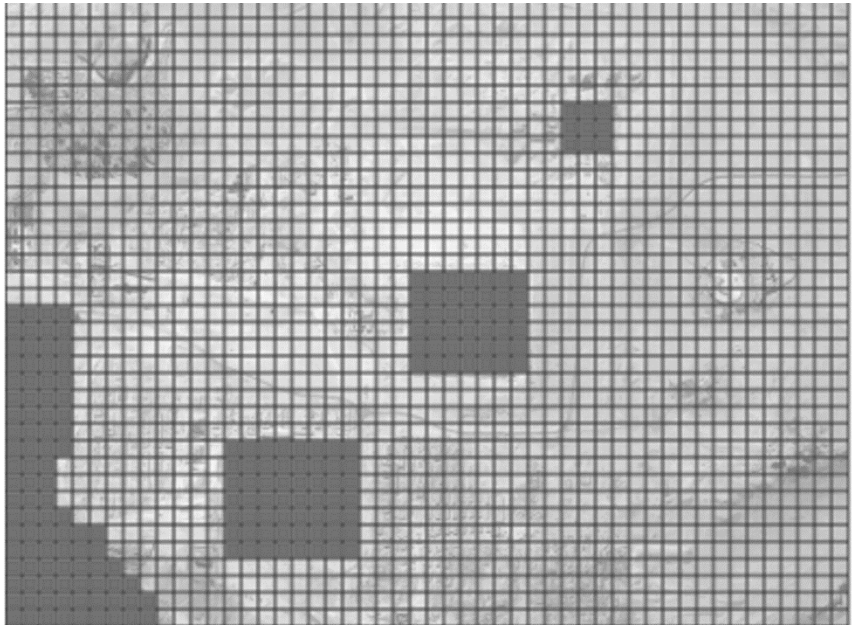
02

(Heuristic) Search for Path Finding :: Details

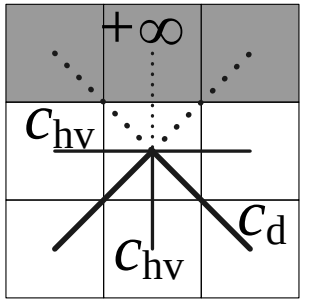
Discretize It

(Continuous) Environment

Grid

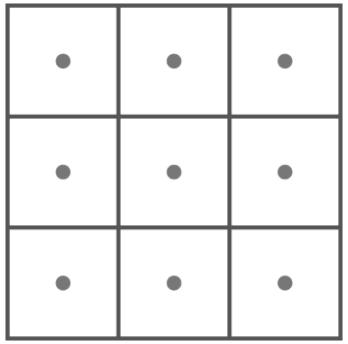


Cost



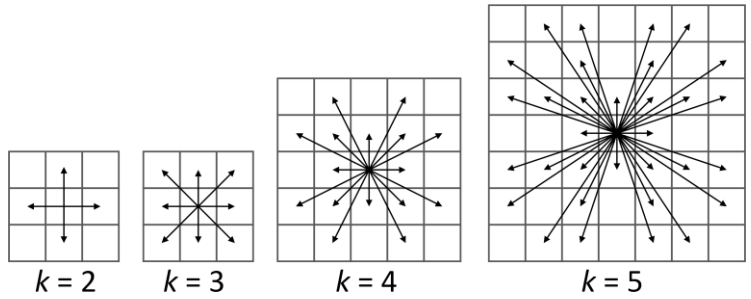
$c_{hv}=1$ $c_d=\sqrt{2}$

States (Vertices) – **position of the robot**
centers of un-blocked grid cells



Actions (Edges) – **move to another position**
2^k moves

$2^2 = 4$ moves {N, E, S, W}
$2^3 = 8$ moves {N, NE, E, SE, S, SW, W, NW}
….



Action/Edge Cost – **Euclidean distance**

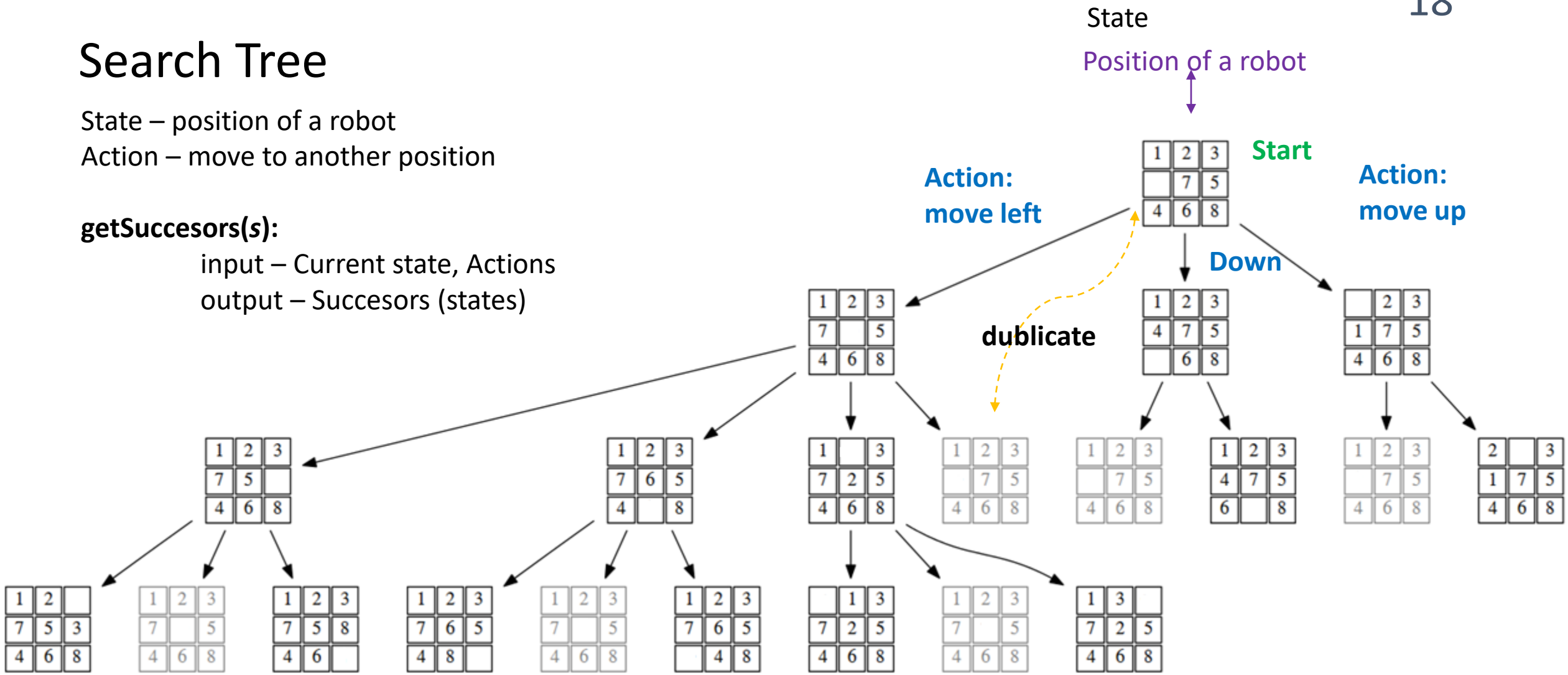
AIRI × Университет Сириус

Search Tree

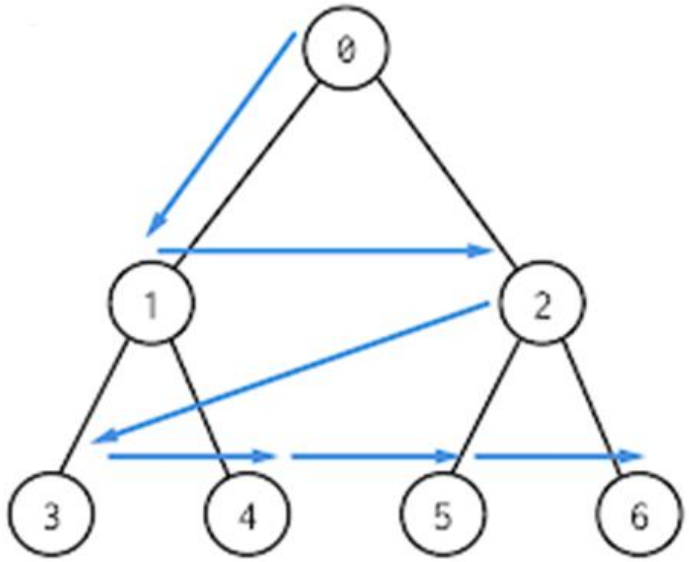
State – position of a robot
Action – move to another position

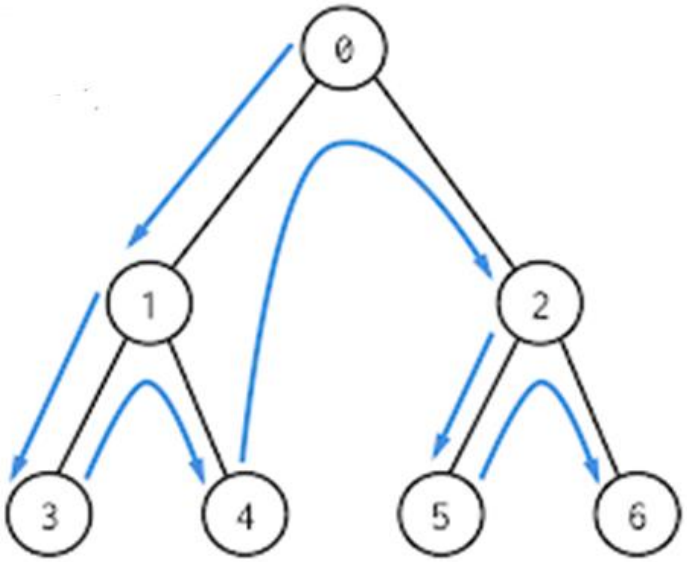
getSuccesors(*s*):

input – Current state, Actions
output – Succesors (states)



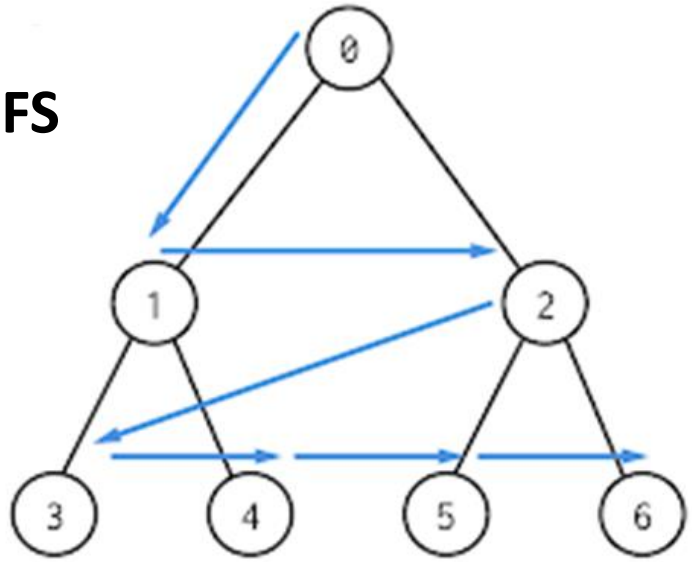
Searching The Tree



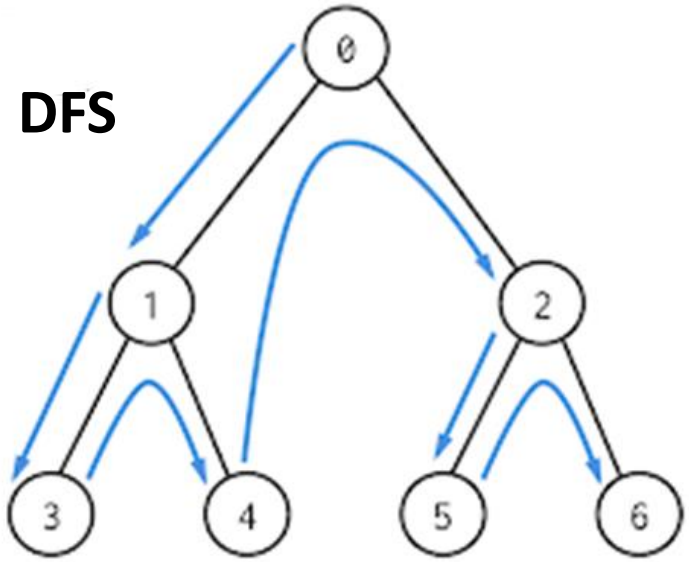


Searching The Tree

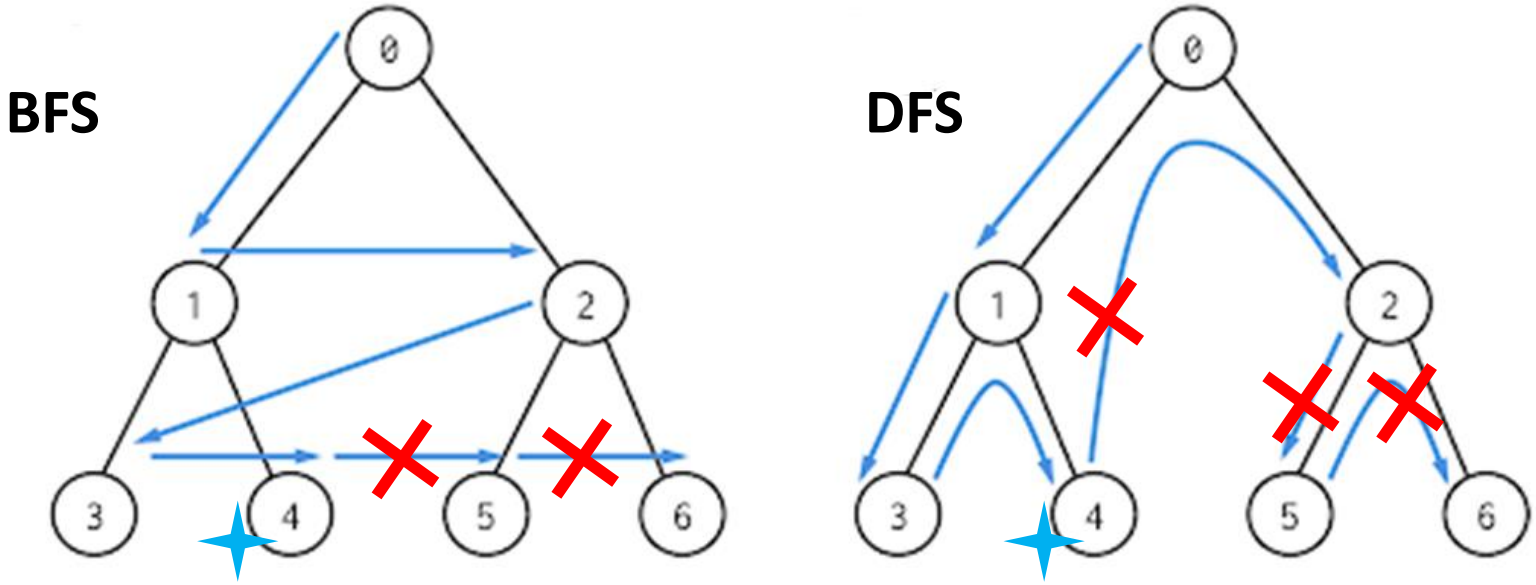
BFS



DFS



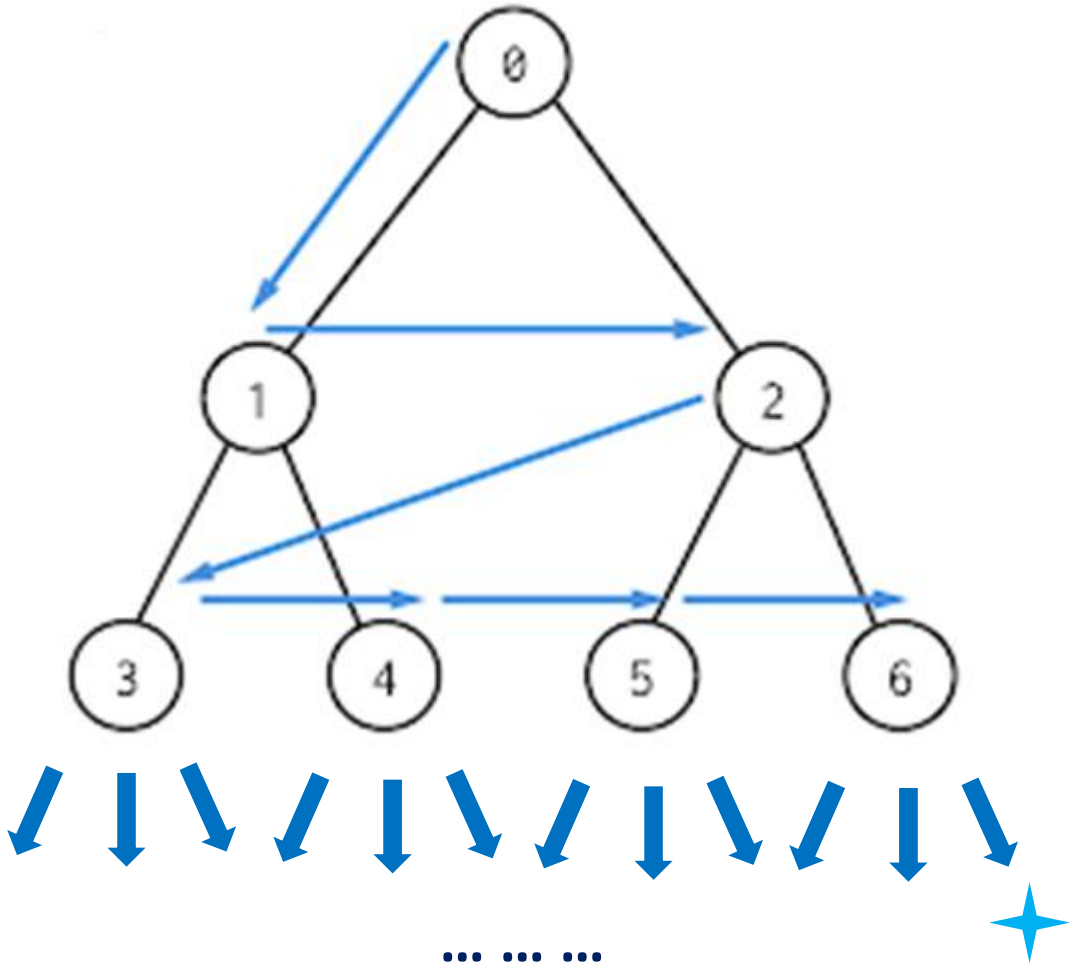
Searching The Tree



1. Tree is generated while the search
2. Not traversal, but search

   Stop criteria – goal state (vertex) is encountered
3. Special care for duplicates detection

BFS

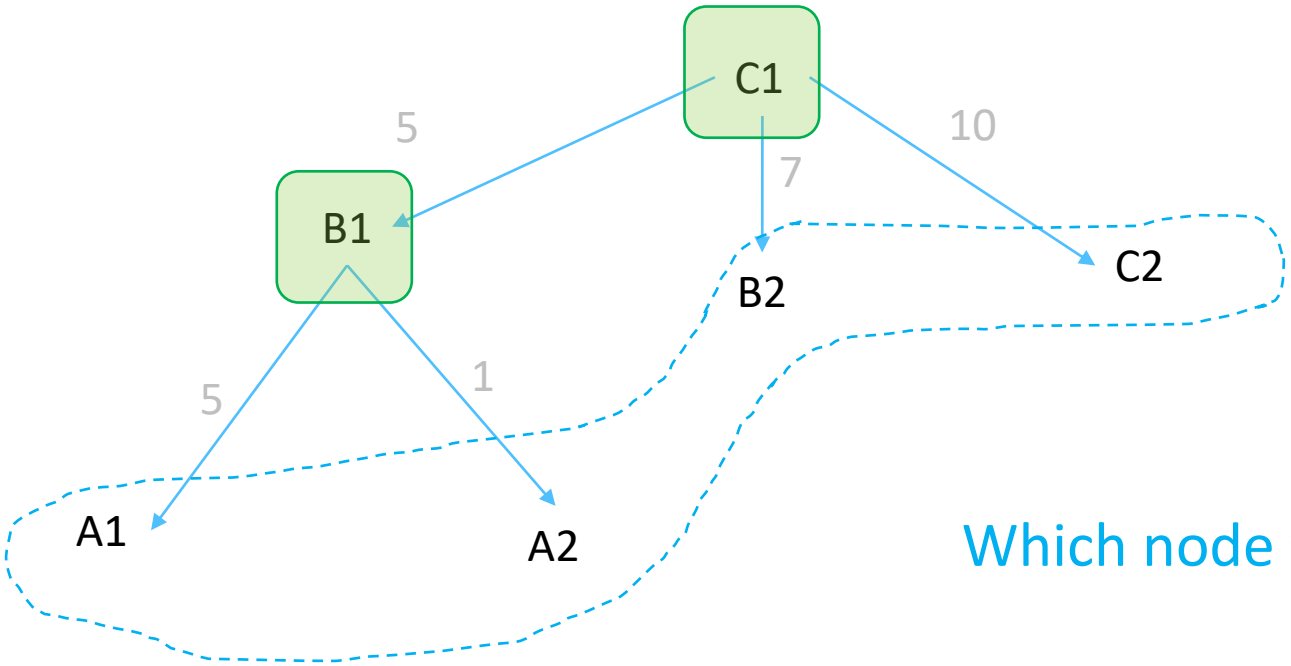


IDEA = explore ‘layer by layer’
to guarantee **optimality**

Queue = {start}, Visited = {}

While …
  cur_state = Queue.Pop (FIFO)
  Visited.Add(cur_state)
  Generate successors (and prune duplicates)
  Add successors to Queue
…

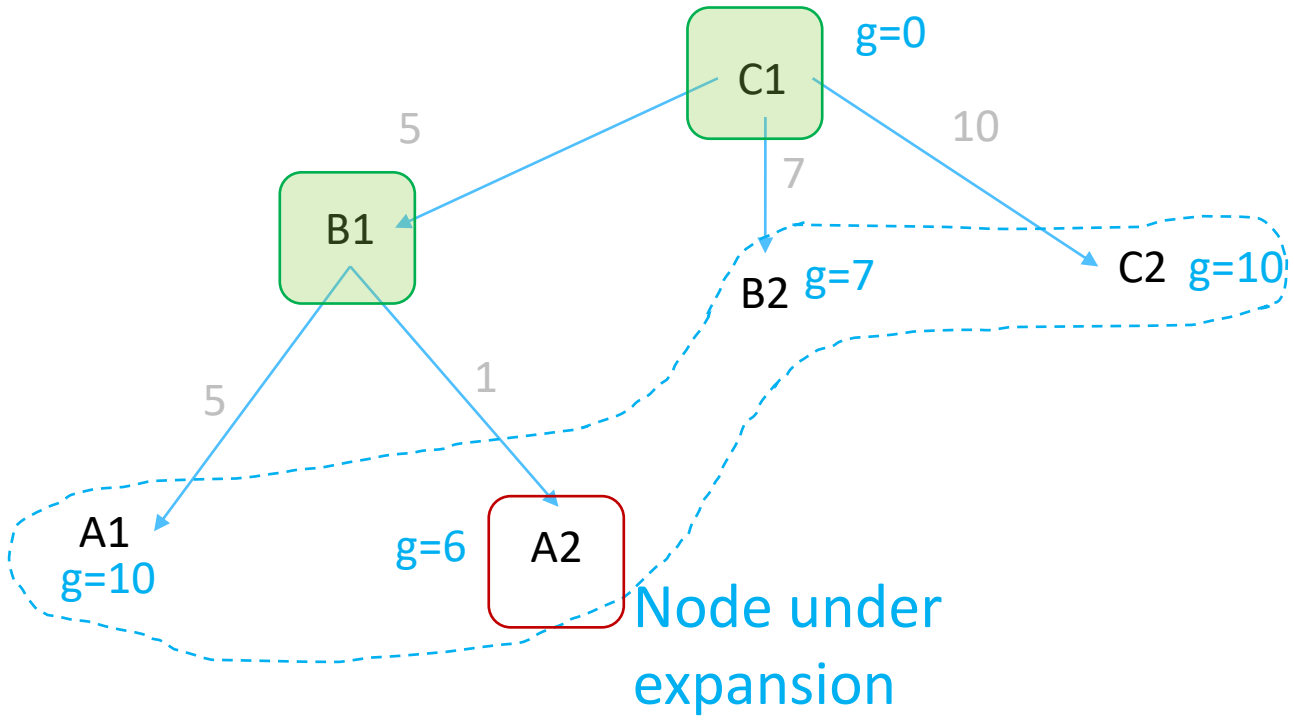
DIJKSTRA (~=BFS with Costs)



Which node will be expanded next???

DIJKSTRA

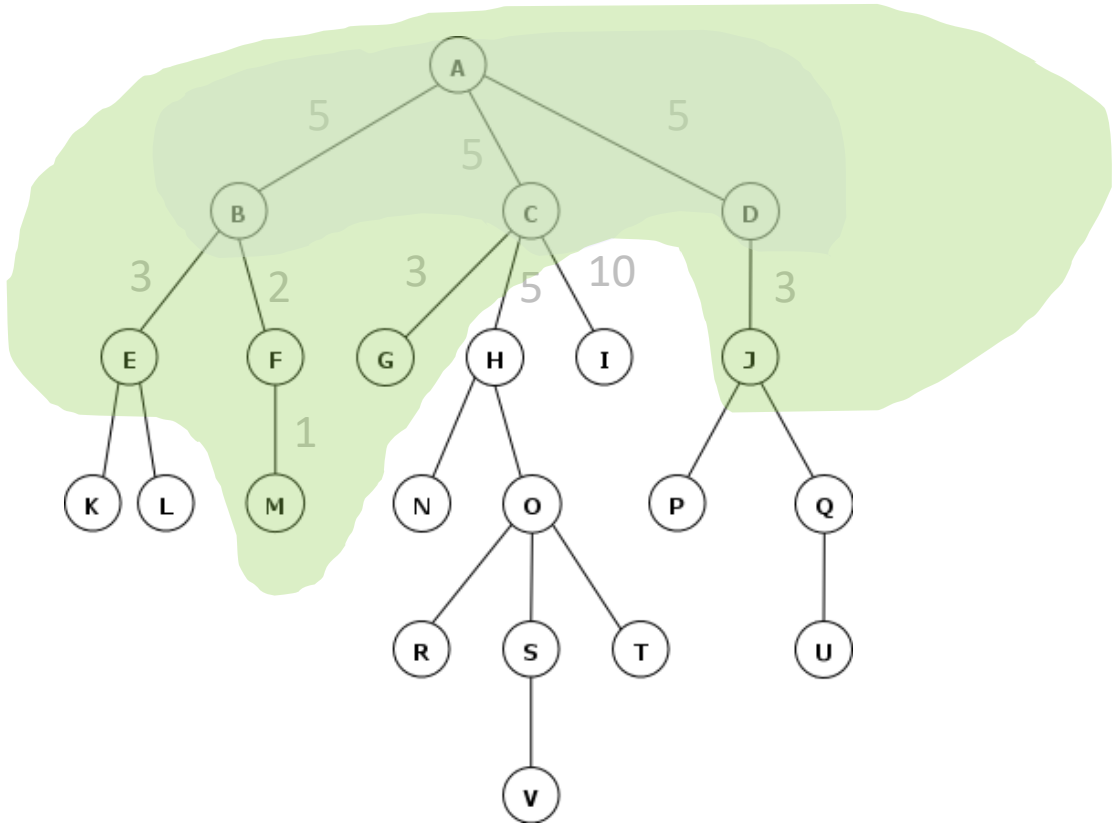
IDEA = explore '**cost** layer by **cost** layer' to guarantee **optimality**



OPEN = {start}, **CLOSED** = {}

While …
  cur_state = state with **min. accumulated cost (g-value)** in OPEN
  OPEN.Remove(cur_state)
  CLOSED.Add(cur_state)
  Generate successors (and prune duplicates)
  Add successors to OPEN
…

DIJKSTRA



explore '**cost** layer by **cost** layer'
to guarantee **optimality**

Cost level: 5
Cost level: 8
Cost level: 10
etc.

Dijkstra

Completeness

????

Optimality

????

Dijkstra

Complete

Dijkstra finds the solution if one exists and if the solution is not existent Dijkstra correctly terminates, returning *failure* output.

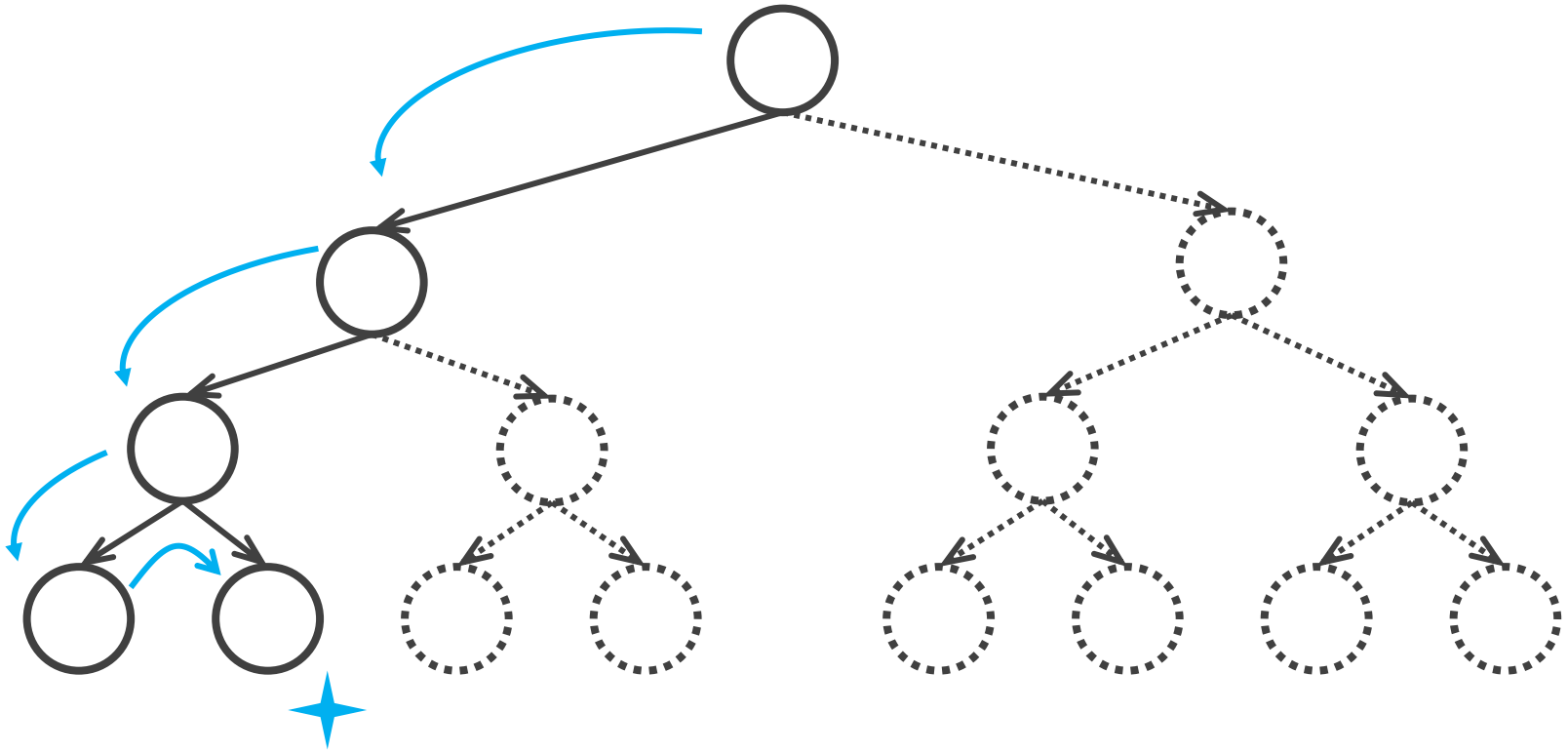
Optimal

The cost of the solution returned by Dijkstra is (theoretically proved) to be minimal.

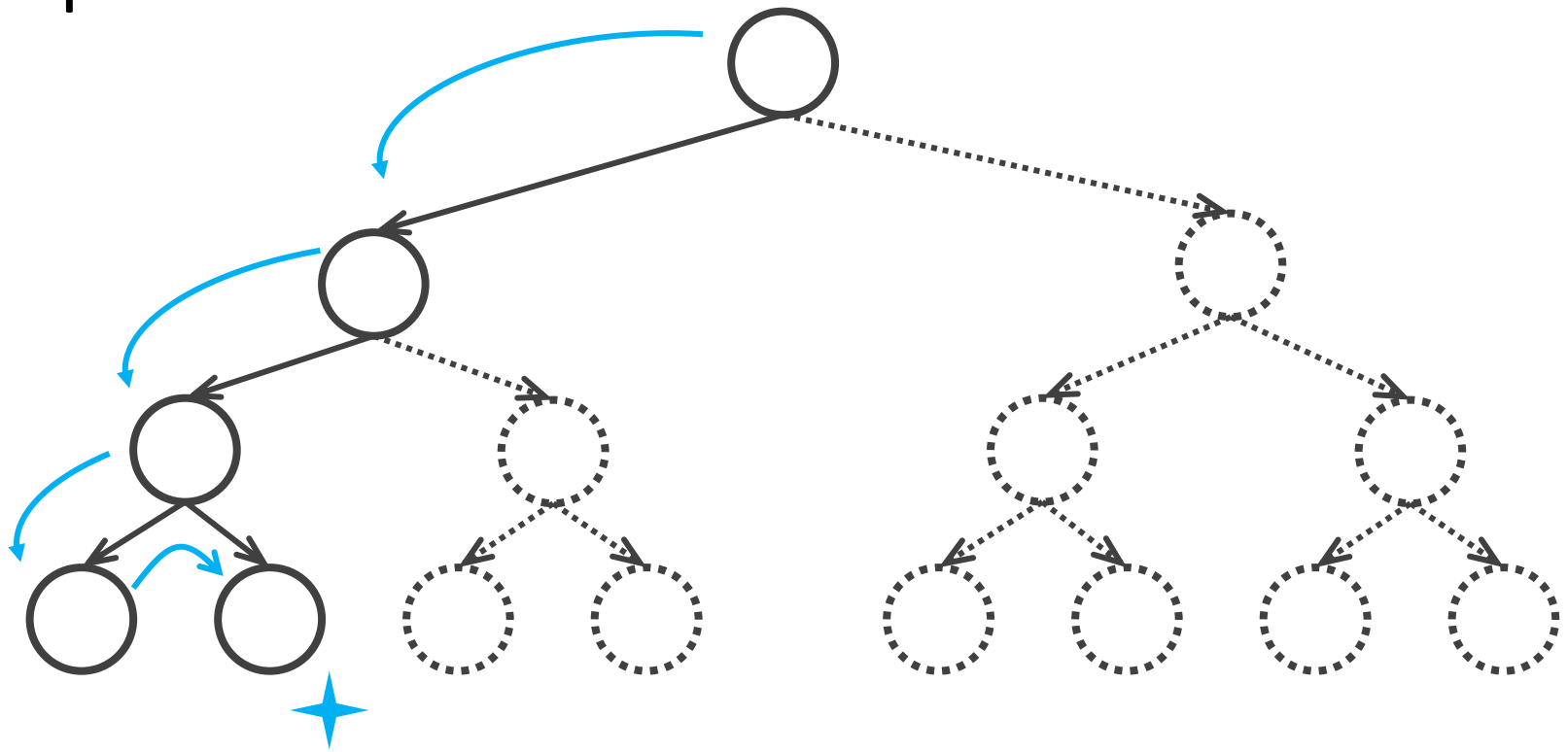
OPEN = {start}
CLOSED = {}

While **OPEN** is not empty
    cur_state = state with **min. g-value** in OPEN
    OPEN.Remove(cur_state)
    if cur_state = goal_state return PathIsFound
    CLOSED.Add(cur_state)
    Generate successors (and prune duplicates)
    Add successors to OPEN
return PathNoFound

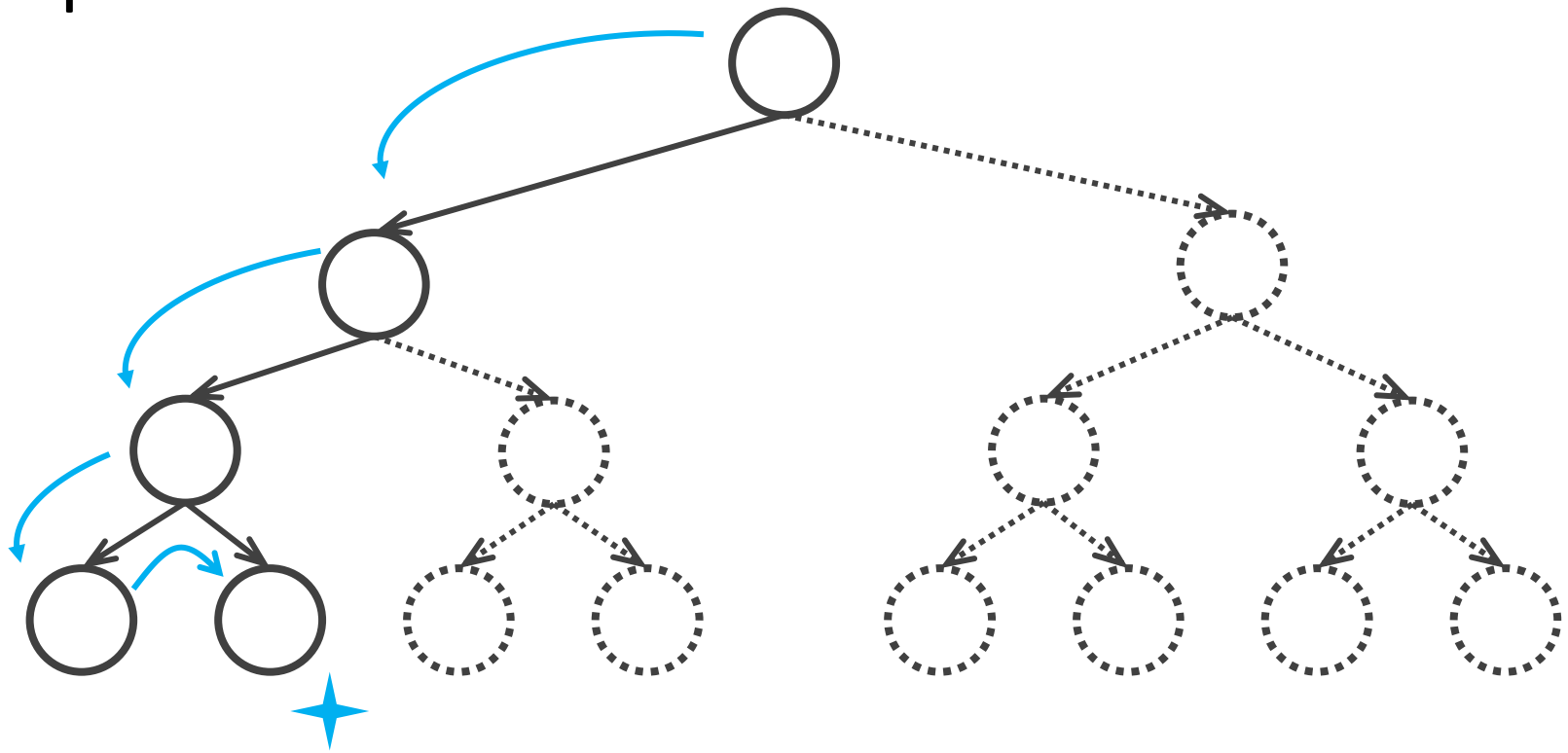
DFS



DFS – Completeness?

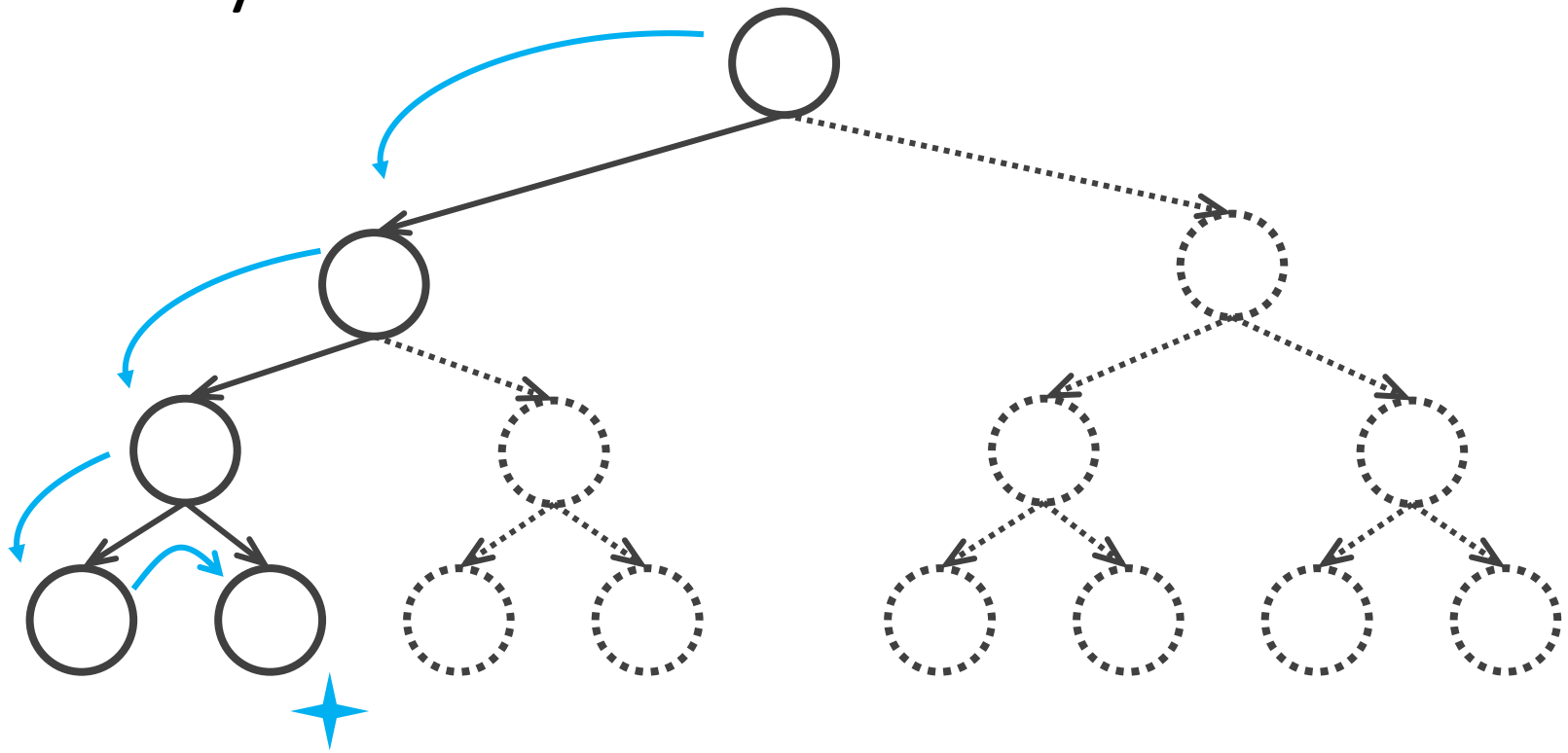


DFS – Completeness?

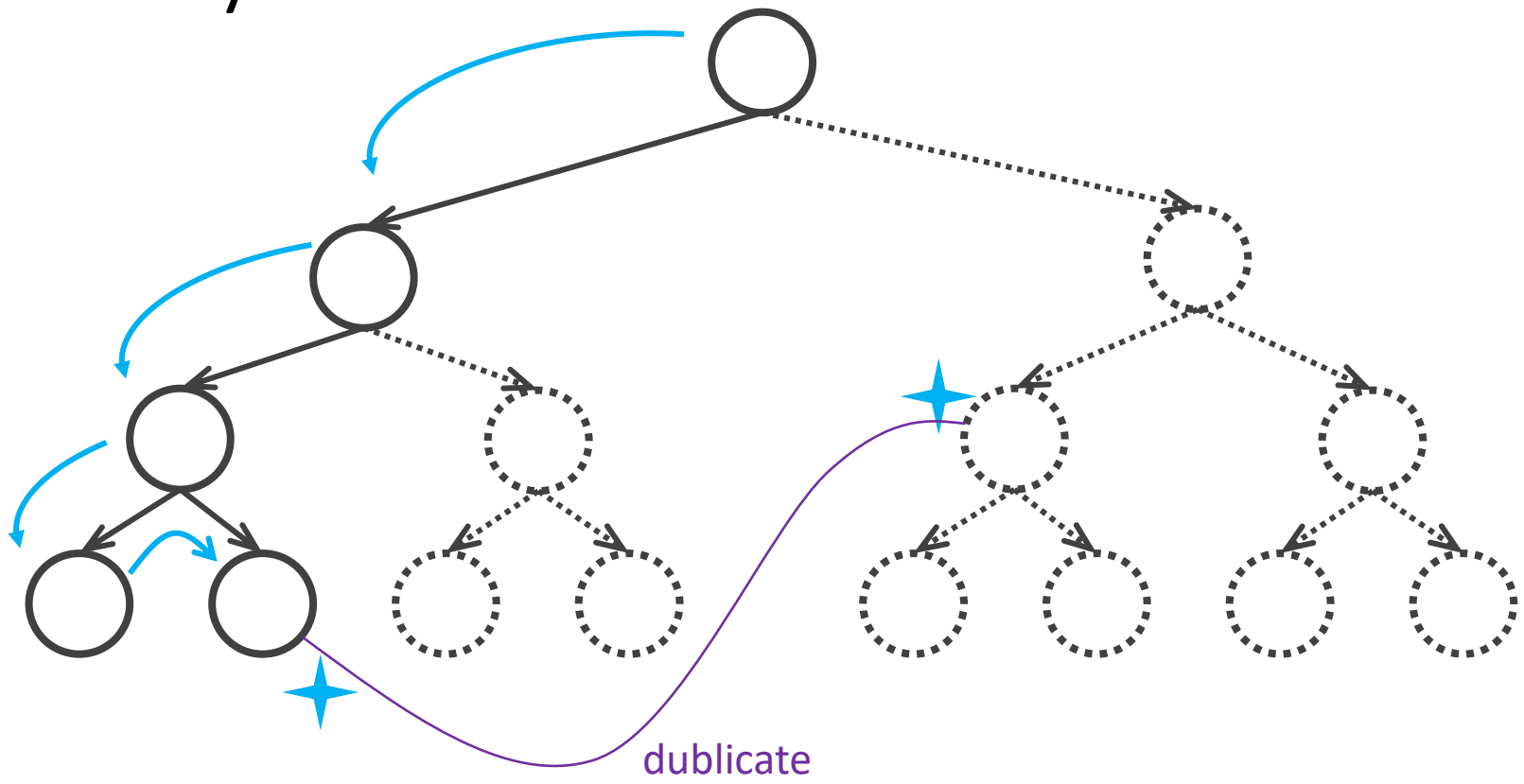


Complete

DFS – Optimality ?



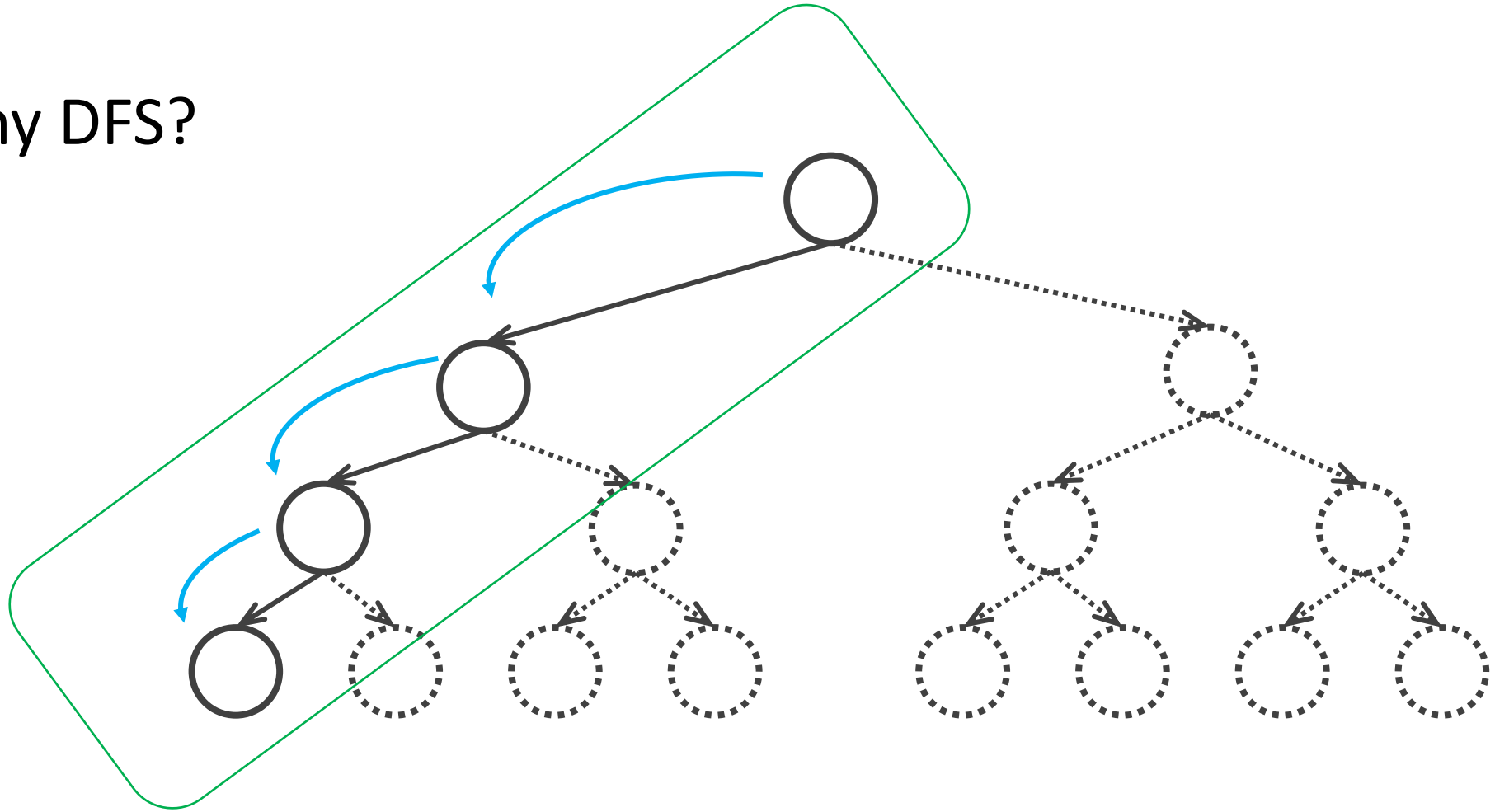
DFS – Optimality ?



Not optimal

WHY DFS?

Why DFS?

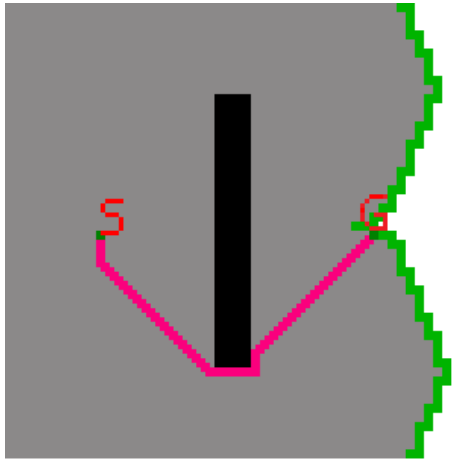


1. Quick descent

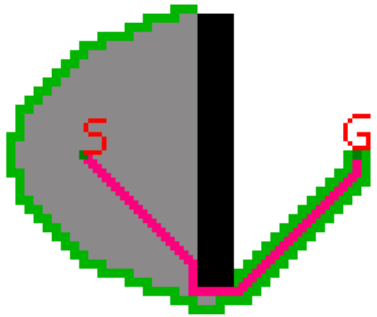
2. Much less nodes is stored

Heuristic Search, A* Search

1. Dijkstra with heuristics
2. BFS + DFS combo



Dijkstra



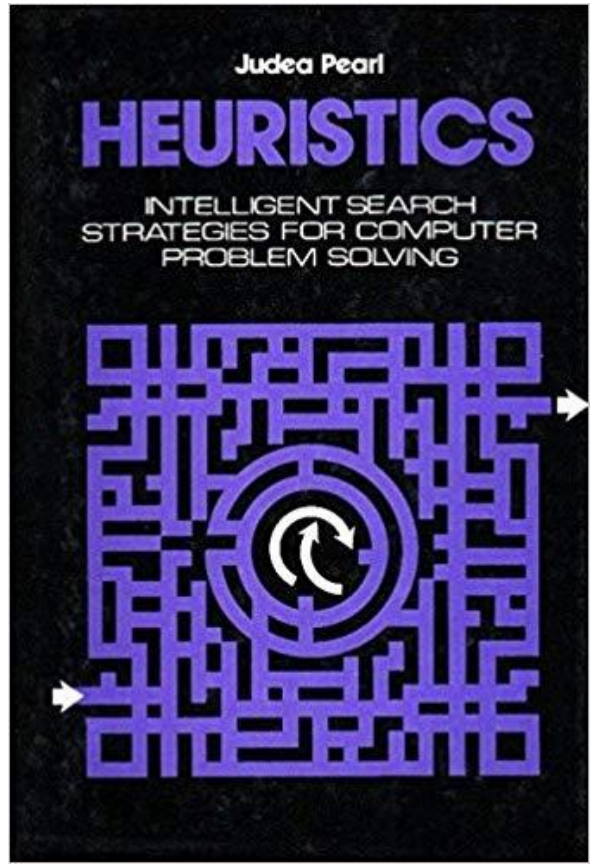
A*

No heuristic is used

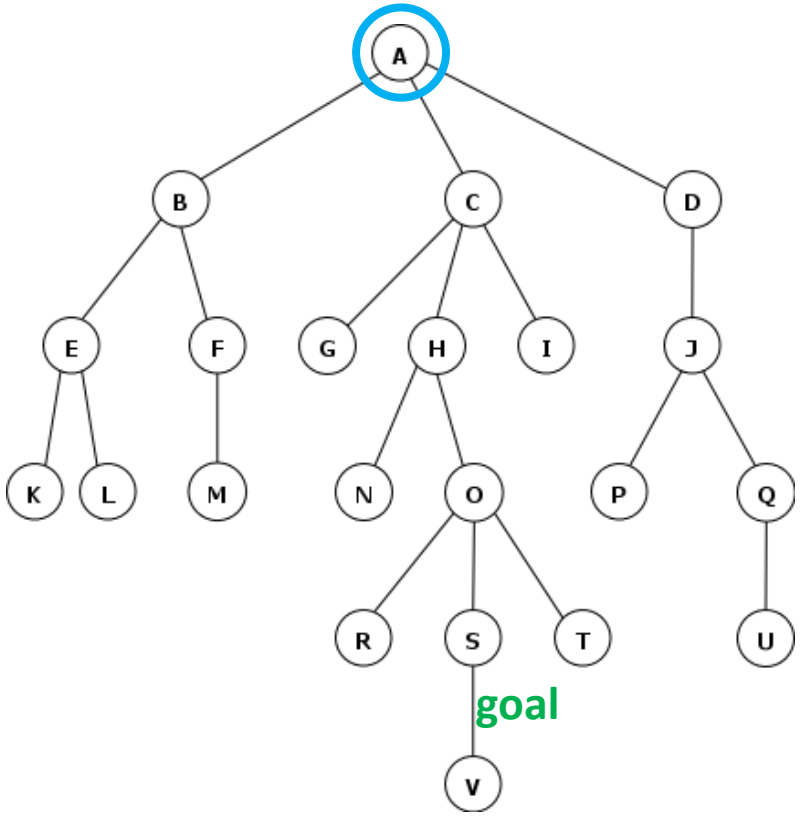
“Good” heuristic is used
=>
Less steps (faster)
Same result

Pearl, J. (1984). Heuristics: intelligent search strategies for computer problem solving.

Good heuristics



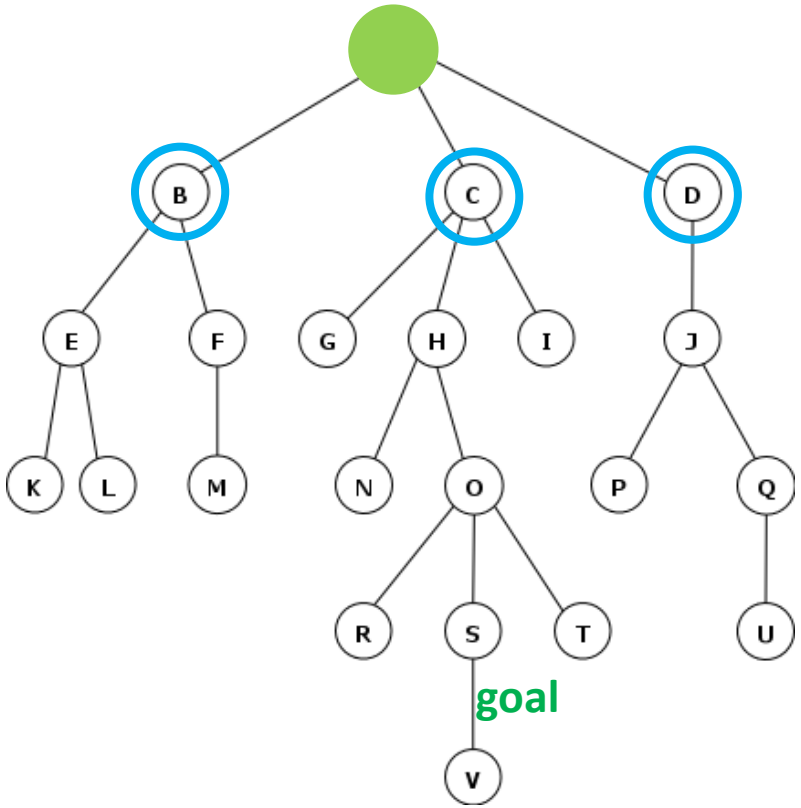
A* Search Tree



CLOSED part of the search tree
g-values = g*-values = true shortest paths (will never decrease any more)

OPEN part of the search tree
g-values may decrease (new paths of better costs might be found)

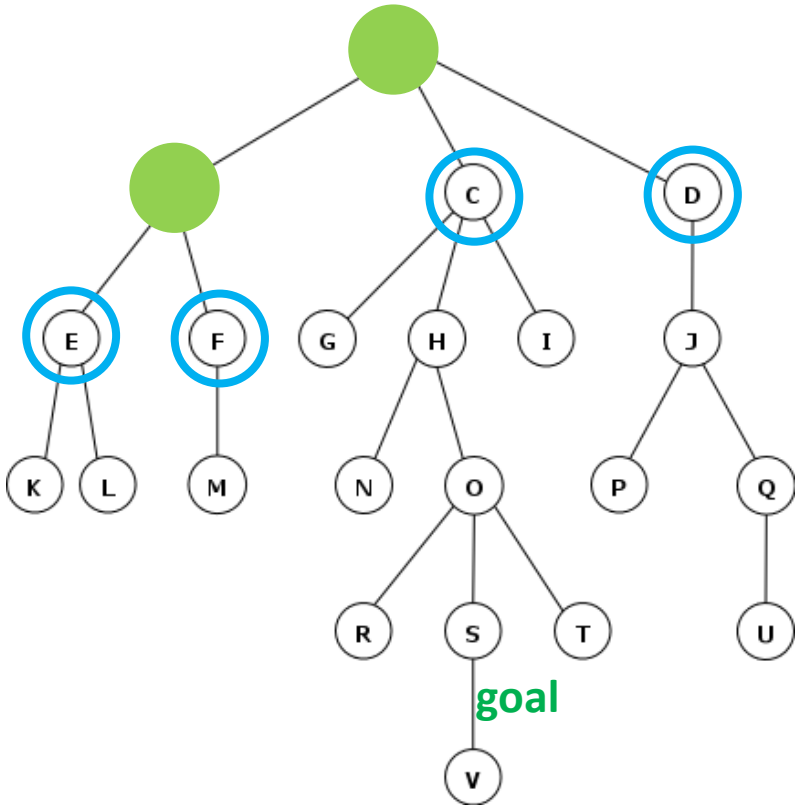
A* Search Tree



CLOSED part of the search tree
g-values = g*-values = true shortest paths (will never decrease any more)

OPEN part of the search tree
g-values may decrease (new paths of better costs might be found)

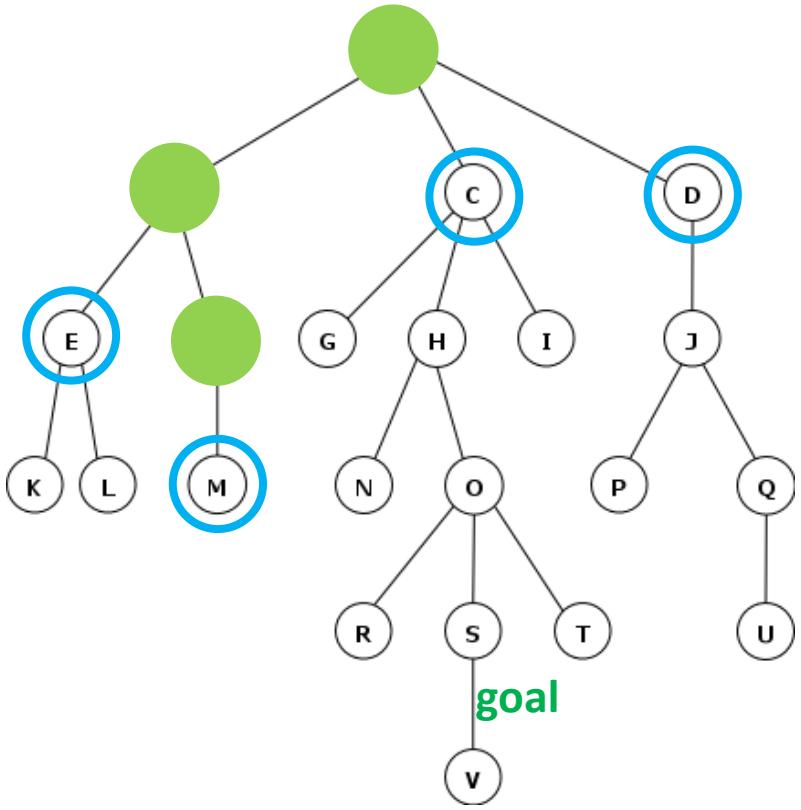
A* Search Tree



CLOSED part of the search tree
g-values = g*-values = true shortest paths (will never decrease any more)

OPEN part of the search tree
g-values may decrease (new paths of better costs might be found)

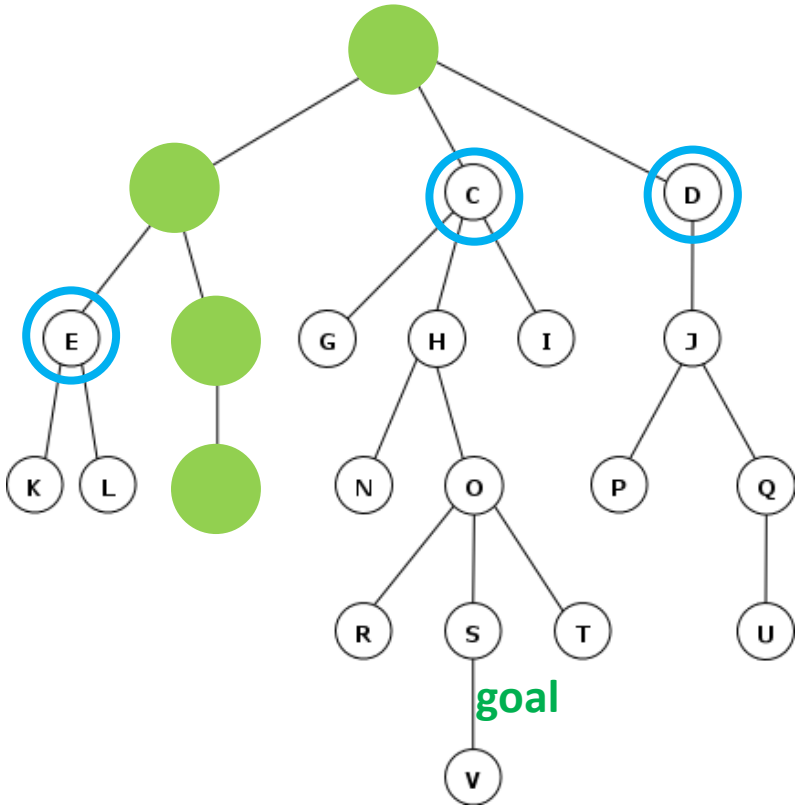
A* Search Tree



CLOSED part of the search tree
g-values = g*-values = true shortest paths (will never decrease any more)

OPEN part of the search tree
g-values may decrease (new paths of better costs might be found)

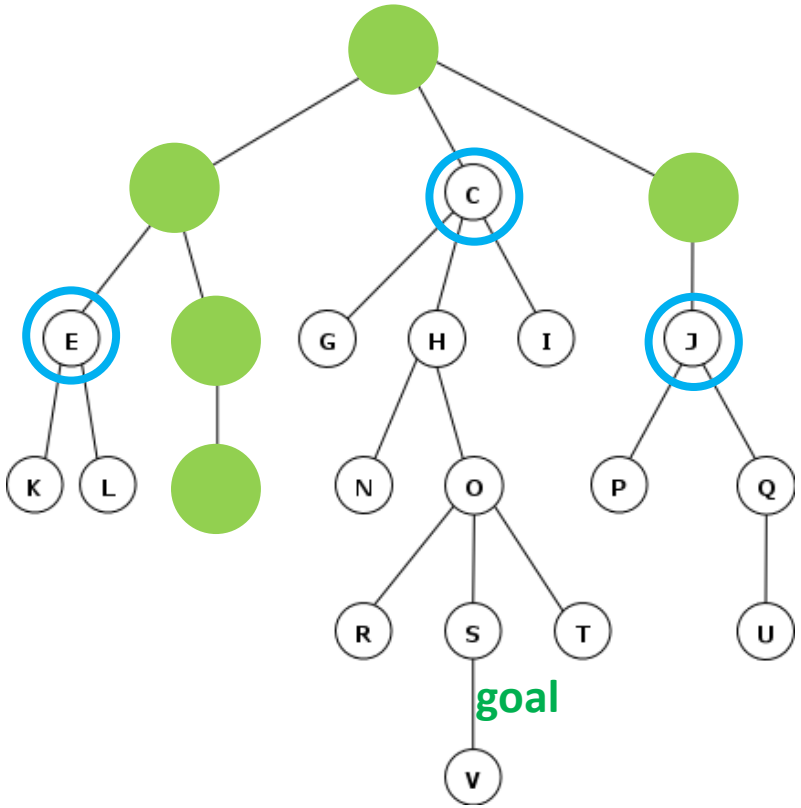
A* Search Tree



CLOSED part of the search tree
g-values = g*-values = true shortest paths (will never decrease any more)

OPEN part of the search tree
g-values may decrease (new paths of better costs might be found)

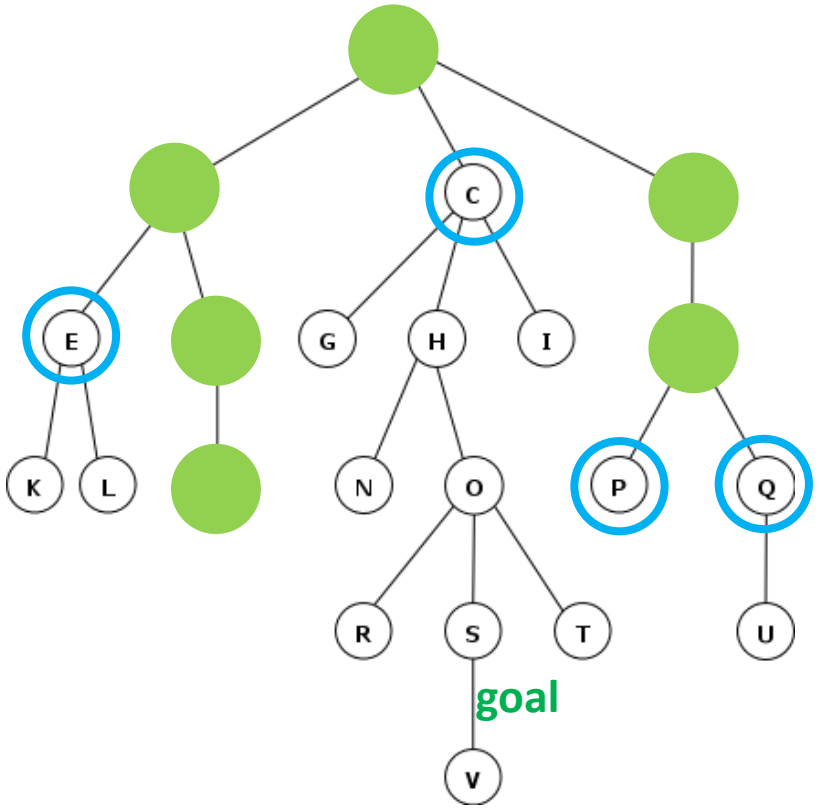
A* Search Tree



CLOSED part of the search tree
g-values = g*-values = true shortest paths (will never decrease any more)

OPEN part of the search tree
g-values may decrease (new paths of better costs might be found)

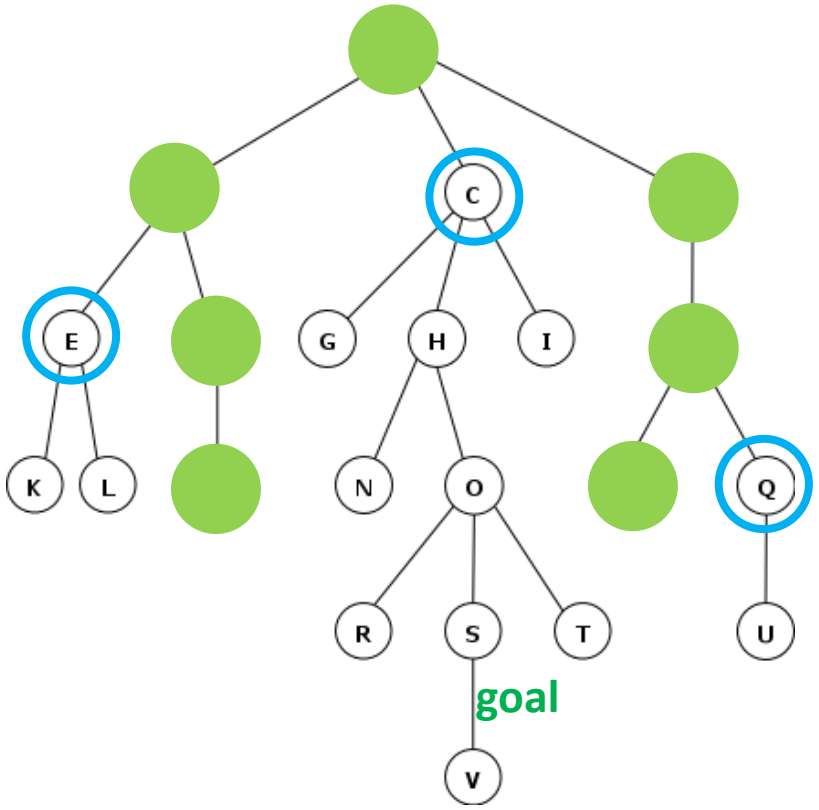
A* Search Tree



CLOSED part of the search tree
g-values = g*-values = true shortest paths (will never decrease any more)

OPEN part of the search tree
g-values may decrease (new paths of better costs might be found)

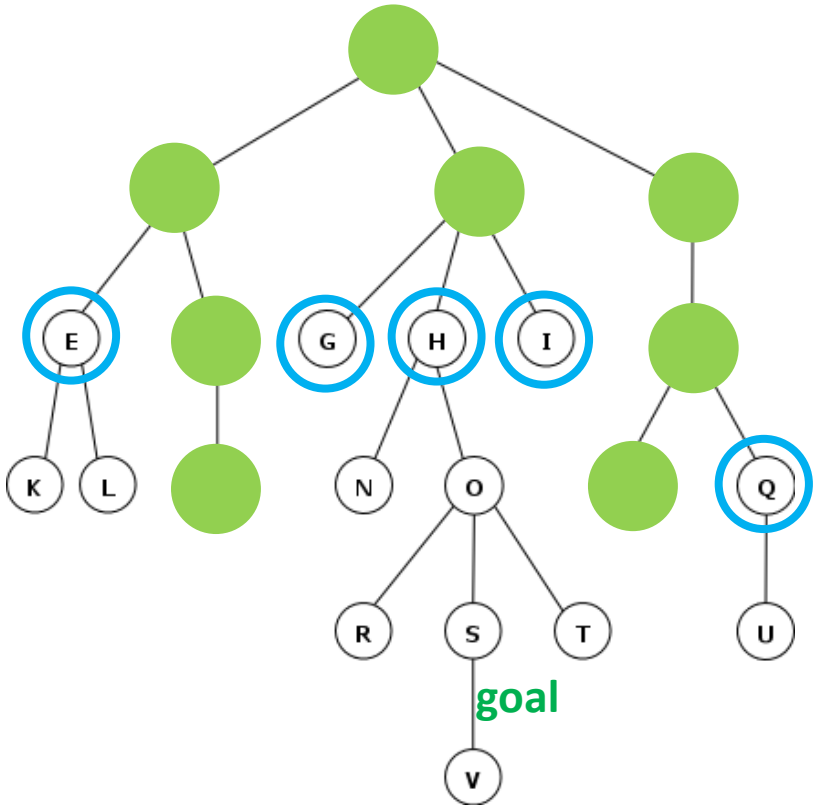
A* Search Tree



CLOSED part of the search tree
g-values = g*-values = true shortest paths (will never decrease any more)

OPEN part of the search tree
g-values may decrease (new paths of better costs might be found)

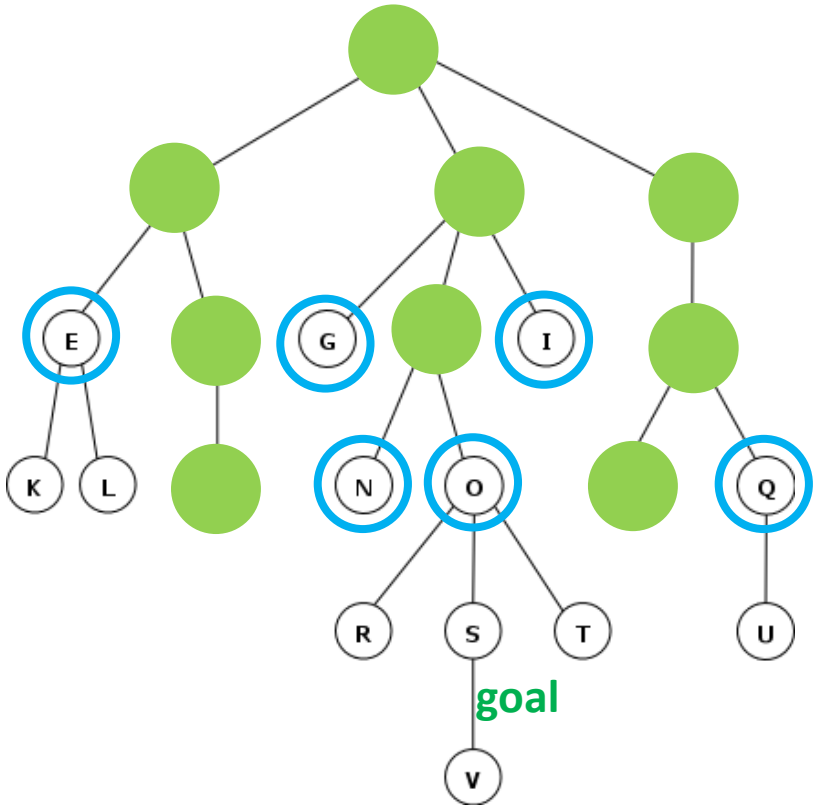
A* Search Tree



CLOSED part of the search tree
g-values = g*-values = true shortest paths (will never decrease any more)

OPEN part of the search tree
g-values may decrease (new paths of better costs might be found)

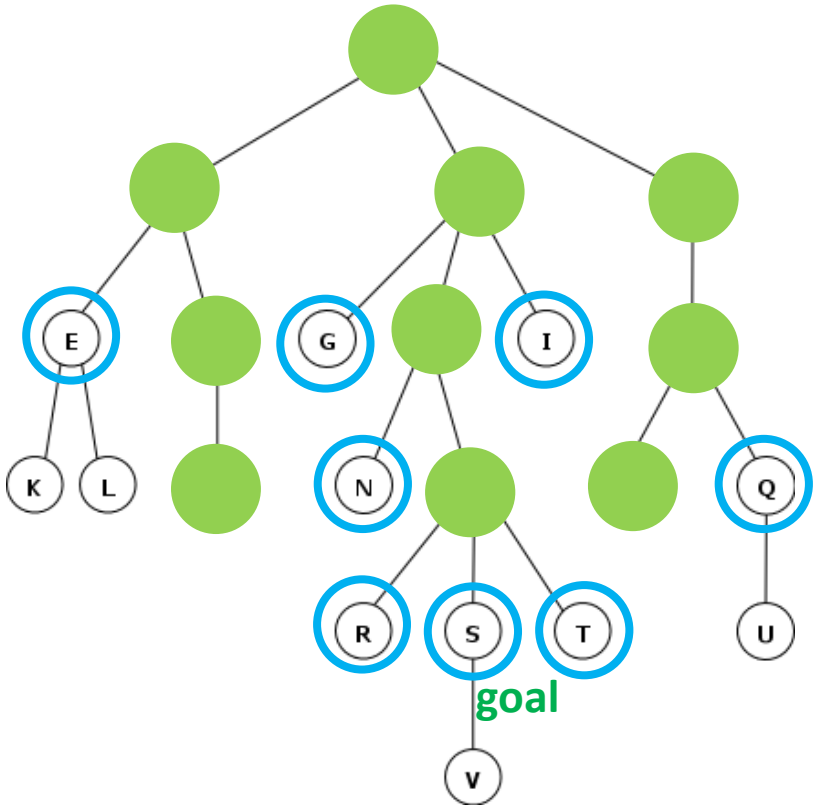
A* Search Tree



CLOSED part of the search tree
g-values = g*-values = true shortest paths (will never decrease any more)

OPEN part of the search tree
g-values may decrease (new paths of better costs might be found)

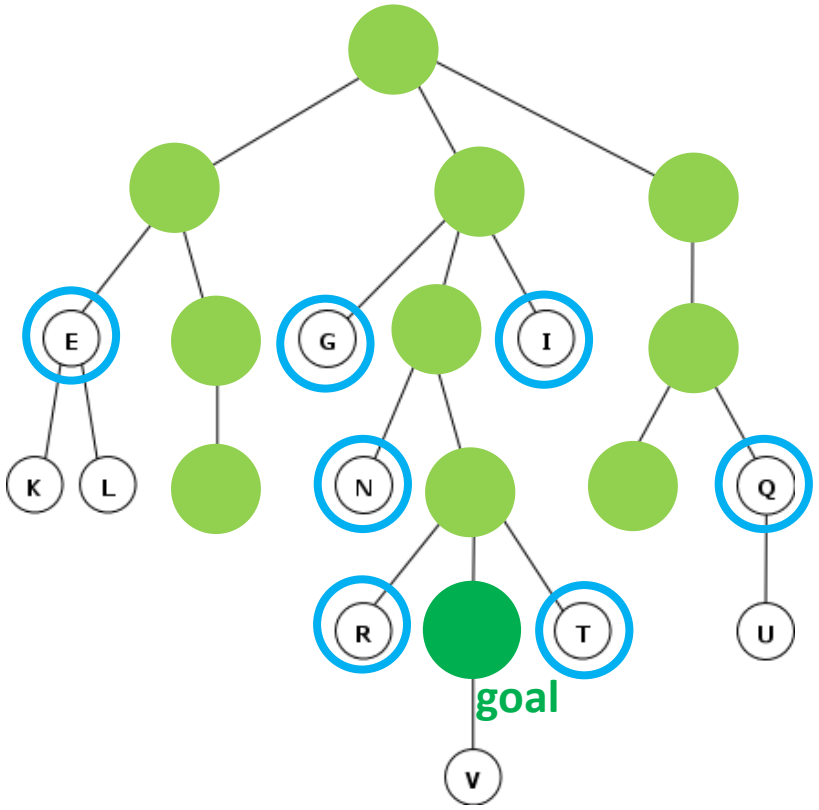
A* Search Tree



CLOSED part of the search tree
g-values = g*-values = true shortest paths (will never decrease any more)

OPEN part of the search tree
g-values may decrease (new paths of better costs might be found)

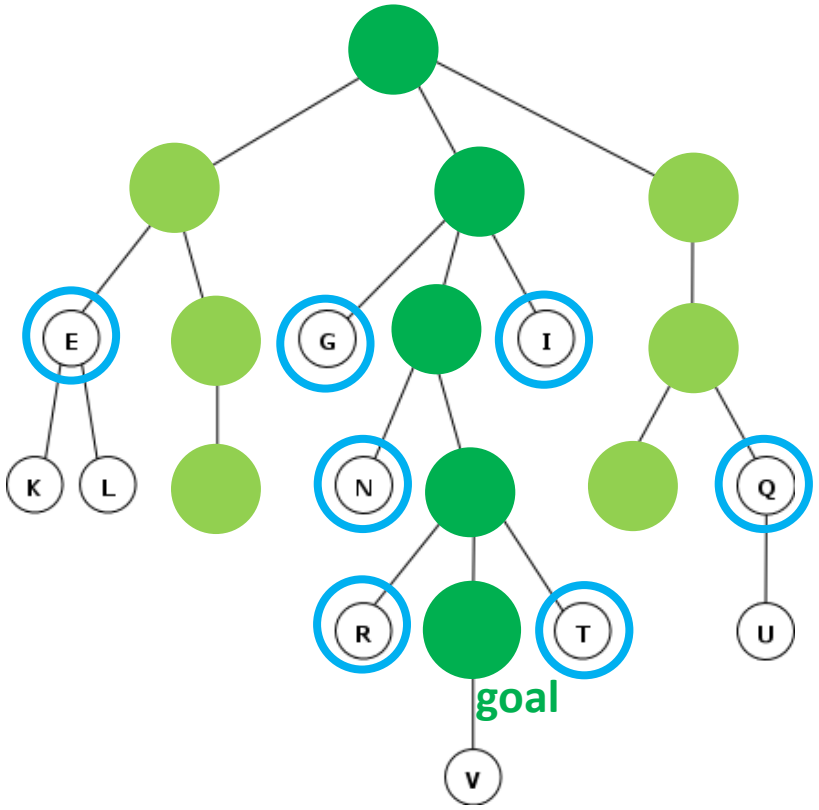
A* Search Tree



CLOSED part of the search tree
g-values = g*-values = true shortest paths (will never decrease any more)

OPEN part of the search tree
g-values may decrease (new paths of better costs might be found)

A* Search Tree



CLOSED part of the search tree
g-values = g*-values = true shortest paths (will never decrease any more)

OPEN part of the search tree
g-values may decrease (new paths of better costs might be found)

Focusing The Search With Heuristics

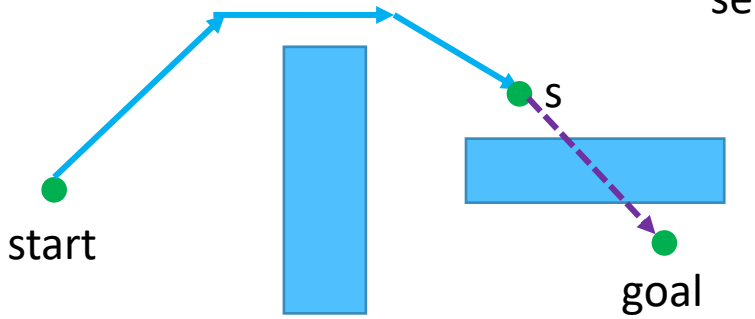
f(s) = g(s) + **h(s)**

g(s) – cost from *start* to *s*

- This cost is computed by the search algorithm
- It is updated as search goes on

h(s) – estimate of the cost from *s* to *goal*

- This is an **educated guess**
- It is NOT (typically) updated as search goes on



f(s) is the approximate cost of reaching *goal* from *start* via *s*

HEURISTIC SEARCH:

On each iteration choose a node that minimizes f(s) not g(s)

Heuristics

Oracle heuristic

h(s)=h*(s) = true path cost from *s* to goal

Heuristics

Oracle heuristic

h(s)=h*(s) = true path cost from *s* to goal

optimal plan?

Heuristics

Oracle heuristic

h(s)=h*(s) = true path cost from *s* to goal

optimal plan **YES**

Heuristics

Oracle heuristic

h(s)=h*(s) = true path cost from *s* to goal

optimal plan **YES**

how many expansions?

Heuristics

Oracle heuristic

h(s)=h*(s) = true path cost from *s* to goal

optimal plan **YES**

only those on (one of) shortest path

Heuristics

Oracle heuristic

h(s)=h*(s) = true path cost from *s* to goal
optimal plan **YES**
only those on (one of) shortest path

Admissible heuristic

h(s) ? h*(s)
optimal plan **YES**

Heuristics

Oracle heuristic

h(s)=h*(s) = true path cost from *s* to goal
optimal plan **YES**
only those on (one of) shortest path

Admissible heuristic

h(s) <= h*(s)
optimal plan **YES**

Heuristics

Oracle heuristic

h(s)=h*(s) = true path cost from *s* to goal
optimal plan **YES**
only those on (one of) shortest path

Admissible heuristic

h(s) <= h*(s)
optimal plan **YES**

Consistent (monotone) heuristic

h(succ(s)) <= h(s) + cost (s, succ(s)), h(goal)=0
optimal plan **YES**

Heuristics

Oracle heuristic

h(s)=h*(s) = true path cost from *s* to goal
optimal plan **YES**
only those on (one of) shortest path

Admissible heuristic

h(s) <= h*(s)
optimal plan **YES**

Consistent (monotone) heuristic

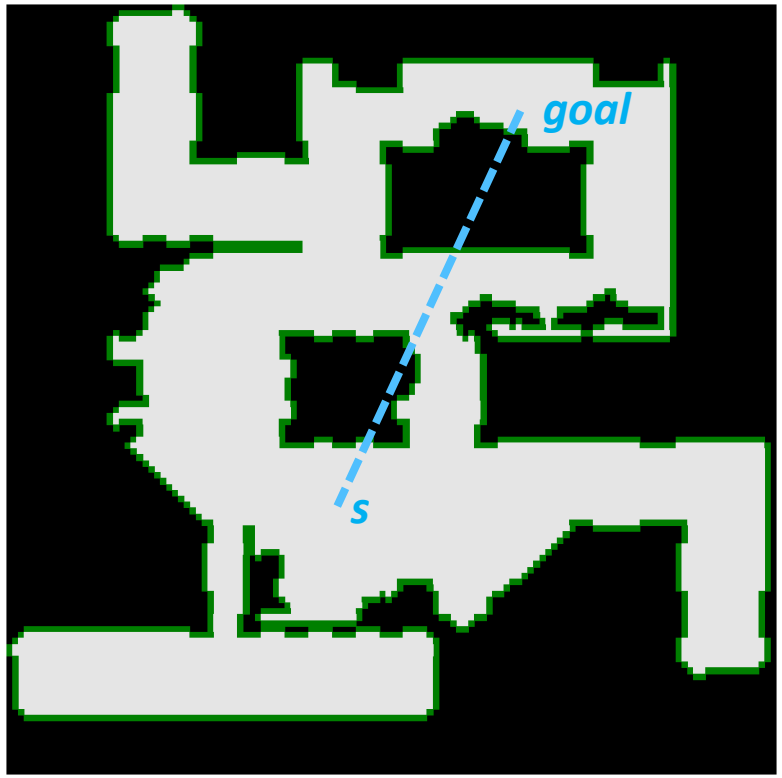
h(succ(s)) <= h(s) + cost (s, succ(s)), h(goal)=0
optimal plan **YES**
no node re-expansions
i.e. can not find a shorter path to an expanded (CLOSED) node

OPEN = {start}
CLOSED = {}

While **OPEN** is not empty
    cur_state = state with **min. f-value** in OPEN
    OPEN.Remove(cur_state)
    if cur_state = goal_state return PathIsFound
    CLOSED.Add(cur_state)
    Generate successors (and prune duplicates)
    Add successors to OPEN
return PathNoFound

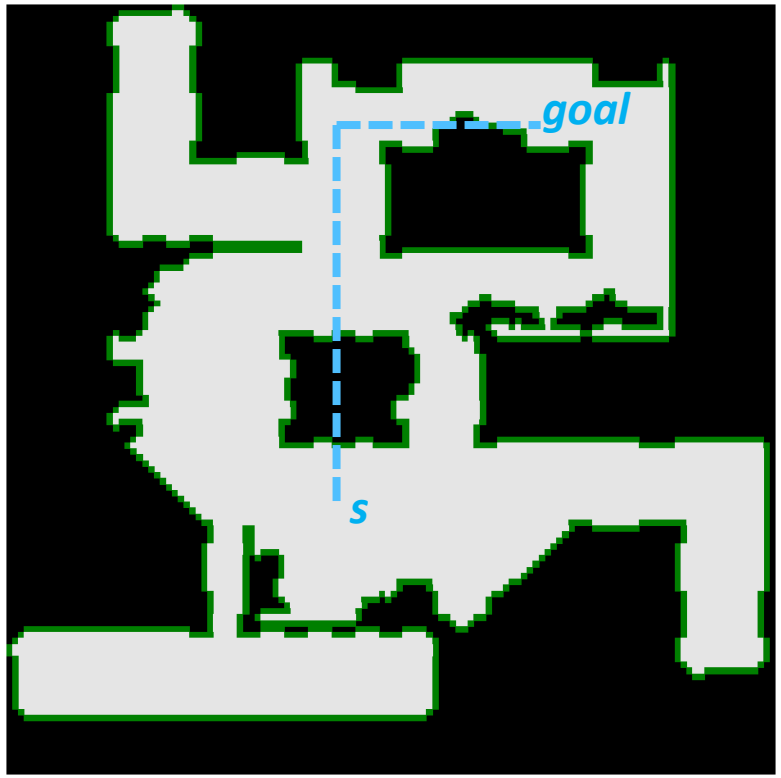
Monotone Heuristics For Grid Worlds

8-connected grid



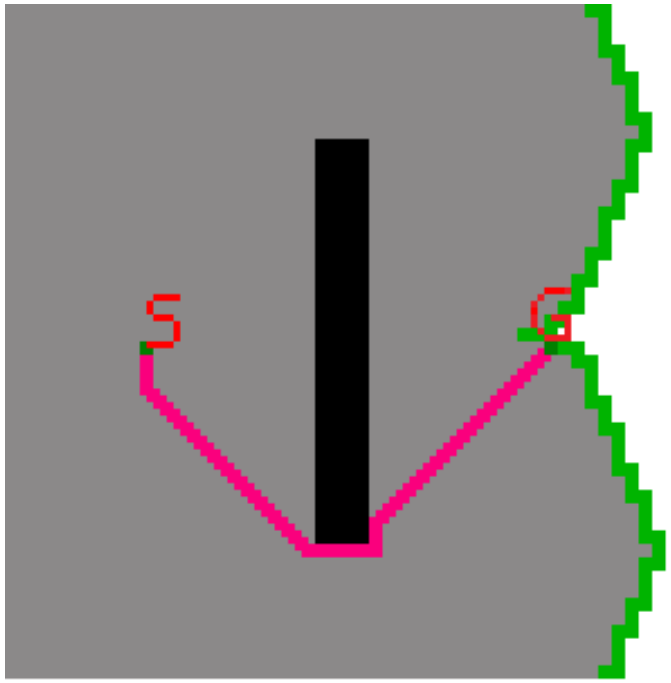
h(s) = Euclidean distance from *s* to *goal*

4-connected grid

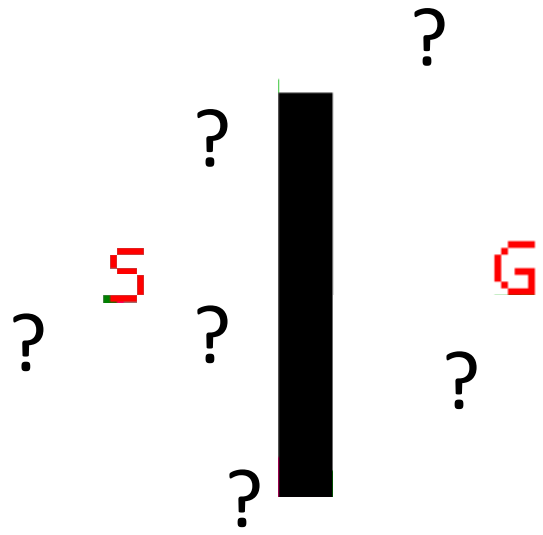


h(s) = Manhattan distance from *s* to *goal*

How Many Expansions?

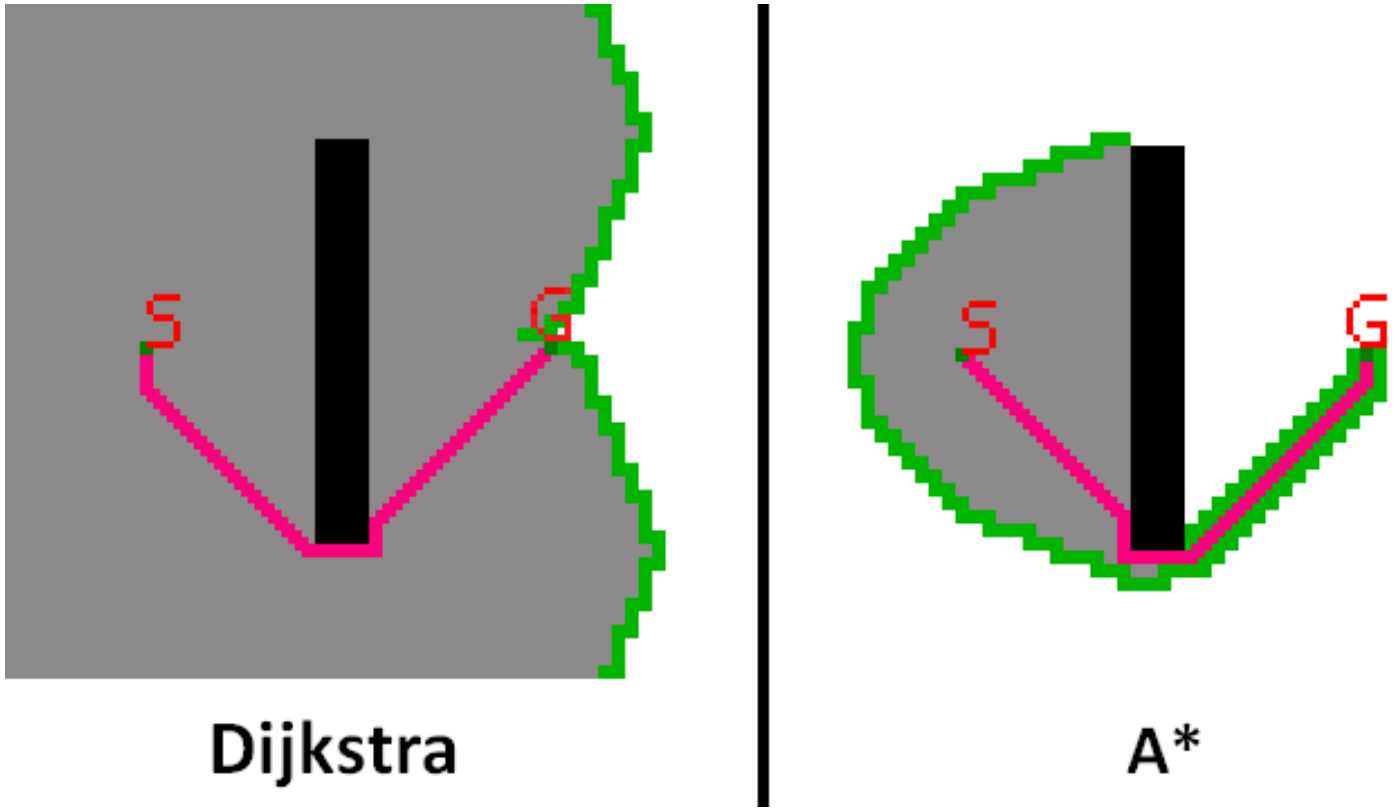


Dijkstra

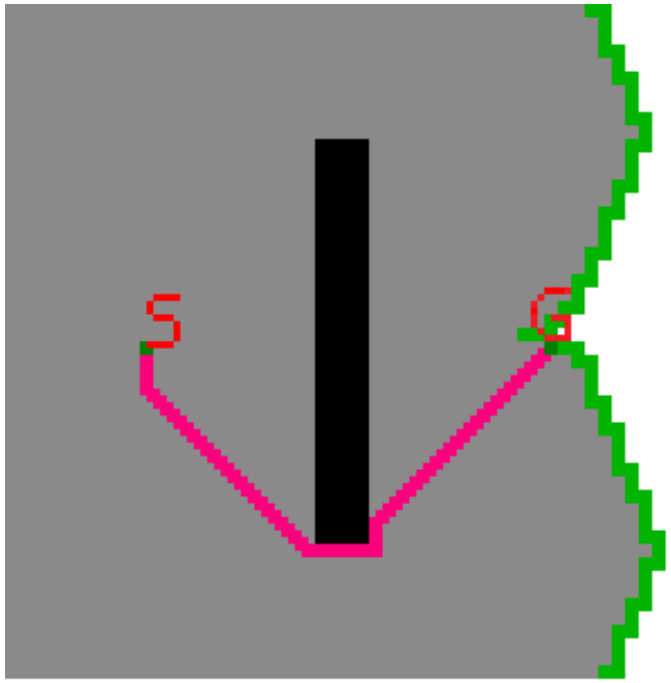


A*

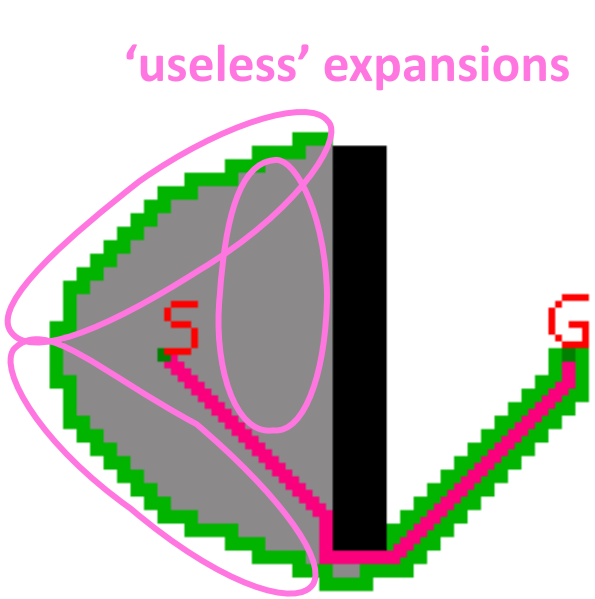
How Many Expansions?



How Many Expansions?

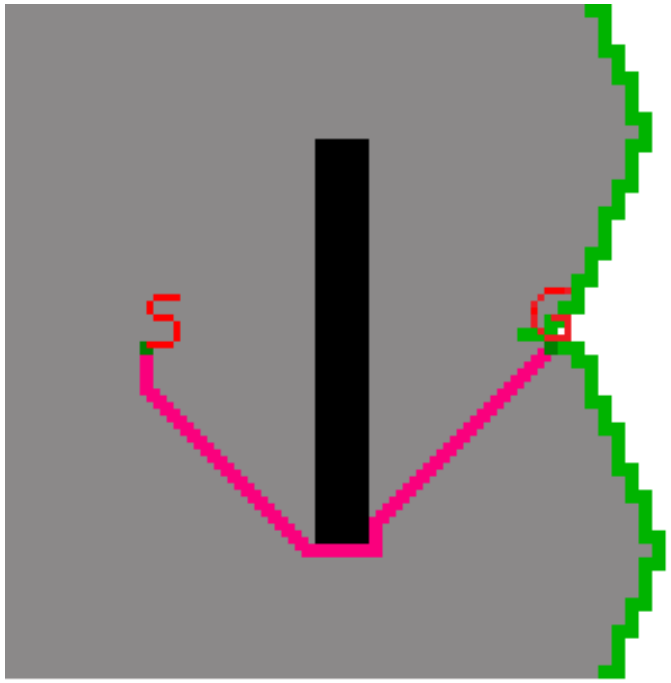


Dijkstra



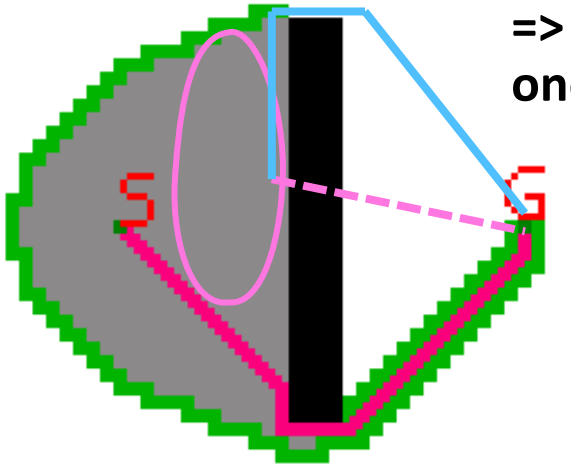
A*

How Many Expansions?



Dijkstra

h-values are way smaller than h*



=> f-values are smaller than the ones lying on the optimal path

A*

How Many Expansions?

C* - cost of the optimal plan, h – consistent heuristic

What nodes will definitely be expanded?

How Many Expansions?

C* - cost of the optimal plan, h – consistent heuristic

What nodes will definitely be expanded?

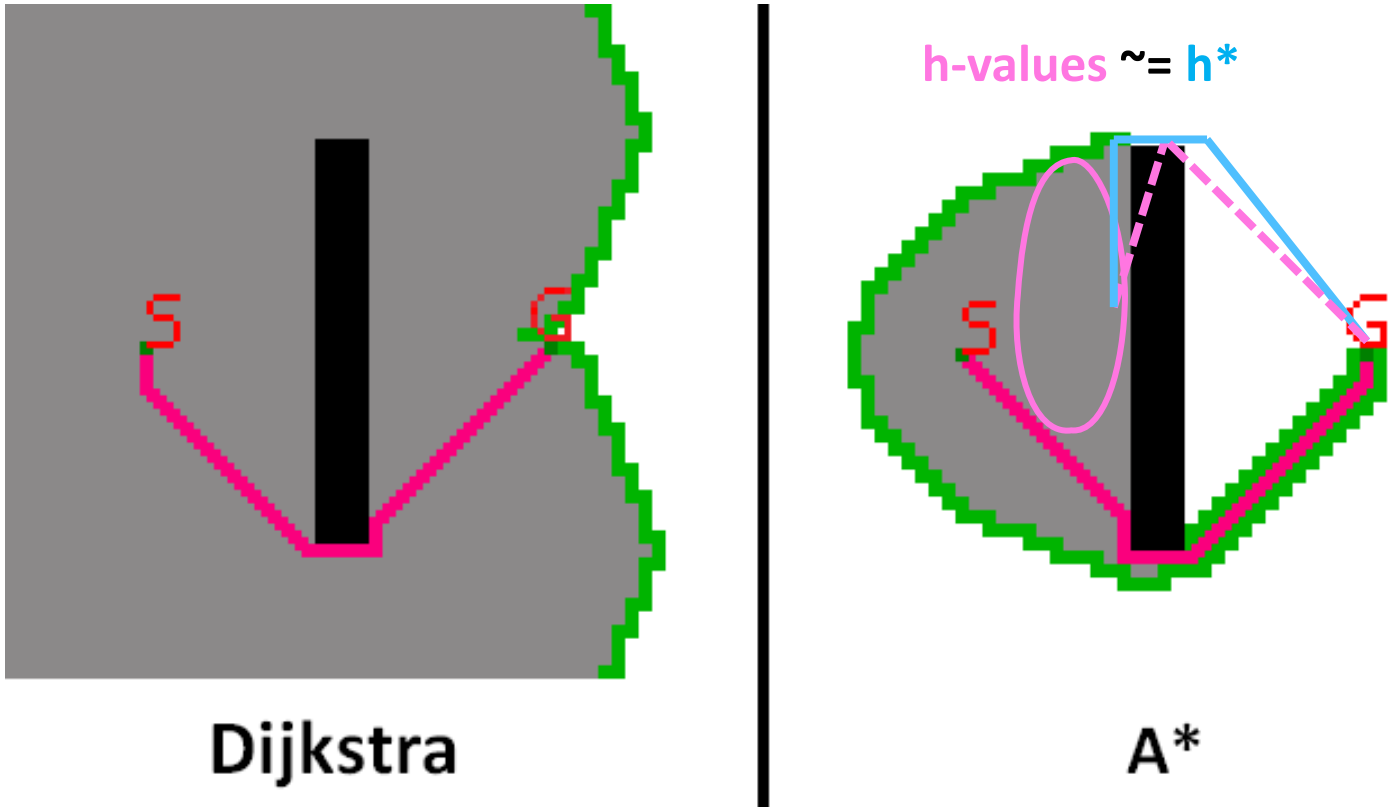
$$\mathbf{f(n) < C^*}$$

$$g(n)+h(n) < C^*$$

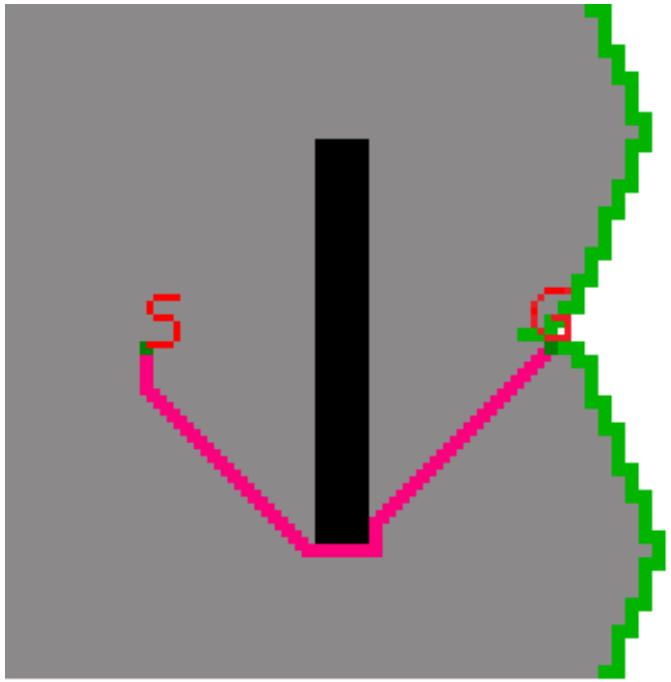
$$0 <= h <= h^*$$

The higher – the better in terms on node expansions

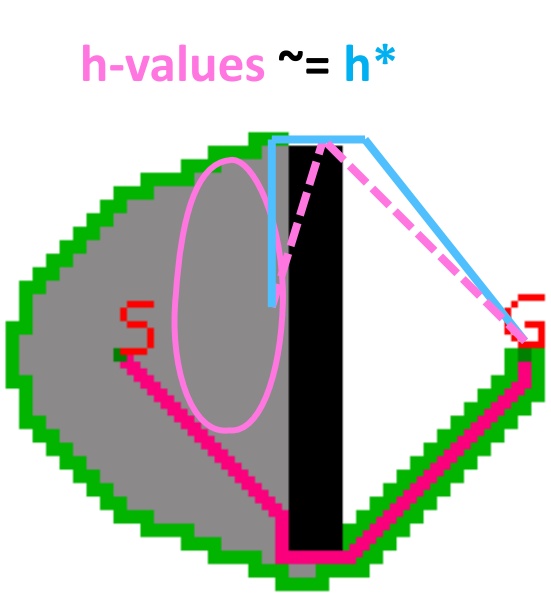
How Can We Make Heuristic More Informative?



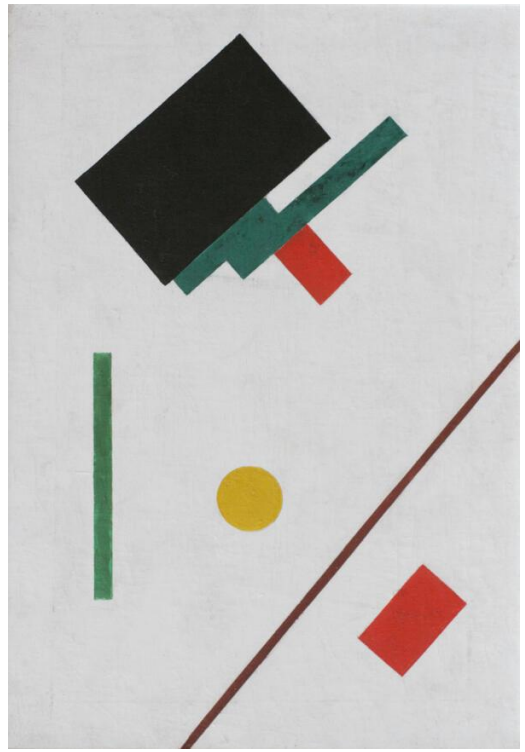
How Can We Make Heuristic More Informative?



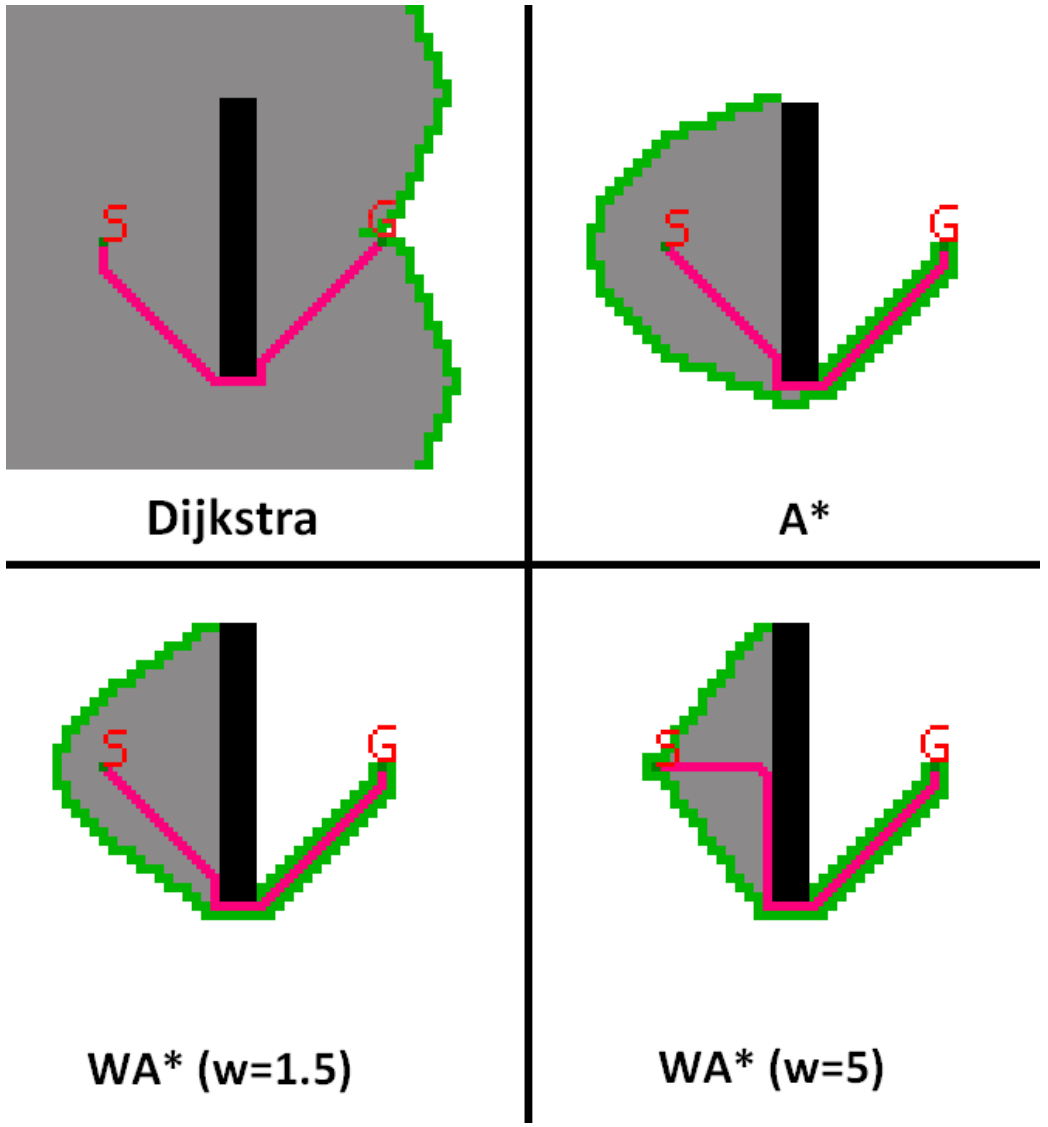
Dijkstra



A*



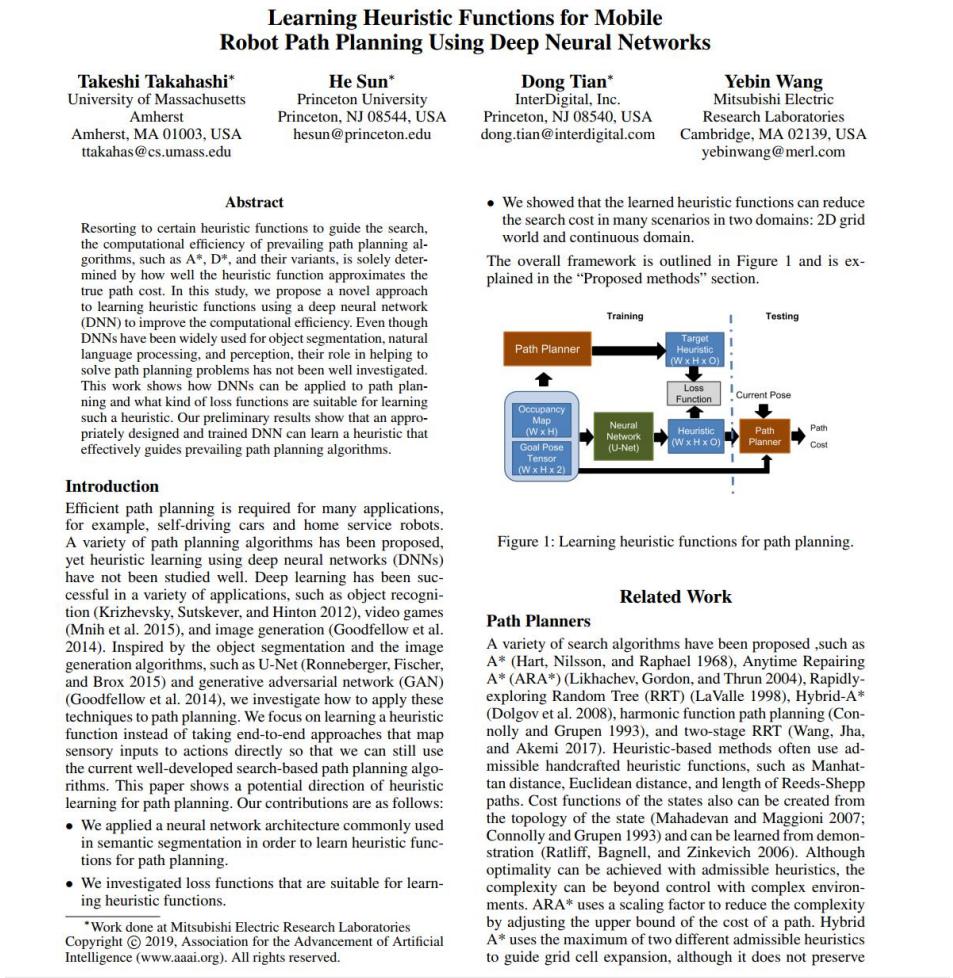
Weighting The Heuristic



03

Integrating Heuristic Search and Machine Learning

Learning Heuristic Functions for Mobile Robot Path Planning Using Deep Neural Networks



Learning Heuristic Functions for Mobile Robot Path Planning Using Deep Neural Networks

Takeshi Takahashi*
University of Massachusetts Amherst
Amherst, MA 01003, USA
ttakahas@cs.umass.edu

He Sun*
Princeton University
Princeton, NJ 08544, USA
hesun@princeton.edu

Dong Tian*
InterDigital, Inc.
Princeton, NJ 08540, USA
dong.tian@interdigital.com

Yebin Wang
Mitsubishi Electric Research Laboratories
Cambridge, MA 02139, USA
yebinwang@merl.com

Abstract

Resorting to certain heuristic functions to guide the search, the computational efficiency of prevailing path planning algorithms, such as A*, D*, and their variants, is solely determined by how well the heuristic function approximates the true path cost. In this study, we propose a novel approach to learning heuristic functions using a deep neural network (DNN) to improve the computational efficiency. Even though DNNs have been widely used for object segmentation, natural language processing, and perception, their role in helping to solve path planning problems has not been well investigated. This work shows how DNNs can be applied to path planning and what kind of loss functions are suitable for learning such a heuristic. Our preliminary results show that an appropriately designed and trained DNN can learn a heuristic that effectively guides prevailing path planning algorithms.

Introduction

Efficient path planning is required for many applications, for example, self-driving cars and home service robots. A variety of path planning algorithms has been proposed, yet heuristic learning using deep neural networks (DNNs) have not been studied well. Deep learning has been successful in a variety of applications, such as object recognition (Krizhevsky, Sutskever, and Hinton 2012), video games (Mnih et al. 2015), and image generation (Goodfellow et al. 2014). Inspired by the object segmentation and the image generation algorithms, such as U-Net (Ronneberger, Fischer, and Brox 2015) and generative adversarial network (GAN) (Goodfellow et al. 2014), we investigate how to apply these techniques to path planning. We focus on learning a heuristic function instead of taking end-to-end approaches that map sensory inputs to actions directly so that we can still use the current well-developed search-based path planning algorithms. This paper shows a potential direction of heuristic learning for path planning. Our contributions are as follows:

- We applied a neural network architecture commonly used in semantic segmentation in order to learn heuristic functions for path planning.
- We investigated loss functions that are suitable for learning heuristic functions.
- We showed that the learned heuristic functions can reduce the search cost in many scenarios in two domains: 2D grid world and continuous domain.

The overall framework is outlined in Figure 1 and is explained in the "Proposed methods" section.

Figure 1: Learning heuristic functions for path planning.

Related Work

Path Planners

A variety of search algorithms have been proposed ,such as A* (Hart, Nilsson, and Raphael 1968), Anytime Repairing A* (ARA*) (Likhachev, Gordon, and Thrun 2004), Rapidly-exploring Random Tree (RRT) (LaValle 1998), Hybrid-A* (Dolgov et al. 2008), harmonic function path planning (Connolly and Grupen 1993), and two-stage RRT (Wang, Jha, and Akemi 2017). Heuristic-based methods often use admissible handcrafted heuristic functions, such as Manhattan distance, Euclidean distance, and length of Reeds-Shepp paths. Cost functions of the states also can be created from the topology of the state (Mahadevan and Maggioni 2007; Connolly and Grupen 1993) and can be learned from demonstration (Ratliff, Bagnell, and Zinkevich 2006). Although optimality can be achieved with admissible heuristics, the complexity can be beyond control with complex environments. ARA* uses a scaling factor to reduce the complexity by adjusting the upper bound of the cost of a path. Hybrid A* uses the maximum of two different admissible heuristics to guide grid cell expansion, although it does not preserve

*Work done at Mitsubishi Electric Research Laboratories

The 29th International Conference on Planning and Scheduling

ICAPS'19

https://ojs.aaai.org/index.php/ICAPS/article/view/3545

AIRI × Университет Сириус

Learning Heuristic Functions for Mobile Robot Path Planning Using Deep Neural Networks

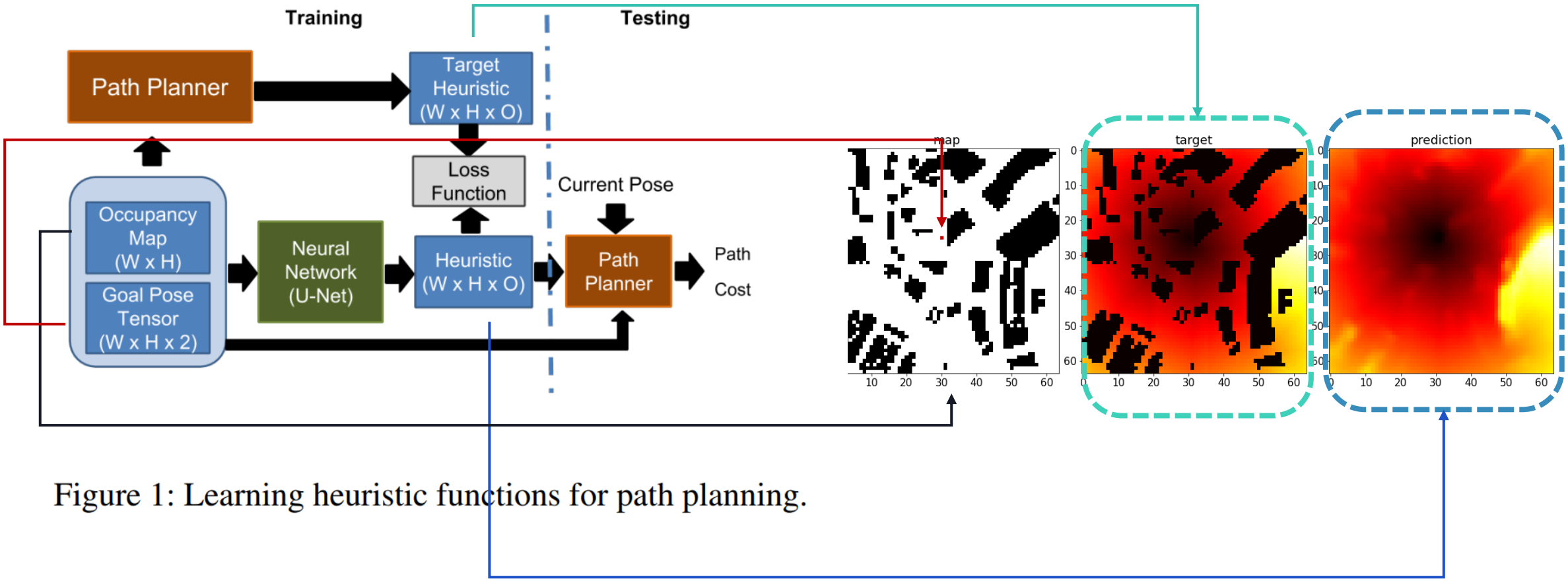


Figure 1: Learning heuristic functions for path planning.

Learning Heuristic Functions for Mobile Robot Path Planning Using Deep Neural Networks :: Loss

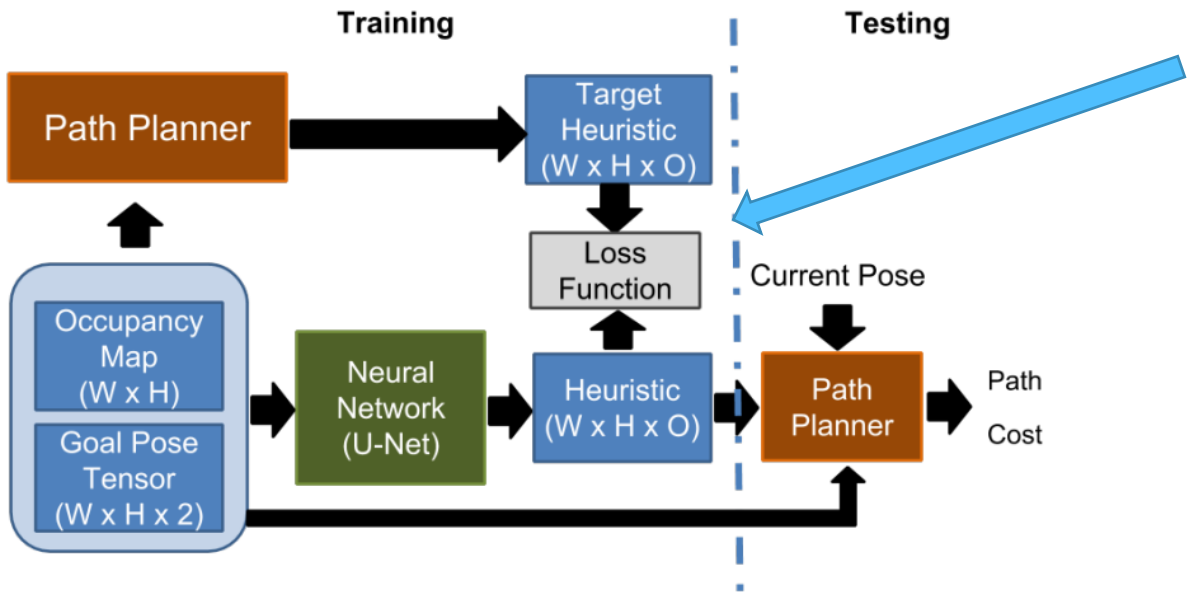


Figure 1: Learning heuristic functions for path planning.

$$MSE(h, h^*) = \frac{1}{N} \sum_{i=0}^{N-1} (h(i) - h^*(i))^2$$

$$MAE(h, h^*) = \frac{1}{N} \sum_{i=0}^{N-1} |h(i) - h^*(i)|$$

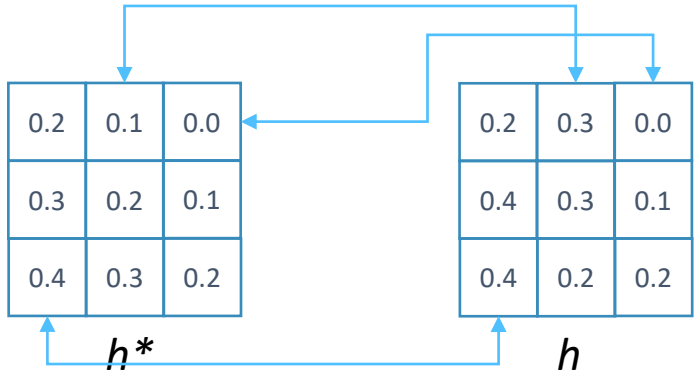
$$Loss_{piece}(h, h^*) = \frac{1}{N} \Bigg(\alpha_1 \sum_{i=0}^{N-1} |h(i) - h^*(i)| \, [h(i) < h_{min}(i)] + \sum_{i=0}^{N-1} |h(i) - h^*(i)| \, [h_{min}(i) \leq h(i) \leq h^*] + \alpha_2 \sum_{i=0}^{N-1} |h(i) - h^*(i)| \, [h^*(i) < h(i)] \Bigg)$$

Learning Heuristic Functions for Mobile Robot Path Planning Using Deep Neural Networks :: Loss

$$MSE(h, h^*) = \frac{1}{N} \sum_{i=0}^{N-1} (h(i) - h^*(i))^2$$

$$MAE(h, h^*) = \frac{1}{N} \sum_{i=0}^{N-1} |h(i) - h^*(i)|$$

$$Loss_{piece}(h, h^*) = \frac{1}{N} \Bigg(\alpha_1 \sum_{i=0}^{N-1} |h(i) - h^*(i)| \, [h(i) < h_{min}(i)] + \sum_{i=0}^{N-1} |h(i) - h^*(i)| \, [h_{min}(i) \leq h(i) \leq h^*] + \alpha_2 \sum_{i=0}^{N-1} |h(i) - h^*(i)| \, [h^*(i) < h(i)] \Bigg)$$



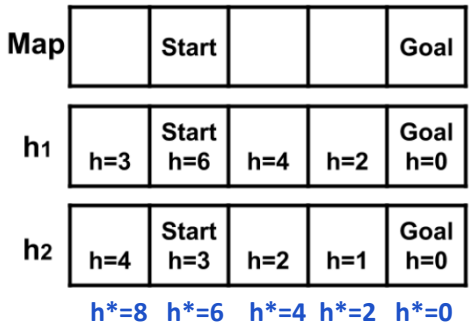
What's the problem?

Learning Heuristic Functions for Mobile Robot Path Planning Using Deep Neural Networks :: Loss

$$MSE(h, h^*) = \frac{1}{N} \sum_{i=0}^{N-1} (h(i) - h^*(i))^2$$

$$MAE(h, h^*) = \frac{1}{N} \sum_{i=0}^{N-1} |h(i) - h^*(i)|$$

$$Loss_{piece}(h, h^*) = \frac{1}{N} \Bigg(\alpha_1 \sum_{i=0}^{N-1} |h(i) - h^*(i)| \, [h(i) < h_{min}(i)] + \sum_{i=0}^{N-1} |h(i) - h^*(i)| \, [h_{min}(i) \le h(i) \le h^*] + \alpha_2 \sum_{i=0}^{N-1} |h(i) - h^*(i)| \, [h^*(i) < h(i)] \Bigg)$$



	MSE	MAE	Search Cost
h_1	5.0	1.0	5
h_2	5.8	1.4	4

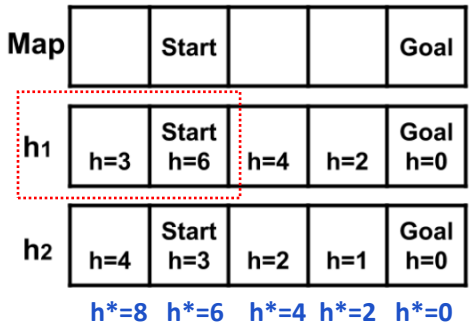
Lower loss != better search performance

Learning Heuristic Functions for Mobile Robot Path Planning Using Deep Neural Networks :: Loss

$$MSE(h, h^*) = \frac{1}{N} \sum_{i=0}^{N-1} (h(i) - h^*(i))^2$$

$$MAE(h, h^*) = \frac{1}{N} \sum_{i=0}^{N-1} |h(i) - h^*(i)|$$

$$Loss_{piece}(h, h^*) = \frac{1}{N} \Bigg(\alpha_1 \sum_{i=0}^{N-1} |h(i) - h^*(i)| \, [h(i) < h_{min}(i)] + \sum_{i=0}^{N-1} |h(i) - h^*(i)| \, [h_{min}(i) \le h(i) \le h^*] + \alpha_2 \sum_{i=0}^{N-1} |h(i) - h^*(i)| \, [h^*(i) < h(i)] \Bigg)$$



	MSE	MAE	Search Cost
h_1	5.0	1.0	5
h_2	5.8	1.4	4

Solution: extra loss term

$$Loss = Loss_1 + \alpha Loss_{grad}$$

Learning Heuristic Functions for Mobile Robot Path Planning Using Deep Neural Networks :: Loss

$$MSE(h, h^*) = \frac{1}{N} \sum_{i=0}^{N-1} (h(i) - h^*(i))^2$$

$$MAE(h, h^*) = \frac{1}{N} \sum_{i=0}^{N-1} |h(i) - h^*(i)|$$

$$Loss_{piece}(h, h^*) = \frac{1}{N} \Bigg(\alpha_1 \sum_{i=0}^{N-1} |h(i) - h^*(i)| \, [h(i) < h_{min}(i)] + \sum_{i=0}^{N-1} |h(i) - h^*(i)| \, [h_{min}(i) \le h(i) \le h^*] + \alpha_2 \sum_{i=0}^{N-1} |h(i) - h^*(i)| \, [h^*(i) < h(i)] \Bigg)$$

$$Loss_{grad}(h, h^*) = \sum_{a \in A} MAE(K_a * h, K_a * h^*)$$

$$K_{north} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Solution: extra loss term

$$Loss = Loss_1 + \alpha Loss_{grad}$$

Learning Heuristic Functions for Mobile Robot Path Planning Using Deep Neural Networks :: Loss

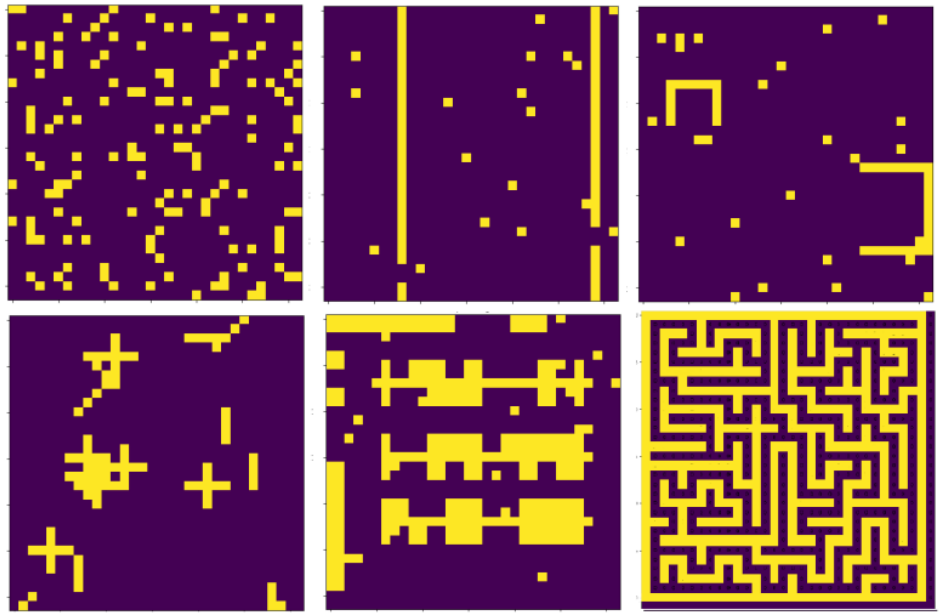
Why not add GANs on top

$$Loss = Loss_1 + \alpha Loss_{grad} + \beta Loss_{WGAN}$$

Learning Heuristic Functions for Mobile Robot Path Planning Using Deep Neural Networks :: Evaluation

1. 2D grids (no orientation)

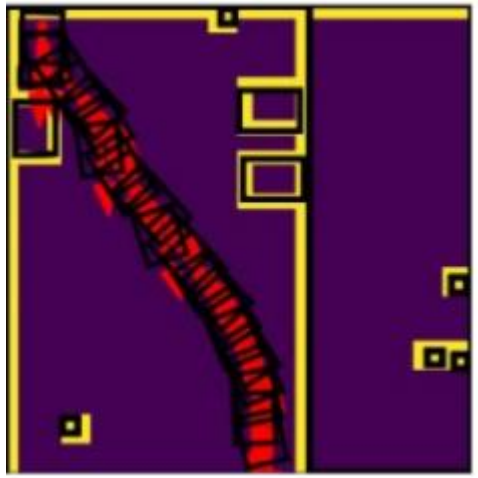
seems like 32 x 32
4-connected



180 000 training examples
10 000 testing examples

from other maps?

2. Parking lot (32 x 32 x 32)



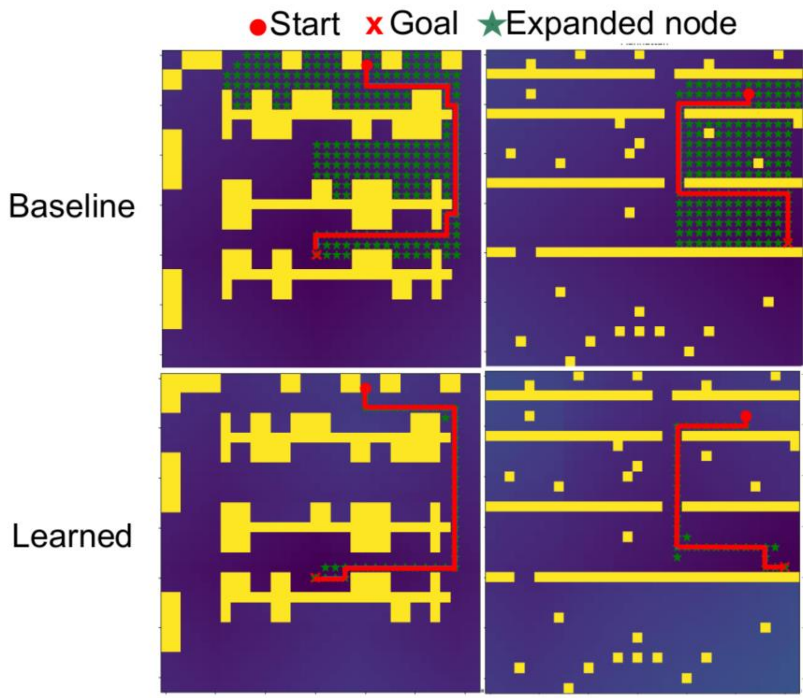
Learning Heuristic Functions for Mobile Robot Path Planning Using Deep Neural Networks :: Empirical results

Difficulty: h*(start)/h_manh(start)

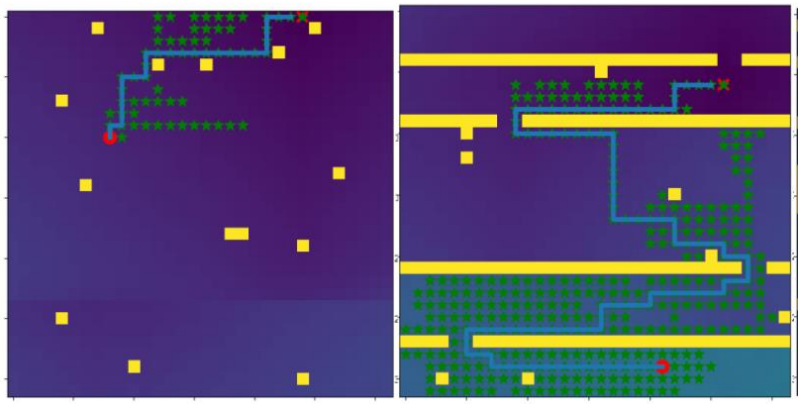
Loss	Complexity Evaluation Ratio of the number of expanded nodes on different task difficulties									
	1.0-1.2	1.2-1.4	1.4-1.6	1.6-1.8	1.8-2.0	2.0-2.2	2.2-2.4	2.4-2.6	2.6-2.8	≥ 2.8
MSE	0.64	0.57	**0.56**	**0.57**	**0.56**	**0.56**	**0.60**	**0.64**	**0.65**	**0.71**
MSE, $loss_{grad}$	0.50	**0.42**	**0.41**	**0.40**	**0.46**	**0.46**	**0.48**	**0.53**	**0.53**	**0.62**
MAE	0.57	**0.46**	**0.45**	**0.42**	**0.48**	**0.51**	**0.58**	**0.60**	**0.59**	**0.62**
MAE, $loss_{grad}$	0.60	**0.45**	**0.45**	**0.46**	**0.47**	**0.51**	**0.51**	**0.60**	**0.55**	**0.65**
MSE, $loss_{WGAN}$	0.58	0.53	**0.52**	**0.49**	**0.47**	**0.50**	**0.51**	**0.53**	**0.63**	**0.71**
MAE, $loss_{grad}$, $loss_{WGAN}$	0.54	**0.45**	**0.42**	**0.41**	**0.43**	**0.45**	**0.47**	**0.53**	**0.56**	**0.61**
Piecewise	0.57	**0.46**	**0.44**	**0.42**	**0.44**	**0.47**	**0.46**	**0.54**	**0.52**	**0.58**
Piecewise, $loss_{grad}$	0.54	**0.45**	**0.45**	**0.43**	**0.46**	**0.51**	**0.48**	**0.58**	**0.51**	**0.58**
Baseline (Manhattan)	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Baseline (Scaled Manhattan)	0.41	0.49	0.63	0.69	0.72	0.75	0.78	0.81	0.82	0.84

Table 1: Results of experiments in the toy domain. The number shows the mean of the ratio of the number of expanded nodes where the ratio $= \frac{N}{N_b}$, N is the number of expanded nodes using the learned heuristic, and N_b is the number of expanded nodes using the Manhattan distance. Task difficulty is computed by Equation 10. If the task difficulty is $[1.0, 1.2]$, this indicates the task is "easy". The bold numbers indicate that the leaned heuristic is better than the baselines.

Learning Heuristic Functions for Mobile Robot Path Planning Using Deep Neural Networks :: Empirical results

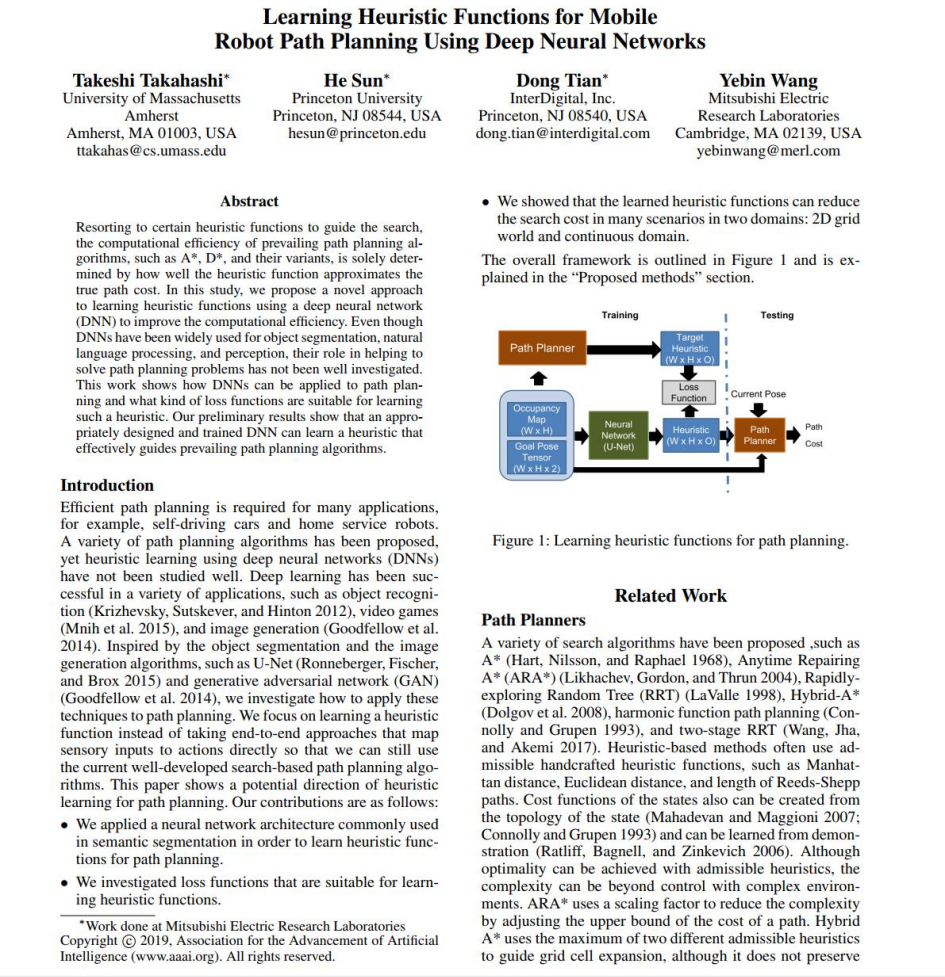


Successful runs



Not that successful runs

Learning Heuristic Functions for Mobile Robot Path Planning Using Deep Neural Networks :: Overall



Learning Heuristic Functions for Mobile Robot Path Planning Using Deep Neural Networks

Takeshi Takahashi* University of Massachusetts Amherst, Amherst, MA 01003, USA, ttakahas@cs.umass.edu
He Sun* Princeton University, Princeton, NJ 08544, USA, hesun@princeton.edu
Dong Tian* InterDigital, Inc., Princeton, NJ 08540, USA, dong.tian@interdigital.com
Yebin Wang Mitsubishi Electric Research Laboratories, Cambridge, MA 02139, USA, yebinwang@merl.com

Abstract

Resorting to certain heuristic functions to guide the search, the computational efficiency of prevailing path planning algorithms, such as A*, D*, and their variants, is solely determined by how well the heuristic function approximates the true path cost. In this study, we propose a novel approach to learning heuristic functions using a deep neural network (DNN) to improve the computational efficiency. Even though DNNs have been widely used for object segmentation, natural language processing, and perception, their role in helping to solve path planning problems has not been well investigated. This work shows how DNNs can be applied to path planning and what kind of loss functions are suitable for learning such a heuristic. Our preliminary results show that an appropriately designed and trained DNN can learn a heuristic that effectively guides prevailing path planning algorithms.

Introduction

Efficient path planning is required for many applications, for example, self-driving cars and home service robots. A variety of path planning algorithms has been proposed, yet heuristic learning using deep neural networks (DNNs) have not been studied well. Deep learning has been successful in a variety of applications, such as object recognition (Krizhevsky, Sutskever, and Hinton 2012), video games (Mnih et al. 2015), and image generation (Goodfellow et al. 2014). Inspired by the object segmentation and the image generation algorithms, such as U-Net (Ronneberger, Fischer, and Brox 2015) and generative adversarial network (GAN) (Goodfellow et al. 2014), we investigate how to apply these techniques to path planning. We focus on learning a heuristic function instead of taking end-to-end approaches that map sensory inputs to actions directly so that we can still use the current well-developed search-based path planning algorithms. This paper shows a potential direction of heuristic learning for path planning. Our contributions are as follows:

- We applied a neural network architecture commonly used in semantic segmentation in order to learn heuristic functions for path planning.
- We investigated loss functions that are suitable for learning heuristic functions.
- We showed that the learned heuristic functions can reduce the search cost in many scenarios in two domains: 2D grid world and continuous domain.

The overall framework is outlined in Figure 1 and is explained in the "Proposed methods" section.

Figure 1: Learning heuristic functions for path planning.

Related Work

Path Planners

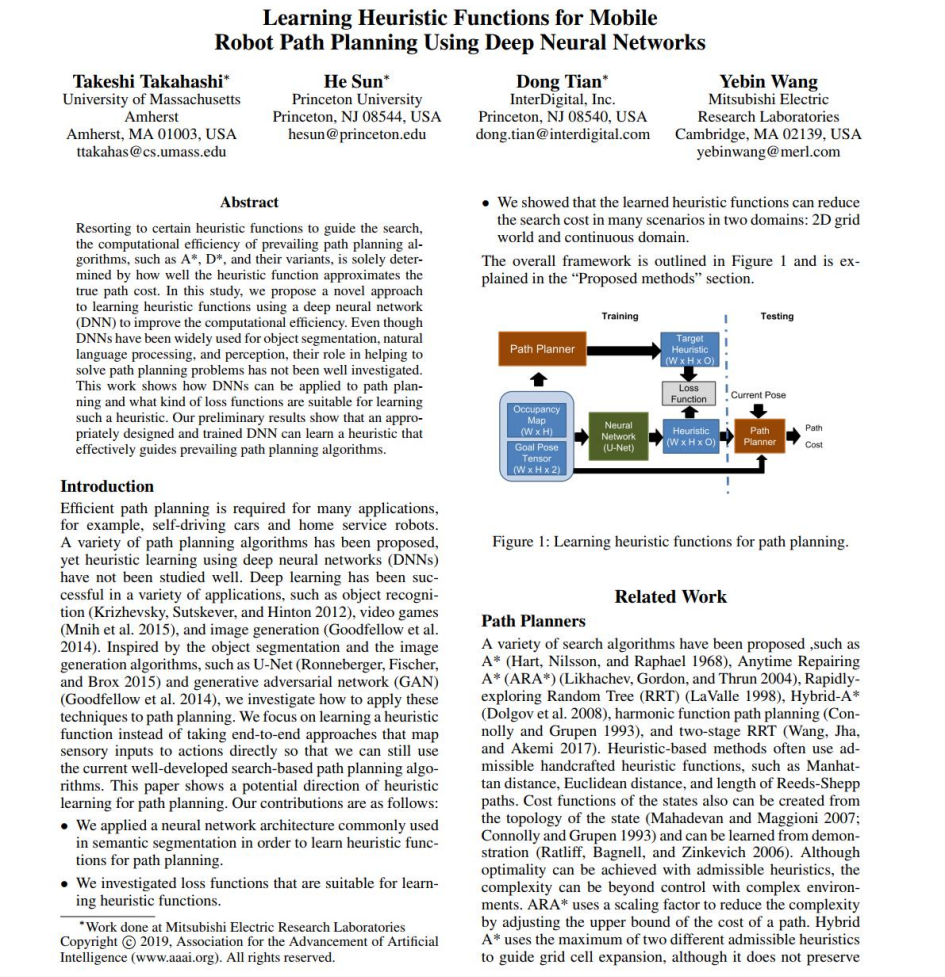
A variety of search algorithms have been proposed ,such as A* (Hart, Nilsson, and Raphael 1968), Anytime Repairing A* (ARA*) (Likhachev, Gordon, and Thrun 2004), Rapidly-exploring Random Tree (RRT) (LaValle 1998), Hybrid-A* (Dolgov et al. 2008), harmonic function path planning (Connolly and Grupen 1993), and two-stage RRT (Wang, Jha, and Akemi 2017). Heuristic-based methods often use admissible handcrafted heuristic functions, such as Manhattan distance, Euclidean distance, and length of Reeds-Shepp paths. Cost functions of the states also can be created from the topology of the state (Mahadevan and Maggioni 2007; Connolly and Grupen 1993) and can be learned from demonstration (Ratliff, Bagnell, and Zinkevich 2006). Although optimality can be achieved with admissible heuristics, the complexity can be beyond control with complex environments. ARA* uses a scaling factor to reduce the complexity by adjusting the upper bound of the cost of a path. Hybrid A* uses the maximum of two different admissible heuristics to guide grid cell expansion, although it does not preserve

*Work done at Mitsubishi Electric Research Laboratories

- Supervised learning
- Data-driven
- CNN (U-net)
- Tuned Loss
- Weak evaluation
  - Description
  - Small dataset
- No Code :(

AIRI × Университет Сириус

Learning Heuristic Functions for Mobile Robot Path Planning Using Deep Neural Networks :: Overall



Learning Heuristic Functions for Mobile Robot Path Planning Using Deep Neural Networks

Takeshi Takahashi* University of Massachusetts Amherst, Amherst, MA 01003, USA, ttakahas@cs.umass.edu
He Sun* Princeton University, Princeton, NJ 08544, USA, hesun@princeton.edu
Dong Tian* InterDigital, Inc., Princeton, NJ 08540, USA, dong.tian@interdigital.com
Yebin Wang Mitsubishi Electric Research Laboratories, Cambridge, MA 02139, USA, yebinwang@merl.com

Abstract

Resorting to certain heuristic functions to guide the search, the computational efficiency of prevailing path planning algorithms, such as A*, D*, and their variants, is solely determined by how well the heuristic function approximates the true path cost. In this study, we propose a novel approach to learning heuristic functions using a deep neural network (DNN) to improve the computational efficiency. Even though DNNs have been widely used for object segmentation, natural language processing, and perception, their role in helping to solve path planning problems has not been well investigated. This work shows how DNNs can be applied to path planning and what kind of loss functions are suitable for learning such a heuristic. Our preliminary results show that an appropriately designed and trained DNN can learn a heuristic that effectively guides prevailing path planning algorithms.

Introduction

Efficient path planning is required for many applications, for example, self-driving cars and home service robots. A variety of path planning algorithms has been proposed, yet heuristic learning using deep neural networks (DNNs) have not been studied well. Deep learning has been successful in a variety of applications, such as object recognition (Krizhevsky, Sutskever, and Hinton 2012), video games (Mnih et al. 2015), and image generation (Goodfellow et al. 2014). Inspired by the object segmentation and the image generation algorithms, such as U-Net (Ronneberger, Fischer, and Brox 2015) and generative adversarial network (GAN) (Goodfellow et al. 2014), we investigate how to apply these techniques to path planning. We focus on learning a heuristic function instead of taking end-to-end approaches that map sensory inputs to actions directly so that we can still use the current well-developed search-based path planning algorithms. This paper shows a potential direction of heuristic learning for path planning. Our contributions are as follows:

- We applied a neural network architecture commonly used in semantic segmentation in order to learn heuristic functions for path planning.
- We investigated loss functions that are suitable for learning heuristic functions.
- We showed that the learned heuristic functions can reduce the search cost in many scenarios in two domains: 2D grid world and continuous domain.

The overall framework is outlined in Figure 1 and is explained in the "Proposed methods" section.

Figure 1: Learning heuristic functions for path planning.

Related Work

Path Planners

A variety of search algorithms have been proposed ,such as A* (Hart, Nilsson, and Raphael 1968), Anytime Repairing A* (ARA*) (Likhachev, Gordon, and Thrun 2004), Rapidly-exploring Random Tree (RRT) (LaValle 1998), Hybrid-A* (Dolgov et al. 2008), harmonic function path planning (Connolly and Grupen 1993), and two-stage RRT (Wang, Jha, and Akemi 2017). Heuristic-based methods often use admissible handcrafted heuristic functions, such as Manhattan distance, Euclidean distance, and length of Reeds-Shepp paths. Cost functions of the states also can be created from the topology of the state (Mahadevan and Maggioni 2007; Connolly and Grupen 1993) and can be learned from demonstration (Ratliff, Bagnell, and Zinkevich 2006). Although optimality can be achieved with admissible heuristics, the complexity can be beyond control with complex environments. ARA* uses a scaling factor to reduce the complexity by adjusting the upper bound of the cost of a path. Hybrid A* uses the maximum of two different admissible heuristics to guide grid cell expansion, although it does not preserve

*Work done at Mitsubishi Electric Research Laboratories

- Supervised learning
- Data-driven
- CNN (U-net)
- Tuned Loss
- Weak evaluation
  - Description
  - Small dataset
- No Code :(

Sounds like a project for the summer school

AIRI × Университет Сириус

Path Planning Using Neural A* Search

Path Planning using Neural A* Search

Ryo Yonetani [*1] **Tatsunori Taniai** [*1] **Mohammadamin Barekatain** [1 2] **Mai Nishimura** [1] **Asako Kanezaki** [3]

Abstract

We present *Neural A**, a novel data-driven search method for path planning problems. Despite the recent increasing attention to data-driven path planning, machine learning approaches to search-based planning are still challenging due to the discrete nature of search algorithms. In this work, we reformulate a canonical A* search algorithm to be differentiable and couple it with a convolutional encoder to form an end-to-end trainable neural network planner. Neural A* solves a path planning problem by encoding a problem instance to a guidance map and then performing the differentiable A* search with the guidance map. By learning to match the search results with ground-truth paths provided by experts, Neural A* can produce a path consistent with the ground truth accurately and efficiently. Our extensive experiments confirmed that Neural A* outperformed state-of-the-art data-driven planners in terms of the search optimality and efficiency trade-off. Furthermore, Neural A* successfully predicted realistic human trajectories by directly performing search-based planning on natural image inputs[1].

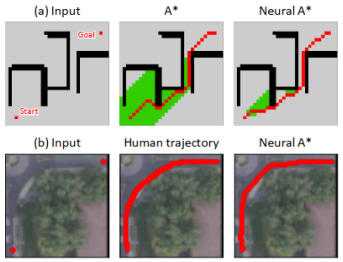


Figure 1. **Two Scenarios of Path Planning with Neural A*.** (a) Point-to-point shortest path search: finding a near-optimal path (red) with fewer node explorations (green) for an input map. (b) Path planning on raw image inputs: accurately predicting a human trajectory (red) on a natural image.

1. Introduction

Path planning refers to the problem of finding a valid low-cost path from start to goal points in an environmental map. *Search-based* planning, including the popular A* search (Hart et al., 1968), is a common approach to path planning problems and has been used in a wide range of applications such as autonomous vehicle navigation (Paden et al., 2016), robot arm manipulation (Smith et al., 2012), and game AI (Abd Algfoor et al., 2015). Compared to other planning approaches such as sampling-based planning (González et al., 2015) and reactive planning (Tamar et al., 2016; Lee et al., 2018), search-based planning is guaranteed to find a solution path, if one exists, by incrementally and extensively exploring the map.

Learning how to plan from expert demonstrations is gathering attention as promising extensions to classical path planners. Recent work has demonstrated major advantages of such data-driven path planning in two scenarios: (1) finding near-optimal paths more efficiently than classical heuristic planners in point-to-point shortest path search problems (Choudhury et al., 2018; Qureshi et al., 2019; Takahashi et al., 2019; Chen et al., 2020; Ichter et al., 2020) and (2) enabling path planning on raw image inputs (Tamar et al., 2016; Lee et al., 2018; Ichter & Pavone, 2019; Vlastelica et al., 2020), which is hard for classical planners unless semantic pixel-wise labeling of the environment is given.

In this study, we address both of these separately studied scenarios in a principled fashion, as highlighted in Fig. 1. In contrast to most of the existing data-driven methods that extend either sampling-based or reactive planning, we pursue a search-based approach to data-driven planning with the intrinsic advantage of guaranteed planning success.

Studies in this direction so far are largely limited due to

[*]Equal contribution [1]OMRON SINIC X, Tokyo, Japan [2]Now at DeepMind, London, UK. [3]Tokyo Institute of Technology, Tokyo, Japan. Correspondence to: Ryo Yonetani <ryo.yonetani@sinicx.com>.

Proceedings of the 38th International Conference on Machine Learning, PMLR 139, 2021.

[1]Project page: https://omron-sinicx.github.io/neural-astar/.

The 38th International Conference on Machine Learning

ICML'21

ICML 2021
38th International Conference on Machine Learning
Online
July 18–24, 2021

https://proceedings.mlr.press/v139/yonetani21a.html

AIRI × Университет Сириус

Path Planning Using Neural A* Search

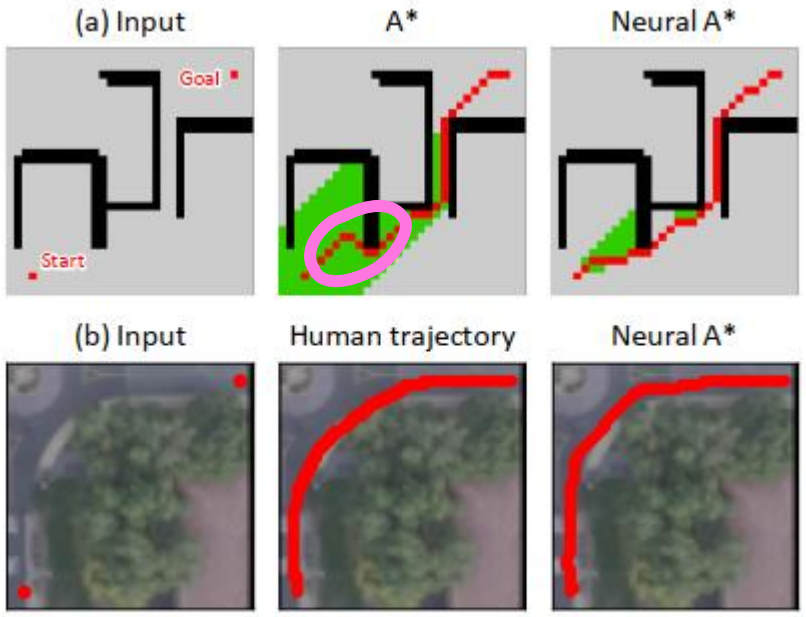


Figure 1. **Two Scenarios of Path Planning with Neural A*.** (a) Point-to-point shortest path search: finding a near-optimal path (red) with fewer node explorations (green) for an input map. (b) Path planning on raw image inputs: accurately predicting a human trajectory (red) on a natural image.

2 scenarios

- Costs (in graph) are known
  - Can obtain gt-paths by Dijsktra/A*/etc
  - 8-connected but uniform cost
    - cost(diag_move) = 1
- Cost are unknown
  - Planning on images
  - User-given gt-paths

Path Planning Using Neural A* Search :: Idea

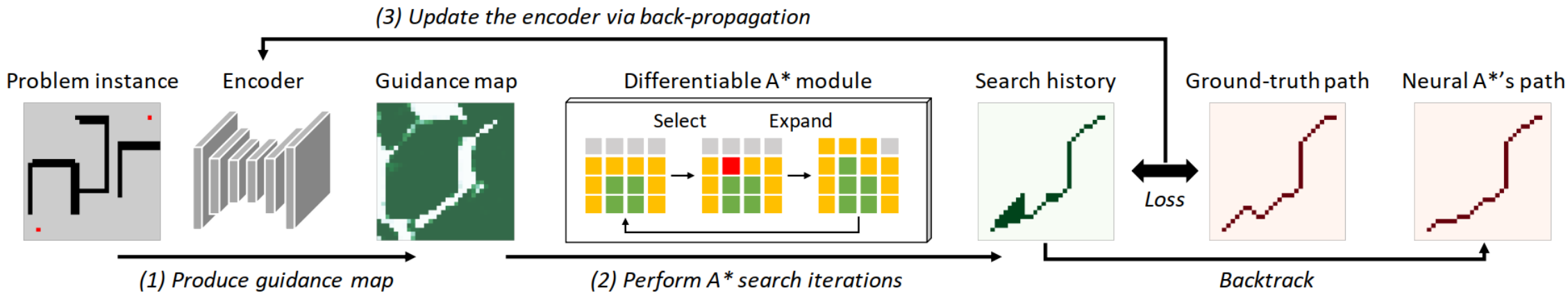


Figure 2. **Schematic Diagram of Neural A*.** (1) A path-planning problem instance is fed to the encoder to produce a guidance map. (2) The differentiable A* module performs a point-to-point shortest path search with the guidance map and outputs a search history and a resulting path. (3) A loss between the search history and the ground-truth path is back-propagated to train the encoder.

Core idea

- **Let's make A* differentiable**
  - To back-propagate through planner
  - To be able to learn end-to-end

Path Planning Using Neural A* Search :: Main Questions

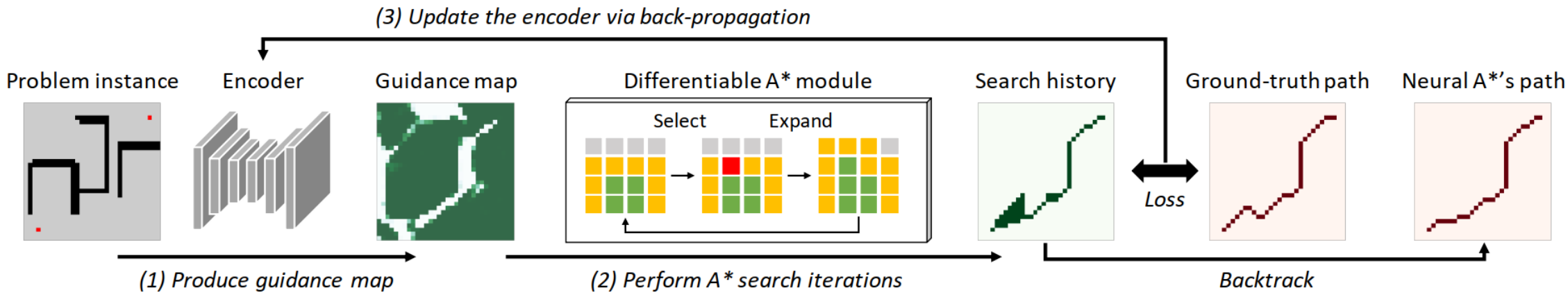


Figure 2. **Schematic Diagram of Neural A*.** (1) A path-planning problem instance is fed to the encoder to produce a guidance map. (2) The differentiable A* module performs a point-to-point shortest path search with the guidance map and outputs a search history and a resulting path. (3) A loss between the search history and the ground-truth path is back-propagated to train the encoder.

Core questions

- **What we are learning**
- **How can A* be differentiable**

Path Planning Using Neural A* Search :: Diff. A*

Algorithm 1 A* Search

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

Initialize $\mathcal{O} \leftarrow v_{\mathrm{s}}$, $\mathcal{C} \leftarrow \emptyset$, $\texttt{Parent}(v_{\mathrm{s}}) \leftarrow \emptyset$.
while $v_{\mathrm{g}} \notin \mathcal{C}$ **do**
    Select $v^* \in \mathcal{O}$ based on Eq. (1).
    Update $O \leftarrow O \setminus v^*$, $\mathcal{C} \leftarrow \mathcal{C} \cup v^*$.
    Extract $\mathcal{V}_{\mathrm{nbr}} \subset \mathcal{V}$ based on Eq. (2).
    for each $v' \in \mathcal{V}_{\mathrm{nbr}}$ **do**
        Update $\mathcal{O} \leftarrow \mathcal{O} \cup v'$, $\texttt{Parent}(v') \leftarrow v^*$.
    end for
end while
$P \leftarrow \texttt{Backtrack}(\texttt{Parent}, v_{\mathrm{g}})$.

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

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

Path Planning Using Neural A* Search :: Diff. A*

Algorithm 1 A* Search

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

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

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

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

v_s, v_g, v^* => V_s, V_g, V^*
one-hot matrices

Open, Close, Neighbors =>
O, C, V_{nbr}
binary matrices

G, H, X, Φ – real-valued matrices
X – vertices costs
known vs. unknown
Φ – guidance cost
that's what we learn

Path Planning Using Neural A* Search :: Diff. A* :: **Node Selection**

Algorithm 1 A* Search

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

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

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

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

v_s, v_g, v^* => V_s, V_g, V^*
one-hot matrices

Open, Close, Neighbors => O, C, V_{nbr}
binary matrices

G, H, X, Φ – real-valued matrices
X – vertices costs
Φ – guidance cost

$$V^* = \mathcal{I}_{\max}\left(\frac{\exp(-(G+H)/\tau) \odot O}{\langle \exp(-(G+H)/\tau), O \rangle}\right), \qquad (3)$$

Forward pass:

$I_{\max}$ = one-hot matrix for argmax

Backward pass:

$I_{\max}$ = Identity function

τ – temperature parameter (chosen empirically)

Path Planning Using Neural A* Search :: Diff. A* :: **Getting Neigh.**

Algorithm 1 A* Search

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

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

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

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

v_s, v_g, v^* => V_s, V_g, V^*
one-hot matrices

Open, Close, Neighbors => O, C, V_{nbr}
binary matrices

G, H, X, Φ – real-valued matrices
X – vertices costs
Φ – guidance cost

$$V_{\text{nbr}} = (V^* * K) \odot X \odot (\mathbb{1} - O) \odot (\mathbb{1} - C), \quad (4)$$

$$K = [[1,1,1]^\top, [1,0,1]^\top, [1,1,1]^\top]$$

Masking the nodes that are:
- Untraversable
- In Open
- In Closed

AIRI × Университет Сириус

Path Planning Using Neural A* Search :: Diff. A* :: **Getting Neigh.**

Algorithm 1 A* Search

Input: Graph $\mathcal{G}$, movement cost c, start v_s, and goal v_g
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: $P \leftarrow \texttt{Backtrack}(\texttt{Parent}, v_g)$.

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

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

v_s, v_g, v^* => V_s, V_g, V^*
one-hot matrices

Open, Close, Neighbors =>
O, C, V_{nbr}
binary matrices

G, H, X, Φ – real-valued matrices
X – vertices costs
Φ – guidance cost

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

For image-based pathfinding
(no X-masking)

Path Planning Using Neural A* Search :: Diff. A* :: **Update G**

Algorithm 1 A* Search

Input: Graph $\mathcal{G}$, movement cost c, start v_s, and goal v_g
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: $P \leftarrow \texttt{Backtrack}(\texttt{Parent}, v_g)$.

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

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

v_s, v_g, v^* => V_s, V_g, V^*
one-hot matrices

Open, Close, Neighbors =>
O, C, V_{nbr}
binary matrices

G, H, X, Φ – real-valued matrices
X – vertices costs
Φ – guidance cost

$$G \leftarrow G' \odot V_{\text{nbr}} + \min(G', G) \odot \bar{V}_{\text{nbr}} + G \odot (\mathbb{1} - V_{\text{nbr}} - \bar{V}_{\text{nbr}}), \qquad (5)$$

$$G' = \langle G, V^* \rangle \cdot \mathbb{1} + \Phi. \qquad (6)$$

G' = cost of v* + **guidance cost** of v'

If new node encountered, assign G'
If node in Open, assign min(G', G)
All other nodes, stay the same

Path Planning Using Neural A* Search :: **Diff. A***

Algorithm 1 A* Search

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

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

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

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

v_s, v_g, v^* => V_s, V_g, V^*
one-hot matrices

Open, Close, Neighbors => O, C, V_{nbr}
binary matrices

G, H, X, Φ – real-valued matrices
X – vertices costs
Φ – guidance cost

$$V^* = \mathcal{I}_{\max} \left(\frac{\exp(-(G+H)/\tau) \odot O}{\langle \exp(-(G+H)/\tau), O \rangle} \right), \qquad (3)$$

$$V_{\text{nbr}} = (V^* * K) \odot X \odot (\mathbb{1} - O) \odot (\mathbb{1} - C), \qquad (4)$$

$$\begin{aligned} G &\leftarrow G' \odot V_{\text{nbr}} + \min(G', G) \odot \bar{V}_{\text{nbr}} \\ &\quad + G \odot (\mathbb{1} - V_{\text{nbr}} - \bar{V}_{\text{nbr}}), \qquad (5) \\ G' &= \langle G, V^* \rangle \cdot \mathbb{1} + \Phi. \qquad (6) \end{aligned}$$

Differentiable Selection and Updating Neighbors

AIRI × Университет Сириус

Path Planning Using Neural A* Search :: Diff. A* :: Training

Loss = discrepancy between the gt-path and the Closed list

$$\mathcal{L} = \|C - \bar{P}\|_1 / |\mathcal{V}|. \qquad (7)$$

Encoder = UNet with the VGG16 backbone
Input = (X, Vs, Vg) = (map, start, goal)

Minibatch training

Detaching gradients

In practice, we disable the gradients of $\mathcal{O}$ in Eq. (3), $V_{\mathrm{nbr}}, \bar{V}_{\mathrm{nbr}}$ in Eq. (5), and G in Eq. (6) by detaching them from back-propagation chains. Doing so effectively informs the encoder of the importance of the guidance cost Φ in Eq. (6) for the node selection in Eq. (3), while simplifying recurrent back-propagation chains to stabilize the training process and reduce large memory consumption[2].

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**

Path Planning Using Neural A* Search :: Diff. A* :: Eval

Plenty of competitors

Meaningful dataset
still some question re. generation of the instances

Code

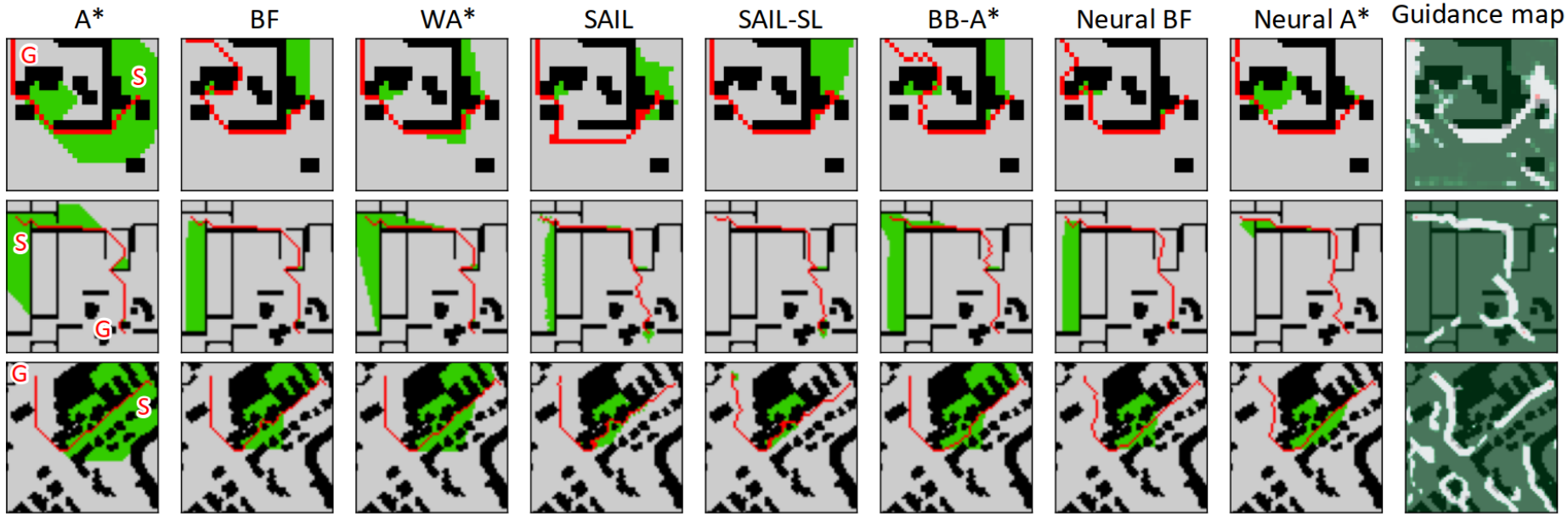


Figure 3. **Selected Path Planning Results.** Black pixels indicate obstacles. Start nodes (indicated by "S"), goal nodes (indicated by "G"), and found paths are annotated in red. Other explored nodes are colored in green. In the rightmost column, guidance maps are overlaid on the input maps where regions with lower costs are visualized in white. More results are presented in Sec. B of the supplementary material.

Path Planning Using Neural A* Search :: Diff. A* :: Eval

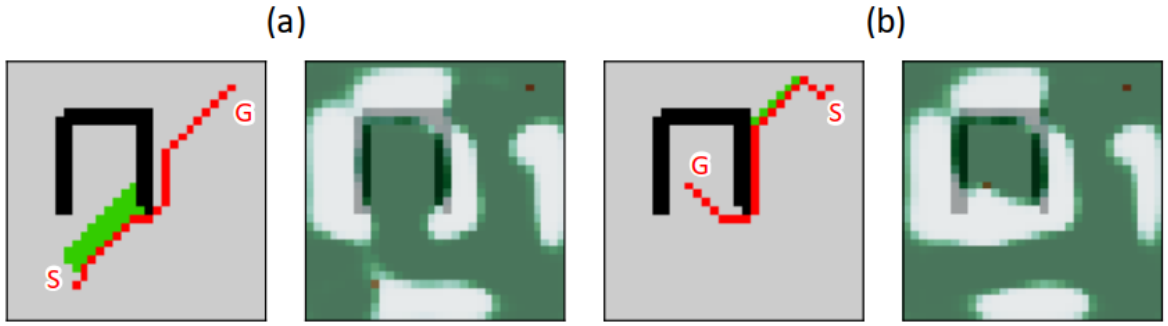


Figure 4. **Adaptive Encoding Results.** (a) The U-shaped dead-end obstacle is placed between the start and goal nodes; (b) The goal node is located inside the U-shaped dead end.

It Works

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)**

Path Planning Using Neural A* Search :: Diff. A* :: Eval

Table 3. **Quantitative Results on SDD.** Bootstrap means and 95% confidence bounds of the chamfer distance between predicted and ground-truth pedestrian trajectories.

	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)**

It Works

and for images as well

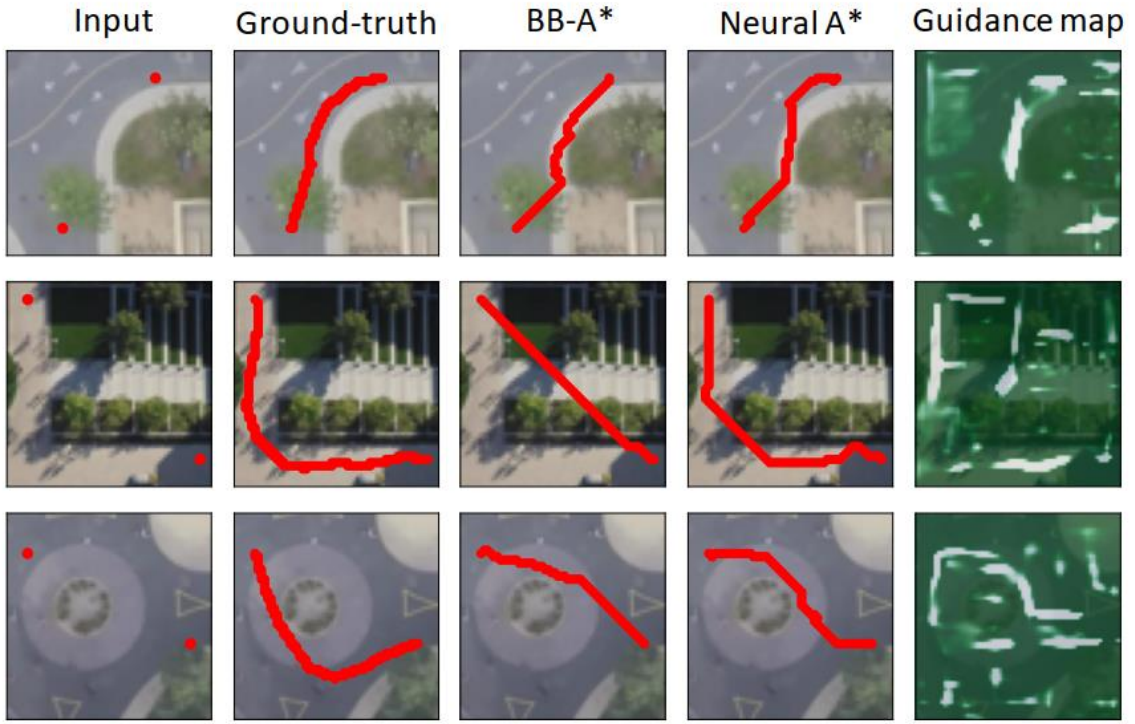


Figure 5. **Path Planning Examples on SDD.** Neural A* predicted paths similar to actual pedestrian trajectories.

Path Planning Using Neural A* Search

Path Planning using Neural A* Search

Ryo Yonetani [*1] Tatsunori Taniai [*1] Mohammadamin Barekatain [1 2] Mai Nishimura [1] Asako Kanezaki [3]

Abstract

We present *Neural A**, a novel data-driven search method for path planning problems. Despite the recent increasing attention to data-driven path planning, machine learning approaches to search-based planning are still challenging due to the discrete nature of search algorithms. In this work, we reformulate a canonical A* search algorithm to be differentiable and couple it with a convolutional encoder to form an end-to-end trainable neural network planner. Neural A* solves a path planning problem by encoding a problem instance to a guidance map and then performing the differentiable A* search with the guidance map. By learning to match the search results with ground-truth paths provided by experts, Neural A* can produce a path consistent with the ground truth accurately and efficiently. Our extensive experiments confirmed that Neural A* outperformed state-of-the-art data-driven planners in terms of the search optimality and efficiency trade-off. Furthermore, Neural A* successfully predicted realistic human trajectories by directly performing search-based planning on natural image inputs[1].

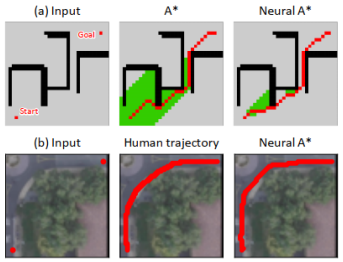


Figure 1. **Two Scenarios of Path Planning with Neural A*.** (a) Point-to-point shortest path search: finding a near-optimal path (red) with fewer node explorations (green) for an input map. (b) Path planning on raw image inputs: accurately predicting a human trajectory (red) on a natural image.

1. Introduction

Path planning refers to the problem of finding a valid low-cost path from start to goal points in an environmental map. *Search-based* planning, including the popular A* search (Hart et al., 1968), is a common approach to path planning problems and has been used in a wide range of applications such as autonomous vehicle navigation (Paden et al., 2016), robot arm manipulation (Smith et al., 2012), and game AI (Abd Algfoor et al., 2015). Compared to other planning approaches such as sampling-based planning (González et al., 2015) and reactive planning (Tamar et al., 2016; Lee et al., 2018), search-based planning is guaranteed to find a solution path, if one exists, by incrementally and extensively exploring the map.

Learning how to plan from expert demonstrations is gathering attention as promising extensions to classical path planners. Recent work has demonstrated major advantages of such data-driven path planning in two scenarios: (1) finding near-optimal paths more efficiently than classical heuristic planners in point-to-point shortest path search problems (Choudhury et al., 2018; Qureshi et al., 2019; Takahashi et al., 2019; Chen et al., 2020; Ichter et al., 2020) and (2) enabling path planning on raw image inputs (Tamar et al., 2016; Lee et al., 2018; Ichter & Pavone, 2019; Vlastelica et al., 2020), which is hard for classical planners unless semantic pixel-wise labeling of the environment is given.

In this study, we address both of these separately studied scenarios in a principled fashion, as highlighted in Fig. 1. In contrast to most of the existing data-driven methods that extend either sampling-based or reactive planning, we pursue a search-based approach to data-driven planning with the intrinsic advantage of guaranteed planning success.

Studies in this direction so far are largely limited due to

[*]Equal contribution [1]OMRON SINIC X, Tokyo, Japan [2]Now at DeepMind, London, UK. [3]Tokyo Institute of Technology, Tokyo, Japan. Correspondence to: Ryo Yonetani <ryo.yonetani@sinicx.com>.

Proceedings of the 38th International Conference on Machine Learning, PMLR 139, 2021.

[1]Project page: https://omron-sinicx.github.io/neural-astar/.

- Differentiable search module
- End-to-end learning
- Works on grids and on images
- Convincing experiments
- Code

- Limited setup for grids
  - 8-connected but uniform-cost move
- Some peculiarities re.training
  - Detaching gradients etc.
- Learning ‘weird’ heuristic

AIRI × Университет Сириус

Differentiation Of Blackbox Combinatorial Solvers

DIFFERENTIATION OF BLACKBOX COMBINATORIAL SOLVERS

Marin Vlastelica[1*], **Anselm Paulus**[1*], **Vít Musil**[2], **Georg Martius**[1], **Michal Rolínek**[1]
[1] Max-Planck-Institute for Intelligent Systems, Tübingen, Germany
[2] Università degli Studi di Firenze, Italy
{marin.vlastelica, anselm.paulus, georg.martius, michal.rolinek}@tuebingen.mpg.de
vit.musil@unifi.it

ABSTRACT

Achieving fusion of deep learning with combinatorial algorithms promises transformative changes to artificial intelligence. One possible approach is to introduce combinatorial building blocks into neural networks. Such end-to-end architectures have the potential to tackle combinatorial problems on raw input data such as ensuring global consistency in multi-object tracking or route planning on maps in robotics. In this work, we present a method that implements an efficient backward pass through blackbox implementations of combinatorial solvers with linear objective functions. We provide both theoretical and experimental backing. In particular, we incorporate the Gurobi MIP solver, Blossom V algorithm, and Dijkstra's algorithm into architectures that extract suitable features from raw inputs for the traveling salesman problem, the min-cost perfect matching problem and the shortest path problem. The code is available at

https://github.com/martius-lab/blackbox-backprop.

1 INTRODUCTION

The toolbox of popular methods in computer science currently sees a split into two major components. On the one hand, there are classical algorithmic techniques from discrete optimization – graph algorithms, SAT-solvers, integer programming solvers – often with heavily optimized implementations and theoretical guarantees on runtime and performance. On the other hand, there is the realm of deep learning allowing data-driven feature extraction as well as the flexible design of end-to-end architectures. The fusion of deep learning with combinatorial optimization is desirable both for foundational reasons – extending the reach of deep learning to data with large combinatorial complexity – and in practical applications. These often occur for example in computer vision problems that require solving a combinatorial sub-task on top of features extracted from raw input such as establishing global consistency in multi-object tracking from a sequence of frames.

The fundamental problem with constructing hybrid architectures is differentiability of the combinatorial components. State-of-the-art approaches pursue the following paradigm: introduce suitable approximations or modifications of the objective function or of a baseline algorithm that eventually yield a differentiable computation. The resulting algorithms are often sub-optimal in terms of runtime, performance and optimality guarantees when compared to their *unmodified* counterparts. While the sources of sub-optimality vary from example to example, there is a common theme: any differentiable algorithm in particular outputs continuous values and as such it solves a *relaxation* of the original problem. It is well-known in combinatorial optimization theory that even strong and practical convex relaxations induce lower bounds on the approximation ratio for large classes of problems (Raghavendra, 2008; Thapper & Živný, 2017) which makes them inherently sub-optimal. This inability to incorporate the best implementations of the best algorithms is unsatisfactory.

In this paper, we propose a method that, at the cost of one hyperparameter, implements a backward pass for a **blackbox implementation** of a combinatorial algorithm or a solver that optimizes a linear

*These authors contributed equally.

1

The 8th International Conference on Learning Representations

ICLR 2020

https://iclr.cc/virtual_2020/poster_BkevoJSYPB.html

https://openreview.net/pdf?id=BkevoJSYPB

AIRI × Университет Сириус

Differentiation Of Blackbox Combinatorial Solvers

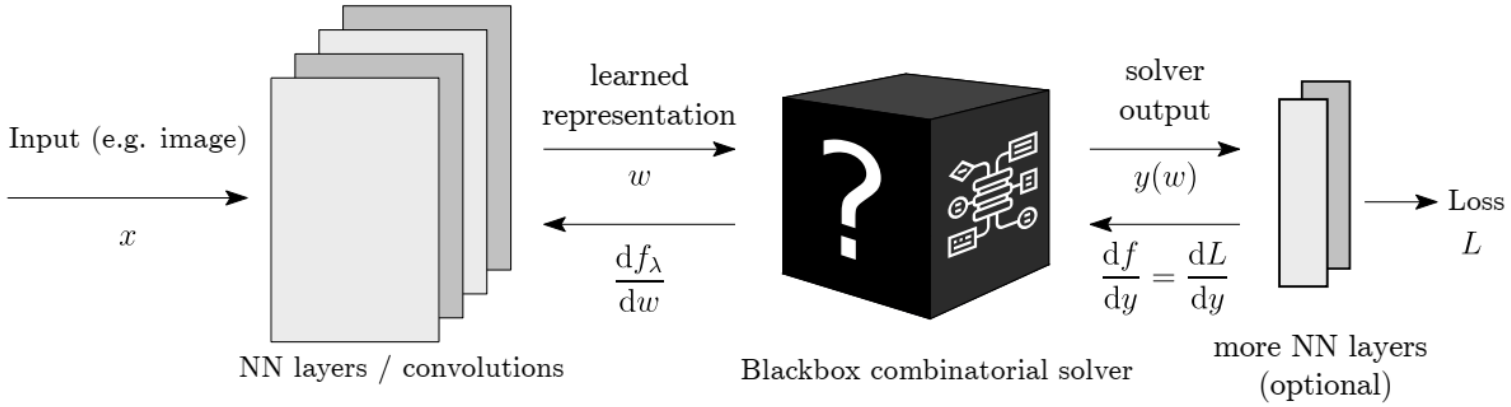
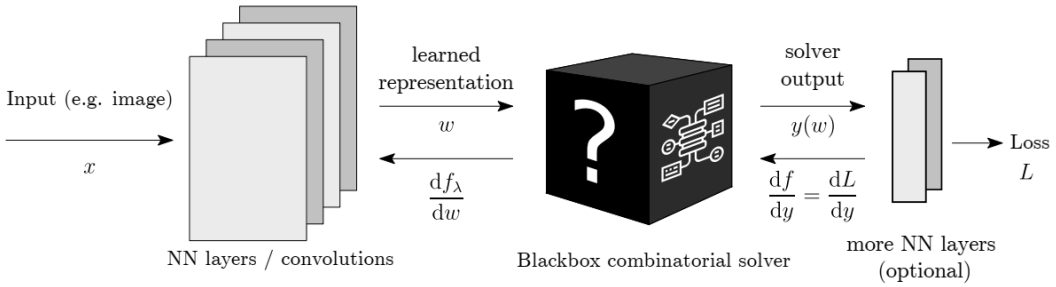
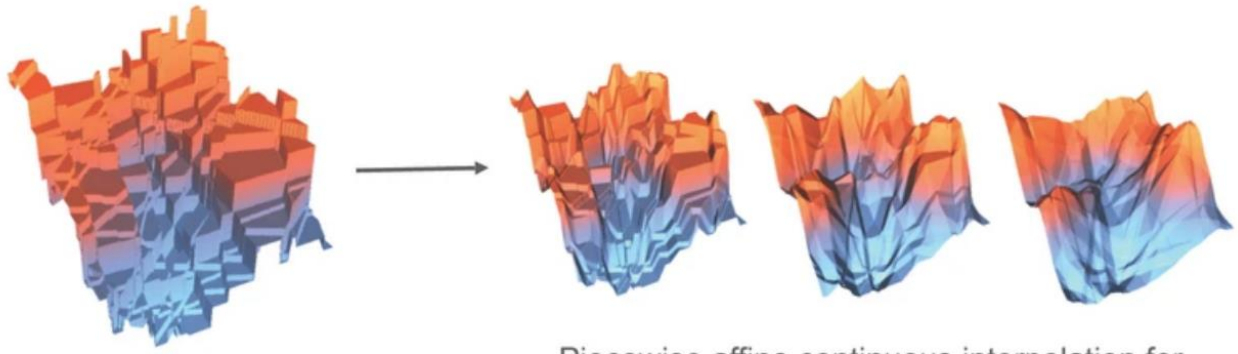


Figure 1: Architecture design enabled by Theorem 1. Blackbox combinatorial solver embedded into a neural network.

Plug-n-Play Combinatorial Solver Into The Differentiable Pipeline

Differentiation Of Blackbox Combinatorial Solvers





Piecewise affine continuous interpolation for hyperparameter values of λ = 0.2, 0.5, 1.0

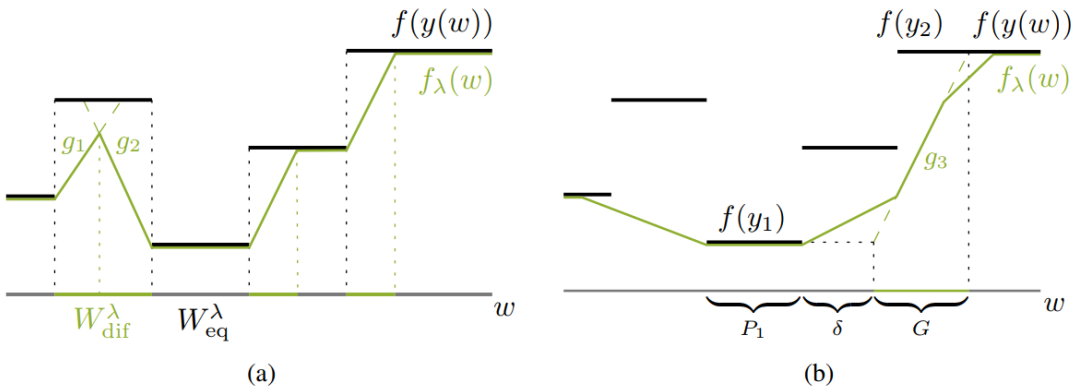


Figure 2: Continuous interpolation of a piecewise constant function. (a) f_λ for a small value of λ; the set W^λ_{eq} is still substantial and only two interpolators g_1 and g_2 are incomplete. Also, all interpolators are 0-interpolators. (b) f_λ for a high value of λ; most interpolators are incomplete and we also encounter a δ-interpolator g_3 (between y_1 and y_2) which attains the value $f(y_1)$ δ-away from the set P_1. Despite losing some local structure for high λ, the gradient of f_λ is still informative.

$$f(y) = L(\hat{y}) + \frac{\mathrm{d}L}{\mathrm{d}y}(\hat{y}) \cdot (y - \hat{y}) \quad \text{we have} \quad \frac{\mathrm{d}f\big(y(w)\big)}{\mathrm{d}w} = \frac{\mathrm{d}L}{\mathrm{d}w}$$

AIRI × Университет Сириус

Differentiation Of Blackbox Combinatorial Solvers

Algorithm 1 Forward and Backward Pass

function FORWARDPASS($\hat{w}$)
  $\hat{y} := \textbf{Solver}(\hat{w})$ // $\hat{y} = y(\hat{w})$
  save $\hat{w}$ and $\hat{y}$ for backward pass
  return $\hat{y}$

function BACKWARDPASS($\frac{\mathrm{d}L}{\mathrm{d}y}(\hat{y})$, λ)
  load $\hat{w}$ and $\hat{y}$ from forward pass
  $w' := \hat{w} + \lambda \cdot \frac{\mathrm{d}L}{\mathrm{d}y}(\hat{y})$
    // Calculate perturbed weights
  $y_\lambda := \textbf{Solver}(w')$
  return $\nabla_w f_\lambda(\hat{w}) := -\frac{1}{\lambda}\left[\hat{y} - y_\lambda\right]$
    // Gradient of continuous interpolation

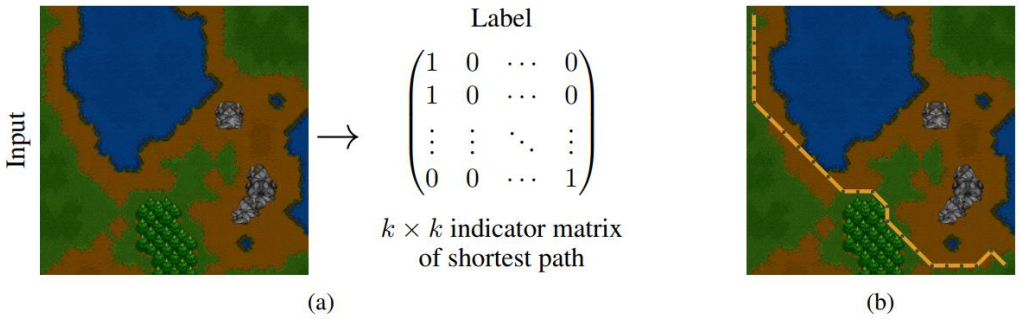


Figure 4: The SP(k) dataset. (a) Each input is a $k \times k$ grid of tiles corresponding to a Warcraft II terrain map, the respective label is a the matrix indicating the shortest path from top left to bottom right. (b) is a different map with correctly predicted shortest path.

Table 2: **Results for Warcraft shortest path.** Reported is the accuracy, i.e. percentage of paths with the optimal costs. Standard deviations are over five restarts.

	Embedding Dijkstra		ResNet18	
k	Train %	Test %	Train %	Test %
12	99.7 ± 0.0	96.0 ± 0.3	100.0 ± 0.0	23.0 ± 0.3
18	98.9 ± 0.2	94.4 ± 0.2	99.9 ± 0.0	0.7 ± 0.3
24	97.8 ± 0.2	94.4 ± 0.6	100.0 ± 0.0	0.0 ± 0.0
30	97.4 ± 0.1	94.0 ± 0.3	95.6 ± 0.5	0.0 ± 0.0

Differentiation Of Blackbox Combinatorial Solvers

DIFFERENTIATION OF BLACKBOX COMBINATORIAL SOLVERS

Marin Vlastelica[1]*, **Anselm Paulus**[1]*, **Vít Musil**[2], **Georg Martius**[1], **Michal Rolínek**[1]
[1] Max-Planck-Institute for Intelligent Systems, Tübingen, Germany
[2] Università degli Studi di Firenze, Italy
{marin.vlastelica, anselm.paulus, georg.martius, michal.rolinek}@tuebingen.mpg.de
vit.musil@unifi.it

ABSTRACT

Achieving fusion of deep learning with combinatorial algorithms promises transformative changes to artificial intelligence. One possible approach is to introduce combinatorial building blocks into neural networks. Such end-to-end architectures have the potential to tackle combinatorial problems on raw input data such as ensuring global consistency in multi-object tracking or route planning on maps in robotics. In this work, we present a method that implements an efficient backward pass through blackbox implementations of combinatorial solvers with linear objective functions. We provide both theoretical and experimental backing. In particular, we incorporate the Gurobi MIP solver, Blossom V algorithm, and Dijkstra's algorithm into architectures that extract suitable features from raw inputs for the traveling salesman problem, the min-cost perfect matching problem and the shortest path problem. The code is available at

https://github.com/martius-lab/blackbox-backprop.

1 INTRODUCTION

The toolbox of popular methods in computer science currently sees a split into two major components. On the one hand, there are classical algorithmic techniques from discrete optimization – graph algorithms, SAT-solvers, integer programming solvers – often with heavily optimized implementations and theoretical guarantees on runtime and performance. On the other hand, there is the realm of deep learning allowing data-driven feature extraction as well as the flexible design of end-to-end architectures. The fusion of deep learning with combinatorial optimization is desirable both for foundational reasons – extending the reach of deep learning to data with large combinatorial complexity – and in practical applications. These often occur for example in computer vision problems that require solving a combinatorial sub-task on top of features extracted from raw input such as establishing global consistency in multi-object tracking from a sequence of frames.

The fundamental problem with constructing hybrid architectures is differentiability of the combinatorial components. State-of-the-art approaches pursue the following paradigm: introduce suitable approximations or modifications of the objective function or of a baseline algorithm that eventually yield a differentiable computation. The resulting algorithms are often sub-optimal in terms of runtime, performance and optimality guarantees when compared to their *unmodified* counterparts. While the sources of sub-optimality vary from example to example, there is a common theme: any differentiable algorithm in particular outputs continuous values and as such it solves a *relaxation* of the original problem. It is well-known in combinatorial optimization theory that even strong and practical convex relaxations induce lower bounds on the approximation ratio for large classes of problems (Raghavendra, 2008; Thapper & Živný, 2017) which makes them inherently sub-optimal. This inability to incorporate the best implementations of the best algorithms is unsatisfactory.

In this paper, we propose a method that, at the cost of one hyperparameter, implements a backward pass for a **blackbox implementation** of a combinatorial algorithm or a solver that optimizes a linear

*These authors contributed equally.

Powerful general idea

Need to think more about this…

AIRI × Университет Сириус

Изображение сгенерировано моделью ruDALL-E от Сбера по запросу "Супрематический логотип AIRI"

AIRI

Artificial Intelligence Research Institute

airi.net

airi_research_institute

AIRI Institute

AIRI Institute

AIRI_inst

artificial-intelligence-research-institute